FORMAL SPECIFICATION IN Z NOTATIONS FOR SECURE QUERY PROCESSING IN DEDUCTIVE DATABASE SYSTEMS

Ketul Rajanikant Patel
B.E., Hemchandracharya North Gujarat University, Gujarat, India, 2007

PROJECT

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

at

CALIFORNIA STATE UNIVERSITY, SACRAMENTO

FALL
2010
FORMAL SPECIFICATION IN Z NOTATIONS FOR SECURE QUERY PROCESSING IN
DEDUCTIVE DATABASE SYSTEMS

A Project

by

Ketul Rajanikant Patel

Approved by:

__________________________, Committee Chair
Cui Zhang, Ph.D

__________________________, Second Reader
William Mitchell, Ph.D

____________________________
Date
Student: Ketul Rajanikant Patel

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

__________________________________________, Graduate Coordinator
Nikrouz Faroughi, Ph.D  Date

Department of Computer Science
Abstract

of

FORMAL SPECIFICATION IN Z NOTATIONS FOR SECURE QUERY PROCESSING IN DEDUCTIVE DATABASE SYSTEMS

by

Ketul Rajanikant Patel

Security is an important non-functional requirement for nearly any software system. There should not be any room for security errors in software systems like databases and other critical systems, where data is stored and should be retrieved through authentication. Informal specifications are easy to write and read, but they suffer from contradiction and ambiguities. Formal Specifications are the most efficient and flawless way of defining security constraints.

This project specifies secure query processing in deductive database systems using the well know formal language Z. The system state and invariants of deductive database systems are formally modeled. The security constraints of table manipulation operations and other administrative operations are formally specified in terms of the preconditions and postconditions of these operations.

______________________, Committee Chair
Cui Zhang, Ph.D

______________________
Date
I would like to express gratitude to my project advisor Dr. Cui Zhang for her direction and supporting me to complete this project. I am thankful to my second reader Dr. William Mitchell for his assistance. I am utterly grateful to my parents Yashodhara and Rajanikant Patel for their love, moral support and encouragement.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>vi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td><strong>1. INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>2. BACKGROUND AND RELATED WORKS</strong></td>
<td>3</td>
</tr>
<tr>
<td><strong>3. SYSTEM ARCHITECTURE</strong></td>
<td>7</td>
</tr>
<tr>
<td><strong>4. FORMAL SPECIFICATIONS</strong></td>
<td>9</td>
</tr>
<tr>
<td>4.1 System State Specification</td>
<td>9</td>
</tr>
<tr>
<td>4.2 User Login Specification</td>
<td>13</td>
</tr>
<tr>
<td>4.3 Authorization Specification</td>
<td>16</td>
</tr>
<tr>
<td>4.4 Specification for Adding Table to Tablelist</td>
<td>18</td>
</tr>
<tr>
<td>4.5 Specification for Giving Table Rights to Users</td>
<td>20</td>
</tr>
<tr>
<td>4.6 Specification for Adding Column to Table</td>
<td>22</td>
</tr>
<tr>
<td>4.7 Specification for Deleting Column from Table</td>
<td>24</td>
</tr>
<tr>
<td>4.8 Specification for Deleting Table from Matrix</td>
<td>26</td>
</tr>
<tr>
<td><strong>5. SPECIFICATIONS FOR TABLE MANIPULATION OPERATIONS</strong></td>
<td>28</td>
</tr>
<tr>
<td>5.1 Specification for Read Table Operation</td>
<td>28</td>
</tr>
<tr>
<td>5.2 Specification for Edit/Update Table Operation</td>
<td>29</td>
</tr>
<tr>
<td>5.3 Specification for Insert Operation</td>
<td>32</td>
</tr>
<tr>
<td>5.4 Specification for Delete Operation</td>
<td>34</td>
</tr>
<tr>
<td><strong>6. SPECIFICATIONS FOR PROVIDING COLUMN LEVEL SECURITY</strong></td>
<td>36</td>
</tr>
<tr>
<td>6.1 System State specification</td>
<td>36</td>
</tr>
<tr>
<td>6.2 Specification for Giving Access Rights to User</td>
<td>38</td>
</tr>
<tr>
<td>6.3 Specification for Read Table Operation</td>
<td>40</td>
</tr>
<tr>
<td>6.4 Specification for Edit/Update Table Operation</td>
<td>42</td>
</tr>
<tr>
<td>6.5 Specification for Insert Operation</td>
<td>44</td>
</tr>
<tr>
<td>6.6 Specification for Delete Operation</td>
<td>46</td>
</tr>
</tbody>
</table>
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Figure 1. Basic System Architecture for Deductive Database System</td>
<td>7</td>
</tr>
<tr>
<td>2.</td>
<td>Figure 2. Basic System State</td>
<td>9</td>
</tr>
<tr>
<td>3.</td>
<td>Figure 3. System State with User Authentication</td>
<td>10</td>
</tr>
<tr>
<td>4.</td>
<td>Figure 4. System State with Both User Authentication and Access Control Matrix</td>
<td>11</td>
</tr>
<tr>
<td>5.</td>
<td>Figure 5. System State with Elements for Giving Table Access Rights</td>
<td>11</td>
</tr>
<tr>
<td>6.</td>
<td>Figure 6. System State Specification with User Access Rights</td>
<td>12</td>
</tr>
<tr>
<td>7.</td>
<td>Figure 7. Complete System State Specification</td>
<td>13</td>
</tr>
<tr>
<td>8.</td>
<td>Figure 8. Add User to Authe</td>
<td>14</td>
</tr>
<tr>
<td>9.</td>
<td>Figure 9. Specification for adding User to Authentication List</td>
<td>15</td>
</tr>
<tr>
<td>10.</td>
<td>Figure 10. Specification for User Login</td>
<td>16</td>
</tr>
<tr>
<td>11.</td>
<td>Figure 11. Specification for Giving Rights for Access Control Matrix</td>
<td>17</td>
</tr>
<tr>
<td>12.</td>
<td>Figure 12. Specification for Adding Table to Tablist</td>
<td>19</td>
</tr>
<tr>
<td>13.</td>
<td>Figure 13. Specification for Giving Table Rights to User</td>
<td>21</td>
</tr>
<tr>
<td>14.</td>
<td>Figure 14. Specification for Adding Column to Table</td>
<td>23</td>
</tr>
<tr>
<td>15.</td>
<td>Figure 15. Specification for Deleting Column from Table</td>
<td>25</td>
</tr>
<tr>
<td>16.</td>
<td>Figure 16. Specification for Deleting Table from Tablist</td>
<td>27</td>
</tr>
<tr>
<td>17.</td>
<td>Figure 17. Specification for Authorized Table Read Operation</td>
<td>29</td>
</tr>
<tr>
<td>18.</td>
<td>Figure 18. Specification for Authorized Table Edit/Update Operation</td>
<td>31</td>
</tr>
<tr>
<td>19.</td>
<td>Figure 19. Specification for Secure Insert Operation</td>
<td>33</td>
</tr>
<tr>
<td>20.</td>
<td>Figure 20. Specification for Secure Delete Operation</td>
<td>35</td>
</tr>
<tr>
<td>21.</td>
<td>Figure 21. Updated Sys_State Specification</td>
<td>37</td>
</tr>
<tr>
<td>22.</td>
<td>Figure 22. Specification for Giving Table Rights to User</td>
<td>39</td>
</tr>
<tr>
<td>23.</td>
<td>Figure 23. Specification for Authorized Table Read Operation</td>
<td>41</td>
</tr>
<tr>
<td>24.</td>
<td>Figure 24. Specification for Authorized Table Edit/Update Operation</td>
<td>43</td>
</tr>
<tr>
<td>25.</td>
<td>Figure 25. Specification for Secure Insert Operation</td>
<td>45</td>
</tr>
<tr>
<td>26.</td>
<td>Figure 26. Specification for Secure Delete Operation</td>
<td>47</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

Security is an important non-functional requirement for any software system. There should not be any room for flaws in software systems, in which security is the primary concern, e.g., database systems where important data are stored and must be provided only to authenticated and authorized users. According to the research done by the Software Engineering Institute (SEI) at Carnegie Mellon University, the reason for security violations and defects in software systems is imprecise and ambiguous security specifications [1]. The research was focused on effective software development process. Security requirements are typically specified informally using natural languages. This is the easiest way to define security specifications but it often ends up with contradictions and ambiguities, which may lead to violation of security of software system. The only unambiguous way to define security constraints is to use formal languages like Z notation [2], VDM [3].

This project specifies secure query processing for deductive databases. Concurrency control is also specified in this report. The specification languages used in this project is Z notation, which is one of the most famous specification languages. To define security policy Access Control Matrix (ACM) is used. The matrix has entries containing the three entities subjects, objects and permissions i.e., add, delete, update etc. A subject can access the object with valid permission. In this project, objects are tables and users are subjects. Whereas permissions are to add, update and delete tables. Several special permissions can be given for accessing the matrix.

This project is carried out in two phases. In the first phase, table level security specifications are defined. A user can have access to a table with valid permission if there is an entry stating permission for accessing the table for that user. In the second phase, specifications for column level security are defined. A user can only access the columns of a table for which the user has valid access rights. For that, the access control matrix used in the first phase is extended.
Formal methods are the most efficient way to define specifications because of the preciseness and accurate interpretation of formal specifications without any ambiguity and contradiction. In addition, formal methods are easier to understand than informal methods, and can facilitate the detection and elimination of defects in early analysis period. Systems without or with less defects are more maintainable and cost effective.

Chapter 2 provides the background and related work of this project. Chapter 3 discusses the abstract system architecture of deductive database systems. The architecture is the first step towards writing specifications for secure query processing for deductive database systems. A valid system state is the backbone for secure and successful operations in the system. Chapter 4 discusses about the system state for deductive database systems. Once specifications for secure system state are provided, specifications for various operations in the system are defined. Specifications for table manipulation operations in deductive database system are discussed in chapter 5. Chapter 4 and chapter 5 discuss the first phase of this project. Chapter 6 discusses about the specifications for secure query processing for column level security. Chapter 7 concludes the report and mentions future enhancements.
Chapter 2

BACKGROUND AND RELATED WORKS

Zaniolo [4] gives brief introduction of development of deductive databases. Horn clause, which is the base of deductive databases, is discussed in this work. This work gives primary knowledge of working principles of deductive databases. This work also describes the development and basic characteristics of the LDL deductive database and language. The user interface for LDL was developed in Java.

Arni et al. [5] discusses design and development of the successor of deductive database system LDL [4]. LDL++ was an attempt to improve the user interface of the LDL deductive database system. An attempt for making query writing easier for users was made in this work. Overall, LDL++ has better usability than its predecessor LDL. In addition, a new version of LDL language was developed for this project and it was given the same name as the database, LDL++. Same as its predecessor, user interface was developed in Java.

Another deductive database system CORAL [6] is also one of the available successful deductive database systems developed so far. The system is claimed to be more efficient and have powerful language support than any other deductive databases. The query language used in this database is CORAL. The user interface for this database is developed in C++. Users have flexibility to write queries in C++ as well as in CORAL.

The above mentioned works represent the design of various deductive database systems developed successfully so far. The most important characteristic any software system needs to perform well is to stay in secure state before and after performing operations. In deductive databases or any database, execution under secure state means that a secure query processing or execution will not violate any constraints, rules or conditions. Research has been done to provide various techniques for secure query processing and to provide security through unauthorized accesses.
Several deductive database query answering techniques are so efficient that they can be implemented for other types of databases too. Yoon et al. explain this in [7], where a deductive query processing technique is implemented over an object-oriented database system. The secure query processing is done in four steps. One major advantage of this strategy is that, query can be executed without accessing facts or data stored in database. This can be considered as a distinctive characteristic, and will make database systems execute a query faster.

Some techniques for providing security to deductive databases have distinctive strategies than other methods. In some techniques database directly tells unauthorized users that, they do not have valid rights to access that information. In deductive databases, some stored information can be derived from information, for which users have valid access rights. To prevent this, we can make databases lie to unauthorized users. Bonatti et al. [5] explain several methods that make deductive database systems to lie to users. Sometimes it is harmful for a database to tell truth to unauthorized users. According to this work, classified information can be stored as a secret. Now, to protect secret information from unauthorized uses, it is suggested in this work that, the details about information being a secret, should be kept a secret too.

Thuraisingham [9] discusses security in MLS (Multi-Level Secure) database systems. In MLS database systems, users have different security clearance levels and data stored in the database has different security levels. Users can access only the information having appropriate security level. According to [9], one of the major issues in security for deductive databases is imprecise and improper inference rules. In this work, various methods for providing secure query processing for various types of MLS databases are explained. A security policy is then developed and implemented on various databases. Two stages of query processing are explained. The first step is query modification. In query modification, query analyzer modifies the query from unauthorized user, trying to access classified information from database. The second step is handling of logical implications.
In Pernul et al.’s work [10], initially security specifications and semantics are written after a careful study of security requirements of a MLS deductive database. Later, the specifications are implemented on a deductive databases system. Security specifications are written in SCL (Security Constraints Language). The specifications are tested on LDL deductive database system. An attempt is made to develop a prototype for standard design development process for deductive database systems. This prototype is helpful for deductive database designers to design secure deductive databases. Bell-LaPedula security model is used for security specifications in this work.

Martin et al. [11] explain integrity constraint checking methods for temporal deductive databases. According to the work, the temporal deductive database is “a deductive database system that supports some aspect of time when the fact is true”. Consistency is the primary requirement for temporal deductive databases whenever it performs any transaction. Transaction is a set of operations performed on database system resources. Any unauthorized or any transaction that leads database to inconsistent state or violates any constraint must not be performed. In Celma et al.’s work [12], different integrity constraint checking methods are compared to decide the better and most efficient ones. First, various methods are classified into two groups: methods that need data access and methods those do not need data access for integrity checking. Later, methods are compared according to the number of data accesses for processing a query.

RBAC (Role Based Access Control) in terms of normal clause logic is implemented by Barker in [13]. In this work, specifications for providing security from unauthorized data accesses and data update requests are specified. In the work of Smith and Keighley [14] formal specifications for secure query transaction in MLS database systems are discussed. In this work, specifications are written in Z notation. Specifications for the SWORD relational database are written in this work. SWORD is a concurrent database system, where more than one user can access database concurrently. For concurrent operations, a concurrency control is needed for each system, and data locking is a common practice. In this work specifications for lockless concurrency control are provided, which leads to read/write concurrency. The term lockless
concurrency is explained with an example in this paper. In database systems, a successful commit operation ensures that all database state changes specified by a transaction are then eventually applied exactly once. Specifications for committing a transaction are also provided.

This project report discusses about protecting deductive database system resources from unauthorized accesses. Most of the works mentioned above discuss the same issue. However, unlike all above mentioned works, in this work pre-conditions and post-conditions for each table manipulation and other administrative operations are specified. The notations used in this project are more straightforward to write and understand than the notations used in above mentioned works.
Chapter 3
SYSTEM ARCHITECTURE

Before formal specifications are presented, the basic system architecture of deductive database systems needs to be discussed. The architecture under discussion is an abstract architecture. The main part of the system, which the discussion is going to focus on, is the part that consists of a Rule Manager and a Rule Database. This is the part, which differentiates deductive databases from relational databases.

![Figure 1. Basic System Architecture for Deductive Database System](image)

As shown in Figure 1 the first component is a User Interface, through which user can interact with the database. Interface can be created with the help of any programming language. Interface of CORAL database was created in C/C++, whereas in LDL and its successor interface was created using Java. The Authentication Manager is responsible for allowing only authenticated users to enter into the system by checking the presence of user in user role database.

After the process of verifying authenticity of users, performed by the Authentication Manager, Concurrency Controller is the next to function, which gives token to user transactions to perform operations. In this system, each resource (i.e. each table) has one token and users have to get the token to
perform operations on respective resource. Once a user is done with the operations on a specific resource, that token is released.

After the Concurrency Controller, the Rule Manager is responsible for allowing only authorized users to perform valid operations on database. Whenever a user tries to execute a query, Rule Manager checks for valid authorization of the user to perform that operation on the table. This means that users are required to have valid rights to perform operations.

Data Manager is responsible for delivering data to a user in response to a valid request. Upon valid request approved by the Rule Manager, Data Manager looks for data in main memory (i.e. RAM). If data is not available in main memory, data is then brought to main memory from secondary storage (i.e. Hard Drive). Memory management part is not the focus of this report.
Chapter 4

FORMAL SPECIFICATIONS

This chapter provides the specifications in Z notation for deductive database systems. In process of writing formal specifications, first step is to define system state. System state contains all conditions required to be satisfied before any database operation. System can start operating once all conditions are satisfied. This work is divided into two phases. In first phase, specifications for table level security are discussed. Whereas in second phase, specifications for column level security are discussed. In this chapter, system state specification is discussed. In later sections, formal specifications for authorization are given.

4.1 System State Specification

This section discusses step by step implementation system state specification. Full formal specifications are given in Appendix A. The system state specifications are based on three elements, users, tables and access rights. Three basic elements in the system state are *users*, *Tablist* and *mode*. Where *users* are authorized users and *Tablist* is set of tables present in the system. Users’ valid rights of operations are mode.

\[
\text{Sys\_State}
\]

\[
\begin{align*}
\text{users: } & \mathbb{P} \text{USER} \\
\text{Tablist: } & \mathbb{P} \text{TABLE} \\
\text{cols: } & \mathbb{P} \text{COLUMN} \\
\text{recs: } & \mathbb{P} \text{RECORD} \\
\text{mode: } & \mathbb{P} \text{MODE}
\end{align*}
\]

Figure 2. Basic System State

Figure 2 shows the specification for system state. The specification is for the simplest form of the basic system. As mentioned before, this system consists of three main elements: *users*, *Tablist* and *mode*. Next
system specification includes elements for user’s authentication. USER, TABLE, MODE, COLUMN, RECORD, MODE AND AUTHENTICATE are predefined sets or types.

\[
\begin{align*}
\text{Sys State} \\
\text{users: } \mathbb{P} \text{USER} \\
\text{Tablist: } \mathbb{P} \text{TABLE} \\
\text{cols: } \mathbb{P} \text{COLUMN} \\
\text{recs: } \mathbb{P} \text{RECORD} \\
\text{mode: } \mathbb{P} \text{MODE} \\
\text{Authenticate: } \mathbb{P} \text{AUTHENTICATE} \\
\text{TabCols: cols } \rightarrow \mathbb{P} \text{recs} \\
\text{TabDetail: Tablist } \rightarrow \text{ TabCols} \\
\text{UserAutho: users } \rightarrow \mathbb{P} \text{Authenticate}
\end{align*}
\]

\[
\text{dom UserAutho } \subseteq \text{ users}
\]

Figure 3. System State with User Authentication

In Figure 3 \textit{UserAutho} is a partial function from user to user’s authentication. Users who are authorized to access database are stored here. Each user can have more than one role. That means, one user can have more than one authentication. Once a user is added to the system, the second step is to add that user to Access Control Matrix. This matrix stores information about the users and their rights to access database tables. Figure 4 shows the specification including elements for the Access Control Matrix. After adding the access control matrix to the system state, the next step is to represent users’ rights to manipulate tables and to alter other users’ rights. A special right to alter Access Control Matrix can also be given to the users.
Sys_State

users: ℙ USER
Tablist: ℙ TABLE
mode: ℙ MODE
cols: ℙ COLUMN
recs: ℙ RECORD
mode: ℙ MODE
matautho: ℙ MatAutho
Authenticate: ℙ AUTHENTICATE
TabCols: cols → ℙ recs
TabDetail: Tablist → TabCols
UserAutho: users → ℙ Authenticate
Matrix: users → ℙ matautho

dom UserAutho ⊆ users
dom Matrix ⊆ users
dom TabDetails ⊆ Tablist
dom TabCols ⊆ cols

Figure 4. System State with Both User Authentication and Access Control Matrix

Sys_State

users: ℙ USER
Tablist: ℙ TABLE
mode: ℙ MODE
cols: ℙ COLUMN
recs: ℙ RECORD
mode: ℙ MODE
matautho: ℙ MatAutho
Authenticate: ℙ AUTHENTICATE
TabCols: cols → ℙ recs
TabDetail: Tablist → TabCols
UserAutho: users → ℙ Authenticate
Matrix: users → ℙ matautho
AccTabAutho: Tablist → ℙ mode
AccTab: users → AccTabAutho

dom UserAutho ⊆ users
dom Matrix ⊆ users
dom TabDetails ⊆ Tablist
dom TabCols ⊆ cols
dom AccTab ⊆ users
dom AccTabAutho ⊆ Tablist

Figure 5. System State with Elements for Giving Table Access Rights
Figure 5 shows the specification including elements for table access rights for users to access control matrix. *AccTabAutho* is a partial function from users to another partial function. This function has a table as domain and a set of access rights as range. Therefore, *AccTabAutho* indicates that a user has access to a set of tables and for each table user has one or more access rights. After tables, there is a need for adding a facility for giving user rights and updating rights. Figure 6 shows the specification with elements for maintaining user rights in the access control matrix of database. Same as for table, Figure 6 includes specification for maintaining user access rights in access control matrix. Figure 7 shows a complete system state specification for the deductive database system.

```
Sys_State
users: ℙ USER
Tablist: ℙ TABLE
mode: ℙ MODE
cols: ℙ COLUMN
recs: ℙ RECORD
mode: ℙ MODE
Authenticate: ℙ AUTHENTICATE
userautho: ℙ UAutho
TabCols: cols ⇸ recs
TabDetail: Tablist ⇸ TabCols
Matrix: users ⇸ ℙ MatAutho
AccTabAutho: Tablist ⇸ ℙ mode
AccTab: users ⇸ AccTabAutho
AccUserAutho: users ⇸ ℙ userautho
UserAutho: users ⇸ ℙ Authenticate
AccUser: users ⇸ AccUserAutho

dom UserAutho ⊆ users
dom Matrix ⊆ users
dom TabDetails ⊆ Tablist
dom TabCols ⊆ cols
dom AccTab ⊆ users
dom AccTabAutho ⊆ Tablist
dom AccUser ⊆ users
dom AccUserAutho ⊆ users
```

Figure 6. System State Specification with User Access Rights
4.2 User Login Specification

Once a system is ready to execute, users can access the system and use it by entering into the system with valid authentication. Once a user has entered his/her credentials, the system will check for valid authentication. If there is a valid match, it will allow a user to access system resources and let a user perform operations with valid rights. According to the system architecture given in Chapter 3, the User Role Manager performs this operation. Users’ login information is stored in User Role Database. The User Role Manager will look for valid match from the User Role Database. But first, users need to be added to the User Role Database. Therefore, one more list which contains users’ identity needs to be added to the
system state specification. This list is called Authe. Authe represents a set of users. Figure 7 shows the system state specification with added User Role list.

\[ \text{AddLoginSuccess} \]
\[ \Delta \text{Sys\_State} \]
\[ u? : \text{USER}; \ m! : \text{Message} \]

\[ u? \notin \text{Authe} \]
\[ \text{Authe}' = \text{Authe} \cup \{u?\} \]
\[ m! = \text{User Added Successfully} \]

\[ \text{LoginExists} \]
\[ \Xi \text{Sys\_State} \]
\[ u? : \text{USER}; \ m! : \text{Message} \]

\[ u? \in \text{Authe} \]
\[ m! = \text{User Already Added} \]

\[ \text{AddLogin} \equiv \text{AddLoginSuccess} \lor \text{LoginExists} \]

Figure 8. Add User to Authe

Figure 8 shows the specification of the operation AddLogin for adding user login to the user list. In AddLoginSuccess, the symbol “\(\Delta\)” before Sys\_State indicates that a successful operation will make changes to the Sys\_State when a user is successfully added to the system. Whereas, LoginExists will display error if a user with same authentication has already been added to the list. The symbol “\(\Xi\)” before Sys\_State means that this operation will not make any change to the Sys\_State. Figure 8 also shows that there are only two possible cases for operation AddLogin. Once a user is added to the list that does not mean that user can enter to the system. After adding a user to the userlist Authe, user’s authentications need to be added. UserAutho is the list where user’s authorization information is added.
Figure 9. Specification for adding User to Authentication List

Figure 9 shows the specification of the operation AuthenticationAdd for adding user’s authorization to the authenticated users list. Here, one user can have more than one authentication. Now a user can enter into the system by providing valid authentications. According to the Figure 9 AuthenticationAdd has three possible cases.
Figure 10. Specification for User Login

\[
\begin{align*}
\text{LoginSuccess} & \iff \Xi \text{ Sys State} \\
& \quad u? : \text{users}; a? : \text{Authenticated}; m! : \text{Message} \\
& \quad a? \in \text{UserAutho}(u?) \\
& \quad m! = \text{Login Successful}
\end{align*}
\]

\[
\begin{align*}
\text{LoginFailed} & \iff \Xi \text{ Sys State} \\
& \quad u? : \text{users}; a? : \text{Authenticated}; m! : \text{Message} \\
& \quad a? \notin \text{UserAutho}(u?) \\
& \quad m! = \text{Login Failed, Invalid Authentications}
\end{align*}
\]

\[
\text{UserLogin} \equiv \text{LoginSuccess} \lor \text{LoginFailed}
\]

Figure 10 shows the specification for user login with valid authentications. \text{LoginSuccess} is the case where the system will respond positively that a user has valid authentications for entering into the system. \text{LoginFailed} is for the situation where a user does not have valid authentications. \text{UserLogin} has only two possible cases.

4.3 Authorization Specification

Once a user is successfully added to the authenticated users, next task is to give different rights for accessing or manipulating resources. These resources include tables, other users and Access Control Matrix. A user having access rights to Access Control Matrix can update structure of tables, other users and Access Control Matrix itself. Figure 11 shows the specification for giving rights to the Access Control Matrix.
MatrixAutho

Δ Sys_State
u?: users; newu?: users; rights?: ℙmatautho; m!: Message

newu? ∈ Auth
newu? ∈ dom UserAutho
u?): ∈ Auth
u?): ∈ dom UserAutho
Mat ∈ Matrix(u?)
Matrix ′(newu?) = Matrix (newu?) ∪ {rights?}
m! = Matrix Rights Added For User

MatrixAuthoNewUserNotFound

Ξ Sys_State
u?: users; newu?: users; rights?: ℙmatautho; m!: Message

newu? /∈ Auth ∨ newu? /∈ dom UserAutho
m! = New User Not In a List

MatrixAuthoUserNotFound

Ξ Sys_State
u?: users; newu?: users; rights?: ℙmatautho; m!: Message

u?): /∈ Auth ∨ u?): /∈ dom UserAutho
m! = User Not Found

MatrixAuthoInvalidAccessRights

Ξ Sys_State
u?: users; newu?: users; rights?: ℙmatautho; m!: Message

newu?): Auth
u?): Auth
Mat /∈ Matrix(u?)
m! = Invalid Rights


Figure 11. Specification for Giving Rights for Access Control Matrix

As shown in Figure 11, the Access Control Matrix authorization process has four possible cases. In the first case, a user is added successfully after satisfying all the conditions. The first condition will check for the existence of a user, to be given ACM rights. As mentioned before, it is not enough for a new user to access system by just adding to the user list. A user is required to have rights to enter into system. In second condition, system checks for valid authorization for a user. After successful operation of both conditions,
system checks for user's right, trying to give ACM rights to a new user. If a user has rights to give ACM rights to a new user, system lets a user to give rights to a new user. *MatrixAuthoNewUserNotFound* gives a message if a new user does not exists in userlist or does not have authentication rights to access the system. This violation of condition prevents a new user from having ACM rights. Same as this operation, *MatrixAuthoUserNotFound* prevents further operation for giving rights for ACM, if it finds a user trying to add a new user that does not exists in the user list or is not authenticated user. *MatrixAuthoInvalidAccessRights* specifies the case when a user does not have valid rights for giving a new user the rights for ACM.

4.4 Specification for Adding Table to Tablelist

In this section, the specification for adding a table to the system is discussed. Users are required to have valid rights for adding tables to the Access Control Matrix. Figure 12 shows the specification for adding a table to the *Tablist*. This operation consists of four possible cases. *MatrixAdd* shows the case for adding a table to the *Tablist* successfully. For this purpose, first of all the existence of a user is checked for both user list and authentication list. Next step is to check for the valid rights for adding a new table to the *Tablist*. The third step is to check whether the table already exists in the system or not, and the table must not be present in the list. If all conditions are satisfied, a table is added to the system successfully.
MatrixAddTable

\[ \Delta \text{Sys\_State} \]
\[
u?:\text{users}; \text{rights}?: \mathbb{P}\text{matautho}; t?:\text{TABLE}; \text{col}?: \mathbb{P}\text{COLUMN}; \text{rec}?: \mathbb{P}\text{RECORD}; \ m! = \text{Message} \]
\[
u? \in \text{Auhte} \]
\[
u? \in \text{dom UserAutho} \]
\[
t?: \notin \text{Tablist} \]
\[
\text{Table} \in \text{Matrix}(u?) \]
\[
\text{Tablist}' = \text{Tablist} \cup \{t?\} \]
\[
\text{TabDetails}(t?)' = \text{TabDetails}(t) \oplus \{\text{Pcol}?, \text{Prec}?\} \]

MatrixAddTableUserNotExists

\[ \Xi \text{Sys\_State} \]
\[
u?:\text{users}; \text{rights}?: \mathbb{P}\text{matautho}; t?:\text{TABLE}; \text{col}?: \mathbb{P}\text{COLUMN}; \text{rec}?: \mathbb{P}\text{RECORD} m! = \text{Message} \]
\[
u? \notin \text{Auhte} \lor u? \notin \text{dom UserAutho} \]
\[
m! = \text{User Does Not Exists} \]

MatrixAddTableExists

\[ \Xi \text{Sys\_State} \]
\[
u?:\text{users}; \text{rights}?: \mathbb{P}\text{matautho}; t?:\text{TABLE}; \text{col}?: \mathbb{P}\text{COLUMN}; \text{rec}?: \mathbb{P}\text{RECORD} m! = \text{Message} \]
\[
t? \in \text{Tablist} \]
\[
m! = \text{Table Already Exists} \]

MatrixAddTableInvalidRights

\[ \Xi \text{Sys\_State} \]
\[
u?:\text{users}; \text{rights}?: \mathbb{P}\text{matautho}; t?:\text{TABLE}; \text{col}?: \mathbb{P}\text{COLUMN}; \text{rec}?: \mathbb{P}\text{RECORD} m! = \text{Message} \]
\[
\text{Table} \in \text{Matrix}(u?) \]
\[
m! = \text{Invalid Rights} \]

AddTable \equiv \text{MatrixAddTable} \lor \text{MatrixAddTableUserNotExists} \lor \text{MatrixAddTableExists} \lor \text{MatrixAddTableInvalidRights} \]

Figure 12. Specification for Adding Table to Tablist

\text{MatrixAddTableUserNotExists} shows the case where a user is not found in the userlist or list of authenticated users. This operation prevents an unknown user from adding a table to the table list. 
\text{MatrixAddTableExists} shows the case where the system finds, a user is trying to add already existing table to the list. This specification prevents the system from creating redundant copy of table. 
\text{MatrixAddTableInvalidRights} shows the case of preventing an unauthorized user from adding table to the Tablist.
4.5 Specification for Giving Table Rights to Users

Figure 13 shows the specification for giving table access rights to authenticated users. `GiveTableRights` process has five possible cases. `AddTableRights` shows the case where table access rights are given to the clients from the users having valid rights for giving table access rights to other users. First the existence of a user is checked and similarly the availability of a client is also checked. If user and client exist in system, the existence of a table is then checked in the `Tablist`. Next step is to check whether a user has valid rights to give access rights of table to client or not. Once all the conditions are satisfied, table access rights for table are added to the Access Control Matrix for a client. The rest of the specification is for cases where above mentioned conditions are violated during the process for giving access rights.
\( \textit{AddTableRights} \)

\[
\begin{align*}
\Delta \text{Sys\_State} \\
u? \!: \text{users}; \text{client}? \!: \text{users}; t? \!: \text{Tablist}; \text{rights}\! : \text{mode}; m! : \text{Message} \\
u? \in \text{Aute} \\
u? \in \text{dom} \text{UserAutho} \\
\text{client}? \in \text{Aute} \\
\text{client}? \in \text{dom} \text{UserAutho} \\
t? \in \text{Tablist} \\
\text{Table} \in \text{Matrix}(u?) \\
\text{AccTab} (u?)' = \text{AccTab} (u) \cup \{t?, \text{rights}\?\} \\
\text{Aute}' = \text{Aute} \\
\text{UserAutho}' = \text{UserAutho} \\
\text{Tablist}' = \text{Tablist} \\
\text{Matrix}' = \text{Matrix} \\
m! = \text{Table Rights Given Successfully} \\
\end{align*}
\]

\( \textit{UserDoesNotExist} \)

\[
\Xi \text{Sys\_State} \\
u? \!: \text{users}; \text{client}? \!: \text{users}; t? \!: \text{Tablist}; \text{rights}\! : \text{mode}; m! : \text{Message} \\
u? \notin \text{Aute} \lor u? \notin \text{dom} \text{UserAutho} \\
m! = \text{User Does Not Exists} \\
\]

\( \textit{ClientDoesNotExist} \)

\[
\Xi \text{Sys\_State} \\
u? \!: \text{users}; \text{client}? \!: \text{users}; t? \!: \text{Tablist}; \text{rights}\! : \text{mode}; m! : \text{Message} \\
\text{client}? \notin \text{Aute} \lor \text{client}? \notin \text{UserAutho} \\
m! = \text{Client Does Not Exists} \\
\]

\( \textit{TableDoesNotExist} \)

\[
\Xi \text{Sys\_State} \\
u? \!: \text{users}; \text{client}? \!: \text{users}; t? \!: \text{Tablist}; \text{rights}\! : \text{mode}; m! : \text{Message} \\
t? \notin \text{Tablist} \\
m! = \text{Table Does Not Exists} \\
\]

\( \textit{InvalidRights} \)

\[
\Xi \text{Sys\_State} \\
u? \!: \text{users}; \text{client}? \!: \text{users}; t? \!: \text{Tablist}; \text{rights}\! : \text{mode}; m! : \text{Message} \\
\text{Table} \notin \text{Matrix}(u?) \\
m! = \text{Invalid Rights} \\
\]

\( \textit{GiveTableRights} \triangleq \textit{AddTableRights} \lor \textit{UserDoesNotExist} \lor \textit{ClientDoesNotExist} \lor \textit{TableDoesNotExist} \lor \textit{InvalidRights} \)

Figure 13. Specification for Giving Table Rights to User
4.6 Specification for Adding Column to Table

In this section, the specification for adding a column to an already added table in the database system is discussed. The specification is similar to that for adding a new table to the system. As shown in Figure 14 the AddColumn process has five possible cases.

As shown in Figure 14 case MatrixAddColumn shows the case where a new column is successfully added to the already existing table. Several conditions must be satisfied before adding a column to the table. These conditions are existence of user and table, valid rights to add a column and absence of a column being added to the table. MatrixAddColumnUserNotExists and MatrixAddColumnTableDoesNotExists cases show the cases where the column user is trying to add and the table in which column is being added are not present in system. Case MatrixAddColumnInvalidRights shows the violation of one of the important constraints, where a user does not have a valid right to add a column in the table. Case MatrixAddColumnAlreadyExists is for the situation where the column already exists in that table.
MatrixAddColumn

\[ \Delta \text{Sys}_\text{State} \]
\[ u? \in \text{users}; \ rights? : \mathbb{P} \text{matautho}; t? : \text{Tablist}; \ col? : \text{COLUMN}; \ rec? : \mathbb{P} \text{RECORD}; \ m! = \text{Message} \]

\[ u? \notin \text{Auhte} \]
\[ u? \notin \text{dom UserAutho} \]
\[ t? : \in \text{Tablist} \]
\[ \text{Table} \in \text{Matrix}(u?) \]
\[ \text{col}? \notin \text{dom TabDetails}(t?) \]
\[ \text{TabDetails}(t?)' = \text{TabDetails}(t?) \cup \{\text{col}? , \mathbb{P} \text{rec}? \} \]
\[ m! = \text{Column Added Successfully to the table} \]

MatrixAddColumnUserNotExists

\[ \Xi \text{Sys}_\text{State} \]
\[ u? : \text{users}; \ rights?: \mathbb{P} \text{matautho}; t?: \text{Tablist}; \ col?: \text{COLUMN}; \ rec?: \mathbb{P} \text{RECORD}; m! = \text{Message} \]

\[ u? \notin \text{Auhke} \lor u? \notin \text{dom UserAutho} \]
\[ m! = \text{User Does Not Exists} \]

MatrixAddColumnTableDoesNotExists

\[ \Xi \text{Sys}_\text{State} \]
\[ u? : \text{users}; \ rights?: \mathbb{P} \text{matautho}; t?: \text{Tablist}; \ col?: \text{COLUMN}; \ rec?: \mathbb{P} \text{RECORD}; m! = \text{Message} \]

\[ t? \notin \text{Tablist} \]
\[ m! = \text{Table Does Not Exists} \]

MatrixAddColumnInvalidRights

\[ \Xi \text{Sys}_\text{State} \]
\[ u?: \text{users}; \ rights?: \mathbb{P} \text{matautho}; t?: \text{Tablist}; \ col?: \text{COLUMN}; \ rec?: \mathbb{P} \text{RECORD}; m! = \text{Message} \]

\[ \text{Table} \in \text{Matrix}(u?) \]
\[ m! = \text{Invalid Rights} \]

MatrixAddColumnAlreadyExists

\[ \Xi \text{Sys}_\text{State} \]
\[ u?: \text{users}; \ rights?: \mathbb{P} \text{matautho}; t?: \text{Tablist}; \ col?: \text{COLUMN}; \ rec?: \mathbb{P} \text{RECORD}; m! = \text{Message} \]
\[ \text{col}? \in \text{dom TabDetails}(t?) \]
\[ m! = \text{Column Already Exists} \]

AddColumn \( \triangleq \) MatrixAddColumn \( \lor \) MatrixAddColumnUserNotExists \( \lor \) MatrixAddColumnTableDoesNotExists \( \lor \) MatrixAddColumnInvalidRights \( \lor \) MatrixAddColumnAlreadyExists

Figure 14. Specification for Adding Column to Table
4.7 Specification for Deleting Column from Table

This section is to discuss the specification for deleting a column from the table. Figure 15 shows the specification for deleting a column from the table with valid access rights. The specification for deleting a column from the table is similar to that for adding a table or adding a column.

Figure 15 shows the specification for DeleteColumn process. MatrixDeleteColumn shows the case where a column is successfully deleted from the table after satisfying certain conditions. Constrains must be satisfied before deleting a column from the table. Unlike the process of adding columns, in this process the column to be deleted must exists in the table. In the process of deleting a column, a column and respective records are removed from the table. As shown in Figure 15 DeleteColumn has five possible cases.
DeleteColumn $\triangleq$ MatrixDeleteColumn $\lor$ MatrixDeleteColumnUserNotExists $\lor$
MatrixDeleteColumnTableDoesNotExists $\lor$
MatrixDeleteColumnInvalidRights $\lor$
MatrixDeleteColumnDoesNotExists

Figure 15. Specification for Deleting Column from Table
4.8 Specification for Deleting Table from Matrix

In this section, the specification for removing or deleting a table from the matrix with valid rights is provided. Figure 16 shows the specification for deleting a table from the system. The specification is similar to that for adding a table. The only difference in condition checking in $\text{MatrixDeleteTable}$ is that a table to be deleted must exist in the system. Another variation in the specification comes in $\text{MatrixDeleteTableDoesNotExist}$, where condition for the table existence is violated. As shown in the Figure 16 $\text{DeleteTable}$ process has four possible cases.
\(\Delta_{\text{Sys State}}\)
\(u? : \text{users}; \ \text{rights}?: \mathbb{P}_\text{matautho}; \ t? : \text{Tablist}; \ m! := \text{Message}\)

\(u? \in \text{Auhte}\)
\(u? \in \text{dom UserAutho}\)
\(t? \in \text{Tablist}\)

\(\text{Table} \in \text{Matrix}(u?)\)
\(\text{Tablist} = \text{Tablist} \cup \{t?\}\)

\(\Xi_{\text{Sys State}}\)
\(u? : \text{users}; \ \text{rights}?: \mathbb{P}_\text{matautho}; \ t? : \text{Tablist}; \ m! := \text{Message}\)

\(u? \notin \text{Auhte} \lor u? \notin \text{dom UserAutho}\)
\(m! = \text{User Does Not Exists}\)

\(\Xi_{\text{Sys State}}\)
\(u? : \text{users}; \ \text{rights}?: \mathbb{P}_\text{matautho}; \ t? : \text{Tablist}; \ m! := \text{Message}\)

\(t? \notin \text{Tablist}\)
\(m! = \text{Table Does Not Exists}\)

\(\Xi_{\text{Sys State}}\)
\(u? : \text{users}; \ \text{rights}?: \mathbb{P}_\text{matautho}; \ t? : \text{Tablist}; \ m! := \text{Message}\)

\(\text{Table} \notin \text{Matrix}(u?)\)
\(m! = \text{Invalid Rights}\)

\(\text{DeleteTable} \equiv \text{MatrixDeleteTable} \lor \text{MatrixDeleteTableUserNotExists} \lor \text{MatrixDeleteTableDoesNotExists} \lor \text{MatrixDeleteTableInvalidRights}\)

Figure 16. Specification for Deleting Table from \(\text{Tablist}\)
Chapter 5

SPECIFICATIONS FOR TABLE MANIPULATION OPERATIONS

This chapter presents specifications for secure table manipulation operations. These operations are table read, edit/update, insert and delete operations. Here, delete means to delete records from the table, not the whole table from the database. The difference between edit/update and insert is that, in edit/update authorized users are allowed to update existing records, whereas in insert authorized users are allowed to insert a new record. Detailed specification with parallel execution control mechanism is given in Appendix B. In first section, the specification for read table operation is given. The second and third sections explain specifications for update and insert operation. The forth section discusses about delete operation.

5.1 Specification for Read Table Operation

In this section, the specification for the operation to securely read a table is discussed. Figure 17 shows that the specification for the ReadTable process has four possible cases. As shown in Figure 17, all the conditions are the same as that in the previously seen specifications. Condition check for the existence of user and table and for the verification of valid rights for table access rights are done as a primary steps towards a secure read operation. The main concern for secure read operation is that the table must not pass through any changes. Contents of the table must remain unchanged before the start and after the successful completion of the operation. As explained earlier, TabDetails contains information about table, its corresponding columns and column data. The state of the system must remain unchanged throughout the operation. The notation for this operation is shown in Figure 17. The other possible cases for ReadTable process for secure read operations are UserDoesNotExist, TableDoesNotExist and InvalidRights are defined in figure 17. As the name suggested UserDoesNotFound shows the case where a user is unknown to the system and does not exists. Similarly, TableDoesNotExist and InvalidRights show the cases where a user is trying to read a table that does not exists in the system and a user trying to read a table does not have a valid read right for that table.
Figure 17. Specification for Authorized Table *Read* Operation

\[ \text{ReadTable} \equiv \text{ReadSuccessful} \lor \text{UserDoesNotExists} \lor \text{TableDoesNotExists} \lor \text{InvalidRights} \]

5.2 Specification for *Edit/Update* Table Operation

As mentioned earlier in this chapter, *edit/update* operation is unlike *insert* operation. In *edit/update* operation the users not having insert right, are neither allowed to add new records nor delete existing records from respective tables. The specification for *edit/update* is slightly different from that for *insert*
operation. Once the authorization of the user for the operation is checked, in next step verification for a secure table manipulation operation is done. In this step, the system makes sure that only users with valid edit/update rights perform the operation.

As shown in Figure 18, `EditUpdateTable` process has five possible cases. In the `EditSuccessful` case, the existence of a user, table and valid table access right for editing table is first checked. The next condition is checked for the existence of column and corresponding record. `Edit` operation does not allow users to insert a new record into the table, but it allows users to edit existing records. That is why combination of column and record is checked before edit operation. After the operation, column and record values are overwritten into the table to avoid redundancy. The specification for cases showing violations of conditions such as non-existence of user, table and valid column-record combination and invalid table access right are also show in Figure 18.
<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EditSuccessful</td>
<td>$\Delta \text{Sys_State}$ $\text{u?#users}; \text{t?#Tablist}; \text{m!=Message}; \text{col?#cols}; \text{rec?#recs}$</td>
</tr>
<tr>
<td></td>
<td>$\text{u? \in \text{Authe}}$ $\text{u? \in \text{UserAutho}}$ $\text{t? \in \text{Tablist}}$ ${\text{t?#, \text{Edit}}} \in \text{AccTab(u?)}$ ${\text{col?#, \text{rec?}}} \in \text{TabDetails(t?)}$ $\text{TabDetails}'=\text{TabDetails}(t?) \oplus {\text{col?#, \text{rec?}}}$ $\text{Tablist}'=\text{Tablist}$ $\text{AccTab}'=\text{AccTab}$ $\text{Authe}'=\text{Authe}$ $\text{UserAutho}'=\text{UserAutho}$ $\text{m!} = \text{Read Successful}$</td>
</tr>
<tr>
<td>UserDoesNotExists</td>
<td>$\Xi \text{Sys_State}$ $\text{u?#users}; \text{t?#Tablist}; \text{m!=Message}; \text{col?#cols}; \text{rec?#recs}$</td>
</tr>
<tr>
<td></td>
<td>$\text{u? \notin \text{Authe}} \lor \text{u? \notin \text{UserAutho}}$ $\text{m!} = \text{User Does Not Exists}$</td>
</tr>
<tr>
<td>TableDoesNotExists</td>
<td>$\Xi \text{Sys_State}$ $\text{u?#users}; \text{t?#Tablist}; \text{m!=Message}; \text{col?#cols}; \text{rec?#recs}$</td>
</tr>
<tr>
<td></td>
<td>$\text{t? \notin \text{Tablist}}$ $\text{m!} = \text{Table Does Not Exists}$</td>
</tr>
<tr>
<td>InvalidRights</td>
<td>$\Xi \text{Sys_State}$ $\text{u?#users}; \text{t?#Tablist}; \text{m!=Message}; \text{col?#cols}; \text{rec?#recs}$</td>
</tr>
<tr>
<td></td>
<td>${\text{t?#, \text{Edit}}} \notin \text{AccTab(u?)}$ $\text{m!} = \text{Invalid Rights}$</td>
</tr>
<tr>
<td>ColRecNotFound</td>
<td>$\Xi \text{Sys_State}$ $\text{u?#users}; \text{t?#Tablist}; \text{m!=Message}; \text{col?#cols}; \text{rec?#recs}$</td>
</tr>
<tr>
<td></td>
<td>${\text{col?#, \text{rec?}}} \notin \text{TabDetails(t?)}$ $\text{m!} = \text{Column Record Not Found}$</td>
</tr>
</tbody>
</table>

**EditUpdateTable** $\triangleq \text{EditSuccessful} \lor \text{UserDoesNotExists} \lor \text{TableDoesNotExists} \lor \text{InvalidRights} \lor \text{ColRecNotFound}$

Figure 18. Specification for Authorized Table Edit/Update Operation
5.3 Specification for Insert Operation

Figure 19 shows the specification for the secure insert record operation. Similar to the previously seen specification for other operations, insert record operation has five possible cases. InsertSuccessful shows the case of inserting a record successfully to the table. Initial conditions for the operation are the same as that of the previous operations. Existence of user, table and valid table manipulation rights are verified before performing insert record operation. Moreover, existence of the column in which a user is trying to insert a new record is also checked. Once every condition is satisfied, record is inserted into the column. Post conditions for the operation are also checked after data is inserted. The difference between edit and insert is given by the specification. In edit data is overwritten into the table, so that overwrite operator is used in the specification. In insert operation, new record is added to the table so, union operator is used in the specification.

The specification for cases for violation of various conditions is also shown in Figure 19. Some of the cases are similar to the cases of the edit operation. For example, a user does not exist, the table not found and a user has invalid rights to perform insert operation. The only difference is that in edit existence of column and corresponding records are checked, whereas in insert only existence of column is confirmed.
\[ \Delta \text{Sys}_{\text{State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; \text{col}? : \text{cols}; \text{rec}? : \text{RECORD} \]

\[ u? \in \text{Authe} \]
\[ u? \in \text{UserAutho} \]
\[ t? \in \text{Tablist} \]
\[ \{ t?, \text{Insert} \} \in \text{AccTab}(u?) \]
\[ \text{col}? \notin \text{dom} \text{TabDetails}(t?) \]
\[ \text{TabDetails}' = \text{TabDetails}(t?) \cup \{ \text{col}?, \text{rec}? \} \]
\[ \text{Tablist}' = \text{Tablist} \]
\[ \text{AccTab}' = \text{AccTab} \]
\[ \text{Authe}' = \text{Authe} \]
\[ \text{UserAutho}' = \text{UserAutho} \]
\[ m! = \text{Insert Successful} \]

\[ \Xi \text{Sys}_{\text{State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; \text{col}? : \text{cols}; \text{rec}? : \text{RECORD} \]

\[ u? \notin \text{Authe} \lor u? \notin \text{UserAutho} \]
\[ m! = \text{User Does Not Exists} \]

\[ \Xi \text{Sys}_{\text{State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; \text{col}? : \text{cols}; \text{rec}? : \text{RECORD} \]

\[ t? \notin \text{Tablist} \]
\[ m! = \text{Table Does Not Exists} \]

\[ \Xi \text{Sys}_{\text{State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; \text{col}? : \text{cols}; \text{rec}? : \text{RECORD} \]

\[ \{ t?, \text{Insert} \} \notin \text{AccTab}(u?) \]
\[ m! = \text{Invalid Rights} \]

\[ \Xi \text{Sys}_{\text{State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; \text{col}? : \text{cols}; \text{rec}? : \text{RECORD} \]

\[ \text{col}? \notin \text{dom} \text{TabDetails}(t?) \]
\[ m! = \text{Column Does Not Exists} \]

\[ \text{InsertData} \triangleq \text{InsertSuccessful} \lor \text{UserDoesNotExist} \lor \text{TableDoesNotExist} \lor \text{InvalidRights} \lor \text{ColumnNotFound} \]

Figure 19. Specification for Secure Insert Operation
5.4 Specification for *Delete* Operation

Users having valid delete record rights for the corresponding tables can delete records from those tables. The specification for deleting records from a table is similar to that of the previously seen *edit* and *insert* operations. The constraints are somewhat similar to the previous specifications for other operations. Figure 20 shows the specification for the *delete* record operation.

Figure 20 shows the specification for deleting a record from the table with valid right. The specification is similar to the specifications for *insert* record operation. The only new condition added is the verification for existence of the record to be deleted from the table. *DeleteSuccessful* shows the case for deleting a record successfully from the table. *DeleteData* process has six possible cases.
DeleteSuccessful

ΔSys_State

ō?:users; t?:Tablist; m! = Message; col?:cols; rec?:recs

ō? ∈ Authe ∧ ō? ∈ UserAutho ∧ t? ∈ Tablist

{t?, Delete} ∈ AccTab(ō?) ∧ col? ∈ dom TabDetails(t?) ∧ rec? ∈ ran TabDetails(t?)

TabDetails′ = TabDetails(t?) ⊕ {col?, rec?}

Tablist′ = Tablist ∧ AccTab′ = AccTab ∧ Authe′ = Authe

UserAutho′ = UserAutho

m! = Delete Successful

UserDoesNotExist

ΣSys_State

ō?:users; t?:Tablist; m! = Message; col?:cols; rec?:recs

ō? ∉ Authe ∨ ō? ∉ UserAutho

m! = User Does Not Exists

TableDoesNotExist

ΣSys_State

ō?:users; t?:Tablist; m! = Message; col?:cols; rec?:recs

t? ∉ Tablist

m! = Table Does Not Exists

InvalidRights

ΣSys_State

ō?:users; t?:Tablist; m! = Message; col?:cols; rec?:recs

{t?, Delete} ∉ AccTab(ō?)

m! = Invalid Rights

ColumnNotFound

ΣSys_State

ō?:users; t?:Tablist; m! = Message; col?:cols; rec?:recs

col? ∉ dom TabDetails(t?)

m! = Column Does Not Exists

RecordNotFound

ΣSys_State

ō?:users; t?:Tablist; m! = Message; col?:cols; rec?:recs

rec? ∉ ran TabDetails(t?)

m! = Record Does Not Exists

DeleteData ≡ DeleteSuccessful ∨ UserDoesNotExist ∨ TableDoesNotExist ∨ InvalidRights ∨ ColumnNotFound ∨ RecordNotFound

Figure 20. Specification for Secure Delete Operation
Chapter 6

SPECIFICATIONS FOR PROVIDING COLUMN LEVEL SECURITY

In previous chapters, providing table level security to deductive database systems is discussed. In Access Control Matrix, tables and rights to access are added to perform manipulation operations on those tables. This mechanism helps to restrict the unauthorized users from accessing tables in system without any valid rights. In real world, sometimes partial access is allowed to the users. In other words, not all the information but some information is allowed to be accessed by users with valid rights. For example, a user can have access to the only certain columns of the table, not all the columns of the table. In this chapter, we shall discuss specifications for granting users column based security to the table and specifications for verifying valid access rights before letting users to perform any operation on system resources.

Specifications for this scenario are almost similar to the scenario discussed in Chapters 4 and 5. There would be some changes in the specification for system state $Sys\_State$. Couple of more relations and functions will be added into the $Sys\_State$. Detailed specifications with concurrent execution control for this section are given in Appendix C. We shall not discuss step by step construction of $Sys\_State$ in this chapter. As mentioned before $Sys\_State$ specification is almost similar to the previous scenario, we shall discuss finalized $Sys\_State$ specification.

6.1 System State specification

In this section, an updated $Sys\_State$ for providing column level security to deductive database system is discussed. Figure 21 shows the specification for extended $Sys\_State$. 
Figure 21 shows the specification for the Sys State updated according to the new requirements for giving column-level security to the system. The only change made to the specification is the inclusion of two new functions, which are part of Access Control Matrix for new requirements. TabColAuth is a function with a list of tables in the system as a domain and a set of columns and access mode as a range. It means, this function facilitates table to respective columns and columns to valid access rights for the users. TabColAuth function acts as a range for the function AccTabCol. This function has list of authenticated users as a domain. These two functions represent the information about whether a user possesses certain set of rights for accessing columns of particular table.
The specification for adding users to the system, inserting and removing tables from the system, giving users rights for Access Control matrix to other users are same as the specifications discussed in Chapter 4. The specifications for secure table manipulation operations and for giving access rights for tables to users are different from the specifications discussed in previous chapter.

6.2 Specification for Giving Access Rights to User

In this section, the specification for giving column level access rights to the users is discussed. The specification is similar to the scenario discussed in previous chapter. Figure 22 shows the specification for giving access rights to the users to access and perform operations on tables.
AddTableRightsNewColumn

\[ \Delta \text{Sys} \_ \text{State} \]

\[ u \,: \text{users}; \, c \,: \text{Tablist}; \, \text{col} \,: \text{cols}; \, \text{rights} \,: \text{ℙ} \_ \text{mode}; \, m \,: \text{Message} \]

\[ u \, \in \, \text{Auhte} \, \land \, u \, \in \, \text{dom UserAutho} \, \land \, c \, \in \, \text{dom TabDetails}(u) \, \land \, \text{Table} \, \in \, \text{Matrix}(u) \]

if \( \neg \exists \, \text{AccTabCol}(\text{client}!) \) then \( \text{AccTabCol}(\text{client}!) = \text{AccTabCol}(\text{client}!) \cup \{ \text{?, (col!?, ℙ \_ \text{mode}!)} \} \)

else ( if \( \text{col}! \in \text{AccTabCol}(\text{client}?, c!) \) then \( \text{AccTabCol}(\text{client}!, c!) = \text{AccTabCol}(\text{client}!, c!) \cup \{ \text{?, ℙ \_ \text{mode}!} \} \)

else \( \text{AccTabCol}(\text{client}!, c!, \text{col}! \cup \{ \text{?, ℙ \_ \text{mode}!} \} \)

\( \text{TabDetails}' = \text{TabDetails} \land \text{Auhte}' = \text{Auhte} \land \text{UserAutho'} = \text{UserAutho} \land \text{Tablist}' = \text{Tablist} \land \text{Matrix'} = \text{Matrix} \)

\( m! = \text{Table Rights Given Successfully} \)

UserDoesNotExists

\[ \exists \text{Sys} \_ \text{State} \]

\[ u \,: \text{users}; \, c \,: \text{Tablist}; \, \text{col} \,: \text{cols}; \, \text{rights} \,: \text{ℙ} \_ \text{mode}; \, m \,: \text{Message} \]

\[ u \, \notin \, \text{Auhte} \lor \, u \, \notin \, \text{dom UserAutho} \]

\( m! = \text{User Does Not Exists} \)

ClientDoesNotExists

\[ \exists \text{Sys} \_ \text{State} \]

\[ u \,: \text{users}; \, c \,: \text{Tablist}; \, \text{col} \,: \text{cols}; \, \text{rights} \,: \text{ℙ} \_ \text{mode}; \, m \,: \text{Message} \]

\[ \text{client}! \, \notin \, \text{Auhte} \lor \, \text{client}! \, \notin \, \text{UserAutho} \]

\( m! = \text{Client Does Not Exists} \)

TableDoesNotExists

\[ \exists \text{Sys} \_ \text{State} \]

\[ u \,: \text{users}; \, c \,: \text{Tablist}; \, \text{col} \,: \text{cols}; \, \text{rights} \,: \text{ℙ} \_ \text{mode}; \, m \,: \text{Message} \]

\[ c! \, \notin \, \text{Tablist} \]

\( m! = \text{Table Does Not Exists} \)

InvalidRights

\[ \exists \text{Sys} \_ \text{State} \]

\[ u \,: \text{users}; \, c \,: \text{Tablist}; \, \text{col} \,: \text{cols}; \, \text{rights} \,: \text{ℙ} \_ \text{mode}; \, m \,: \text{Message} \]

\[ \text{Table} \, \notin \, \text{Matrix}(u!) \]

\( m! = \text{Invalid Rights} \)

ColumnDoesNotExists

\[ \exists \text{Sys} \_ \text{State} \]

\[ u \,: \text{users}; \, c \,: \text{Tablist}; \, \text{col} \,: \text{cols}; \, \text{rights} \,: \text{ℙ} \_ \text{mode}; \, m \,: \text{Message} \]

\[ \text{col}! \, \notin \, \text{dom TabDetails}(c!) \]

\( m! = \text{Column Does Not Exists} \)

GiveTableRights \( \triangleq \) AddTableRights \( \lor \) UserDoesNotExists \( \lor \) ClientDoesNotExists \( \lor \) TableDoesNotExists \( \lor \) InvalidRights \( \lor \) ColumnDoesNotExists)

Figure 22. Specification for Giving Table Rights to User
As shown in Figure 22, *GiveTableRights* process has six possible cases. *AddTableRights* shows the case for giving a user the access rights for particular table and column successfully. Other cases show the cases for violations of various conditions. For example, user, table and column do not exist in the system, and a user does not have valid access rights to give access rights to other users for tables in system.

6.3 Specification for *Read* Table Operation

This section discusses about the specification of performing secure *read* operation for column level access control. As shown in Figure 23, the specification for *ReadTable* process consists of five parts. *ReadSuccessful* shows the case where table *read* operation is done successfully for users having valid access right for *read* table operation.

As shown in Figure 23, *UserDoesNotExist* shows the case where condition for existence of the user is violated. It means this condition occurs when the user trying to read table does not exists in userlist. *TableDoesNotExist* shows the condition where table does not present in the system. *ColumnDoesNotExist* shows the case when the column does not present in the table. *InvalidRights* shows the condition where user does not have valid rights to read records from a particular column.
_ReadSuccessful_

\[ \Xi \text{Sys}_\text{State} \]
\[ u? : \text{users}; t? : \text{Tablist}; c? : \text{cols}; m! = \text{Message} \]
\[ u? \in \text{Authe} \]
\[ u? \notin \text{UserAutho} \]
\[ t? \notin \text{Tablist} \]
\[ c? \notin \text{dom TabDetails}(t?) \]
\[ \{c?, \text{Read}\} \notin \text{AccTabCol}(u?, t?) \]
\[ \text{AccTabCol} = \text{AccTabCol} \]
\[ \text{TabDetails}' = \text{TabDetails} \]
\[ \text{Tablist}' = \text{Tablist} \]
\[ \text{AccTab}' = \text{AccTab} \]
\[ \text{Authe}' = \text{Authe} \]
\[ \text{UserAutho}' = \text{UserAutho} \]
\[ m! = \text{Read Successful} \]

_UserDoesNotExists_

\[ \Xi \text{Sys}_\text{State} \]
\[ u? : \text{users}; t? : \text{Tablist}; c? : \text{cols}; m! = \text{Message} \]
\[ u? \notin \text{Authe} \lor u? \notin \text{UserAutho} \]
\[ m! = \text{USER Does Not Exists} \]

_TableNotExists_

\[ \Xi \text{Sys}_\text{State} \]
\[ u? : \text{users}; t? : \text{Tablist}; c? : \text{cols}; m! = \text{Message} \]
\[ t? \notin \text{Tablist} \]
\[ m! = \text{Table Does Not Exists} \]

_InvalidRights_

\[ \Xi \text{Sys}_\text{State} \]
\[ u? : \text{users}; t? : \text{Tablist}; c? : \text{cols}; m! = \text{Message} \]
\[ \{c?, \text{Read}\} \notin \text{AccTabCol}(u?, t?) \]
\[ m! = \text{Invalid Rights} \]

_ColumnDoesNotExists_

\[ \Xi \text{Sys}_\text{State} \]
\[ u? : \text{users}; t? : \text{Tablist}; c? : \text{cols}; m! = \text{Message} \]
\[ c? \notin \text{dom TabDetails}(t?) \]
\[ m! = \text{Column Does Not Exists} \]

ReadTable \(\equiv\) ReadSuccessful \(\lor\) UserDoesNotExists \(\lor\) TableDoesNotExists \(\lor\) InvalidRights \(\lor\) ColumnDoesNotExists

Figure 23. Specification for Authorized Table Read Operation
6.4 Specification for Edit/Update Table Operation

This section discusses the specification for secure column level edit operation. This operation will not add new records or delete any records from the tables. The specifications for edit and delete operation are slightly different from that for insert and read operations. In edit and delete operations, existence of records is verified before performing operation. In insert and read operation, only the existence of table and column is verified.

Figure 24 shows the specification for performing secure Edit/Update operation with valid access rights. EditSuccessful shows the case for successful edit operation. EditUpdateTable process has five possible cases. As mentioned previously, in this specification record must exists in respective column of the table; user is trying to edit. ColRecNotFound shows the case where the record does not present in the column when a user is trying to edit.
$\Delta \text{Sys\_State}$

$u?_ designed\_user; t?: designed\_tablist; m! = \text{Message}; col?: cols; rec?: recs$

$u? \in \text{Authe}$
$u? \in \text{UserAutho}$
$t? \in \text{Tablist}$
\{col?, rec?\} $\in \text{TabDetails}(t?)$
\{col?, Edit\} $\in \text{AccTabCol}(u?, t?)$
\text{TabDetails}' = \text{TabDetails}(t?) \uplus \{col?, rec?\}$
\text{AccTabCol}' = \text{AccTabCol}$
\text{Tablist}' = \text{Tablist}$
\text{AccTab}' = \text{AccTab}$
\text{Authe}' = \text{Authe}$
\text{UserAutho}' = \text{UserAutho}$
$m! = \text{Read Successful}$

$\Xi \text{Sys\_State}$

$u?_ designed\_user; t?: designed\_tablist; m! = \text{Message}; col?: cols; rec?: recs$

$u? \notin \text{Authe} \lor u? \notin \text{UserAutho}$
$m! = \text{User Does Not Exists}$

$\Xi \text{Sys\_State}$

$u?_ designed\_user; t?: designed\_tablist; m! = \text{Message}; col?: cols; rec?: recs$

$t? \notin \text{Tablist}$
$m! = \text{Table Does Not Exists}$

$\Xi \text{Sys\_State}$

$u?_ designed\_user; t?: designed\_tablist; m! = \text{Message}; col?: cols; rec?: recs$

\{col?, Edit\} $\notin \text{AccTabCol}(u?, t?)$
$m! = \text{Invalid Rights}$

$\Xi \text{Sys\_State}$

$u?_ designed\_user; t?: designed\_tablist; m! = \text{Message}; col?: cols; rec?: recs$

\{col?, rec?\} $\in \text{TabDetails}(t?)$
$m! = \text{Column Record Not Found}$

$\text{EditUpdateTable} \triangleq \text{EditSuccessful} \lor \text{UserDoesNotExist} \lor \text{TableDoesNotExist} \lor \text{InvalidRights} \lor \text{ColRecNotFound}$

Figure 24. Specification for Authorized Table Edit/Update Operation
6.5 Specification for Insert Operation

This section discusses the specification for inserting a new record into the table with valid rights. Figure 25 shows the case for performing a secure insert operation. InsertSuccessful shows the case for successfully inserting a record to the table with valid rights. Unlike the previously seen edit operation, here existence of record is not verified. In this operation, the existence of column is required and it is always verified before performing the insert operation.

As shown in Figure 25, InsertData process has five possible cases. UserDoesNotExists is one the cases showing violation of condition for existence of a user trying to perform insert operation. TableDoesNotExists case shows specifications where condition for existence of table is violated. ColumnNotFound case has specification showing violation of condition for presence of column in which a user is trying to insert a new record.
InsertSuccessful

\[ \Delta_{\text{Sys State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; col?: \text{cols}; rec?: \text{RECORD} \]
\[ u? \in \text{Authe} \]
\[ t? \notin \text{Tablist} \]
\[ \text{col}? \in \text{dom TabDetails}(t?) \]
\[ \{\text{col}, \text{Insert}\} \notin \text{AccTabCol}(u?, t?) \]
\[ \text{TabDetails}' = \text{TabDetails}(t?) \cup \{\text{col}, \text{rec}\?\} \]
\[ \text{AccTabCol}' = \text{AccTabCol} \]
\[ \text{Tablist}' = \text{Tablist} \]
\[ \text{AccTab}' = \text{AccTab} \]
\[ \text{Authe}' = \text{Authe} \]
\[ \text{UserAutho}' = \text{UserAutho} \]
\[ m! = \text{Insert Successful} \]

UserDoesNotExist

\[ \Xi_{\text{Sys State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; col?: \text{cols}; rec?: \text{RECORD} \]
\[ u? \notin \text{Authe} \lor u? \notin \text{UserAutho} \]
\[ m! = \text{User Does Not Exists} \]

TableDoesNotExist

\[ \Xi_{\text{Sys State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; col?: \text{cols}; rec?: \text{RECORD} \]
\[ t? \notin \text{Tablist} \]
\[ m! = \text{Table Does Not Exists} \]

InvalidRights

\[ \Xi_{\text{Sys State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; col?: \text{cols}; rec?: \text{RECORD} \]
\[ \{\text{col}, \text{Insert}\} \notin \text{AccTabCol}(u?, t?) \]
\[ m! = \text{Invalid Rights} \]

ColumnNotFound

\[ \Xi_{\text{Sys State}} \]
\[ u? : \text{users}; t? : \text{Tablist}; m! = \text{Message}; col?: \text{cols}; rec?: \text{RECORD} \]
\[ \text{col}? \notin \text{dom TabDetails}(t?) \]
\[ m! = \text{Column Does Not Exists} \]

InsertData \triangleq \text{InsertSuccessful} \lor \text{UserDoesNotExist} \lor \text{TableDoesNotExist} \lor \text{InvalidRights} \lor \text{ColumnNotFound}

Figure 25. Specification for Secure Insert Operation
6.6 Specification for Delete Operation

In this section, the specification for secure delete operation is discussed. Figure 26 shows the specification of performing secure delete operation on a column. The DeleteData operation has five possible cases. Case DeleteSuccessFul shows the case for performing secure delete operation with valid rights. Rest of the cases shows the cases for violations of conditions for verifying existence of a user, a table, a column and respective record, and valid access right.
\[\text{DeleteSuccessful} \]
\[\Delta \text{Sys\_State}\]
\[\text{u?} \in \text{Authe}\]
\[\text{u?} \in \text{UserAutho}\]
\[\text{t?} \in \text{Tablist}\]
\[\{\text{col?,rec?}\} \in \text{TabDetails}(t?)\]
\[\{\text{col?,Delete}\} \in \text{AccTabCol}(u?,t?)\]
\[\text{TabDetails'} = \text{TabDetails}(t?) \oplus \{\text{col?}\}\]  
\[\text{AccTabCol'} = \text{AccTabCol}\]
\[\text{Tablist'} = \text{Tablist}\]
\[\text{AccTab'} = \text{AccTab}\]
\[\text{Authe'} = \text{Authe}\]
\[\text{UserAutho'} = \text{UserAutho}\]
\[\text{m!} = \text{Delete Successful}\]

\[\text{UserDoesNotExists} \]
\[\Xi \text{Sys\_State}\]
\[\text{u?} \in \text{Authe} \lor \text{u?} \in \text{UserAutho}\]
\[\text{m!} = \text{User Does Not Exists}\]

\[\text{TableDoesNotExists} \]
\[\Xi \text{Sys\_State}\]
\[\text{t?} \in \text{Tablist}\]
\[\text{m!} = \text{Table Does Not Exists}\]

\[\text{InvalidRights} \]
\[\Xi \text{Sys\_State}\]
\[\{\text{col?,Delete}\} \notin \text{AccTabCol}(u?,t?)\]
\[\text{m!} = \text{Invalid Rights}\]

\[\text{ColRecNotFound} \]
\[\Xi \text{Sys\_State}\]
\[\{\text{col?,rec?}\} \notin \text{TabDetails}(t?)\]
\[\text{m!} = \text{Record Does Not Exists}\]

\[\text{DeleteData} \equiv \text{DeleteSuccessful} \lor \text{UserDoesNotExist} \lor \text{TableDoesNotExist} \lor \text{InvalidRights} \lor \text{ColRecNotFound}\]

Figure 26. Specification for Secure Delete Operation
In this chapter, the specifications are provided for providing column level security to deductive database systems for secure query processing. In the first section, the updated $Sys\_State$ specification is presented. The later sections discuss the secure query processing for various table manipulation operations like insert, read, edit and delete.
Chapter 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 Conclusion

This report presents the formal specification aiming at providing secure query processing to deductive database systems for table-level and column-level access control. First deductive database system architecture is abstracted. The system state is specified. Specifications for secure table manipulation operations for providing table-level security are defined. These steps are further extended to address column level security. Query processing is the core part of any database system and requires more attention in terms of security. A secure query processing is a prominent part in the success of database systems. The specifications written and discussed in this project can be used to develop and analyze secure query processing in deductive databases. These specifications are very flexible and can be used for any type of deductive database systems with some modifications.

7.2 Future Enhancements

Specifications for providing more security can be specified by providing more secure query processing. For example, security clearance levels of the users can be specified in certain levels according to the value of record stored in table. Users having certain security clearance can access only those columns having particular value in particular range.

The system architecture can be extended by adding a separate storage for user access history. Stored access history can be used to verify future accesses to the system by the same user. This means the history of accesses to any particular table by a user can be checked when the user is trying to access the table. With the help of this additional feature database systems can achieve better security for query processing.
Rather than giving one token for each table, each table can have multiple read tokens. In this way read concurrency can be improved. Another enhancement is to give edit and delete tokens for each row of table. This enhancement can make database systems perform multiple manipulation operation on single table simultaneously.
APPENDIX A

System State and Administrative Operation Specifications for Table Level Security

[USER]
[TABLE]
[COLUMN]
[RECORD]
[Authenticated]
[Message]
[TOKEN]

MODE ::= Read | Edit | Insert | Delete
UAutho ::= Add | Delete
MatAutho ::= Mat | Table | User

\[
\text{token : Tablist} \rightarrow \text{TOKEN}
\]
Sys_State
users: ℙ USER
mode: ℙ MODE
Authe: ℙ USER
Tablist: ℙ TABLE
cols: ℙ COLUMN
recs: ℙ RECORD
mode: ℙ MODE
matautho: ℙ MatAutho
userautho: ℙ UAutho
TabCols: cols → ℙ recs
TabDetail: Tablist → ℙ TabCols
UserAutho: users → ℙ Authenticate
Matrix: users → ℙ MatAutho
AccTabAutho: Tablist → ℙ mode
AccTab: users → ℙ AccTabAutho
AccUserAutho: users → ℙ userautho
AccUser: users → ℙ AccUserAutho

-dom UserAutho ⊆ users
dom Matrix ⊆ users
dom TabDetails ⊆ Tablist
dom TabCols ⊆ cols
dom AccTabAutho ⊆ Tablist
dom AccTab ⊆ users
dom AccUser ⊆ USER
dom AccUserAutho ⊆ Tablist

ProDB
Sys_State
owner: TOKEN → users

GetToken
ΔProDB
u?: users; t?: TOKEN

u?: ∈ dom owner
owner’ = owner ⊕ {t?, u?}
FreeToken

$\Delta ProDB$

$u? : \text{users}; t? : \text{TOKEN}$

$u? = \text{owner}\ t?$

$\text{owner} = \{t?\} \Box \text{owner}$

AddLoginSuccess

$\Delta Sys\_State$

$u? : \text{USER}; m! : \text{Message}$

$u? \notin \text{Authe}$

$\text{Authe}' = \text{Authe} \cup \{u?\}$

$m! = \text{User Added Successfully}$

LoginExists

$\Xi Sys\_State$

$u? : \text{USER}; m! : \text{Message}$

$u? \in \text{Authe}$

$m! = \text{User Already Added}$

$AddLogin \equiv AddLoginSuccess \lor LoginExists$

AuthenticationAddSuccess

$\Delta Sys\_State$

$u? : \text{users}; newu? : \text{USER}; a? : \text{Authenticated}; m! : \text{Message}$

$newu? \in \text{Authe}$

$User \in \text{Matrix}(u?)$

$User\text{Autho}'(u?) = User\text{Autho}(u?) \cup \{a?\}$

$m! = \text{Authentication Added Successfully}$
\[ \text{AuthenticationNewUserNotExists} \]
\[
\begin{align*}
\Xi & \text{Sys\_State} \\
u? & : \text{users}; \text{newu}? : \text{USER}; \ a? : \text{Authenticated}; \ m! : \text{Message} \\
\text{newu}? & \notin \text{Auth} \\
m! & = \text{User Not Found}
\end{align*}
\]

\[ \text{AuthenticationInvalidRights} \]
\[
\begin{align*}
\Xi & \text{Sys\_State} \\
u? & : \text{users}; \text{newu}? : \text{USER}; \ a? : \text{Authenticated}; \ m! : \text{Message} \\
\text{User} & \notin \text{Matrix}(u?) \\
m! & = \text{Invalid Right}
\end{align*}
\]

\[ \text{AuthenticationAdd} \triangleq \text{AuthenticationAddSuccess} \lor \text{AuthenticationNewUserNotExists} \lor \text{AuthenticationInvalidRights} \]

\[ \text{LoginSuccess} \]
\[
\begin{align*}
\Xi & \text{Sys\_State} \\
u? & : \text{users}; \ a? : \text{Authenticated}; \ m!:\text{Message} \\
\text{a}? & \in \text{UserAutho}(u?) \\
m! & = \text{Login Successful}
\end{align*}
\]

\[ \text{LoginFailed} \]
\[
\begin{align*}
\Xi & \text{Sys\_State} \\
u? & : \text{users}; \ a? : \text{Authenticated}; \ m!:\text{Message} \\
\text{a}? & \notin \text{UserAutho}(u?) \\
m! & = \text{Login Failed, Invalid Authentications}
\end{align*}
\]

\[ \text{UserLogin} \triangleq \text{LoginSuccess} \lor \text{LoginFailed} \]
\[ MatrixAutho \]
\[ \Delta \text{Sys\_State} \]
\[ u\? : \text{users}; \text{newu}\? : \text{users}; \text{rights}\? : \mathcal{P} \text{matautho}; m! : \text{Message} \]

\[ \text{newu}\? \in \text{Authe} \]
\[ \text{newu}\? \in \text{dom UserAutho} \]
\[ u\? : \in \text{Authe} \]
\[ u\? \in \text{dom UserAutho} \]
\[ \text{Mat} \in \text{Matrix}(u\?) \]
\[ \text{Matrix } '(\text{newu}\?) = \text{Matrix } (\text{newu}\?) \cup \{\text{rights}\?\} \]
\[ m! = \text{Matrix Rights Added For User} \]

\[ MatrixAuthoNewUserNotFound \]
\[ \Xi \text{Sys\_State} \]
\[ u\? : \text{users}; \text{newu}\? : \text{users}; \text{rights}\? : \mathcal{P} \text{matautho}; m! : \text{Message} \]

\[ \text{newu}\? \notin \text{Authe} \lor \text{newu}\? \notin \text{dom UserAutho} \]
\[ m! = \text{New User Not In a List} \]

\[ MatrixAuthoUserNotFound \]
\[ \Xi \text{Sys\_State} \]
\[ u\? : \text{users}; \text{newu}\? : \text{users}; \text{rights}\? : \mathcal{P} \text{matautho}; m! : \text{Message} \]

\[ u\? \notin \text{Authe} \lor u\? \notin \text{dom UserAutho} \]
\[ m!= \text{User Not Found} \]

\[ MatrixAuthoInvalidAccessRights \]
\[ \Xi \text{Sys\_State} \]
\[ u\? : \text{users}; \text{newu}\? : \text{users}; \text{rights}\? : \mathcal{P} \text{matautho}; m! : \text{Message} \]

\[ \text{newu}\? : \text{Authe} \]
\[ u\? : \text{Authe} \]
\[ \text{Mat} \notin \text{Matrix}(u\?) \]
\[ m! = \text{Invalid Rights} \]

\[ AddMatrixAutho \triangleq \text{MatrixAutho} \lor \text{MatrixAuthoNewUserNotFound} \lor \text{MatrixAuthoUserNotFound} \]
\[ \lor \text{MatrixAuthoInvalidAccessRights} \]
MatrixAddTable

\[ \Delta \text{Sys\_State} \]
\[ u\? : \text{users}; \ \text{rights}?: \ P\text{matautho}; \ t\? : \text{TABLE}; \ \text{col}?: \ P\text{COLUMN}; \ \text{rec}?: \ P\text{RECORD}; \ m! = \text{Message} \]
\[ u? \in \text{Auhte} \]
\[ u? \in \text{dom UserAutho} \]
\[ t? : \notin \text{Tablist} \]
\[ \text{Table} \in \text{Matrix}(u?) \]
\[ \text{Tablist}' = \text{Tablist} \cup \{t?\} \]
\[ \text{TabDetails}(t?)' = \text{TabDetails}(t) \oplus \{P\text{col}?, P\text{rec}?\} \]

MatrixAddTableUserNotExists

\[ \Xi \text{Sys\_State} \]
\[ u\? : \text{users}; \ \text{rights}?: \ P\text{matautho}; \ t\? : \text{TABLE}; \ \text{col}?: \ P\text{COLUMN}; \ \text{rec}?: \ P\text{RECORD}; \ m! = \text{Message} \]
\[ u? \notin \text{Authe} \lor u? \notin \text{dom UserAutho} \]
\[ m! = \text{User Does Not Exists} \]

MatrixAddTableExists

\[ \Xi \text{Sys\_State} \]
\[ u\? : \text{users}; \ \text{rights}?: \ P\text{matautho}; \ t\? : \text{TABLE}; \ \text{col}?: \ P\text{COLUMN}; \ \text{rec}?: \ P\text{RECORD}; \ m! = \text{Message} \]
\[ t? \in \text{Tablist} \]
\[ m! = \text{Table Already Exists} \]

MatrixAddTableInvalidRights

\[ \Xi \text{Sys\_State} \]
\[ u\? : \text{users}; \ \text{rights}?: \ P\text{matautho}; \ t\? : \text{TABLE}; \ \text{col}?: \ P\text{COLUMN}; \ \text{rec}?: \ P\text{RECORD}; \ m! = \text{Message} \]
\[ \text{Table} \notin \text{Matrix}(u?) \]
\[ m! = \text{Invalid Rights} \]

AddTable \( \triangleq \) MatrixAddTable \( \lor \) MatrixAddTableUserNotExists \( \lor \) MatrixAddTableExists \( \lor \) MatrixAddTableInvalidRights
**AddTableRights**

\[ \Delta \text{Sys\_State} \]
\[ u^? : \text{users}; \ client^? : \text{users}; t^? : \text{Tablist}; \ rights : \mathbb{P}\text{mode}; m^! : \text{Message} \]

\[ u^? \in \text{Auhte} \]
\[ u^? \in \text{domUserAutho} \]
\[ \text{client^?} \in \text{Auhte} \]
\[ \text{client^?} \in \text{dom UserAutho} \]
\[ t^? \in \text{Tablist} \]
\[ \text{Table} \in \text{Matrix}(u^?) \]
\[ \text{AccTab} \ (u^?)^' = \text{AccTab} \ (u) \cup \{t^?, \mathbb{P}\text{rights}^? \} \]
\[ \text{Auhte}^' = \text{Auhte} \]
\[ \text{UserAutho}^' = \text{UserAutho} \]
\[ \text{Tablist}^' = \text{Tablist} \]
\[ \text{Matrix}^' = \text{Matrix} \]
\[ m^! = \text{Table Rights Given Successfully} \]

**UserDoesNotExist**

\[ \Xi \text{Sys\_State} \]
\[ u^? : \text{users}; \ client^? : \text{users}; t^? : \text{Tablist}; \ rights : \mathbb{P}\text{mode}; m^! : \text{Message} \]

\[ u^? \notin \text{Auhte} \lor u^? \notin \text{dom UserAutho} \]
\[ m^! = \text{User Does Not Exists} \]

**ClientDoesNotExist**

\[ \Xi \text{Sys\_State} \]
\[ u^? : \text{users}; \ client^? : \text{users}; t^? : \text{Tablist}; \ rights : \mathbb{P}\text{mode}; m^! : \text{Message} \]

\[ \text{client}^? \notin \text{Auhte} \lor \text{client}^? \notin \text{UserAutho} \]
\[ m^! = \text{Client Does Not Exists} \]

**TableDoesNotExist**

\[ \Xi \text{Sys\_State} \]
\[ u^? : \text{users}; \ client^? : \text{users}; t^? : \text{Tablist}; \ rights : \mathbb{P}\text{mode}; m^! : \text{Message} \]

\[ t^? \notin \text{Tablist} \]
\[ m^! = \text{Table Does Not Exists} \]
InvalidRights
Ξ Sys_State
u?: users; client?:users; t?:Tablist; rights:ℙmode; m!:Message

Table ∈ Matrix(u?)
m! = Invalid Rights

GiveTableRights ∆ AddTableRights ∨ UserDoesNotExists ∨ ClientDoesNotExists ∨ TableDoesNotExists ∨ InvalidRights

MatrixAddColumn
Δ Sys_State
u?:users; rights?: ℙmatautho; t?:Tablist; col?:COLUMN; rec?:ℙRECORD; m!=Message

u? ∈ Auhte
u? ∈ dom UserAutho
t?: ∈ Tablist
u? = owner (token t?)
Table ∈ Matrix(u?)
col? ∈ dom TabDetails(t?)
TabDetails(t?)′ = TabDetails(t) ∪ {col?,ℙr?}
m! = Column Added Successfully to the table

MatrixAddColumnUserNotExists
Ξ Sys_State
u?:users; rights?: ℙmatautho; t?:Tablist; col?:COLUMN; rec?:ℙRECORD; m!=Message

u? ∉ Auhte ∨ u? ∉ dom UserAutho
m! = User Does Not Exists

MatrixAddColumnTableDoesNotExists
Ξ Sys_State
u?:users; rights?: ℙmatautho; t?:Tablist; col?:COLUMN; rec?:ℙRECORD; m!=Message
t? ∉ Tablist
m! = Table Does Not Exists
MatrixAddColumnInvalidRights

Ξ Sys_State
u?:users; rights?: ℙmatautho; t?:Tablist; col?:COLUMN; rec?:ℙ RECORD; m!=Message

Table ∈ Matrix(u?)
m! = Invalid Rights

MatrixAddColumnAlreadyExists

Ξ Sys_State
u?:users; rights?: ℙmatautho; t?:Tablist; col?:COLUMN; rec?:ℙ RECORD; m!=Message
col? ∈ dom TabDetails(t?)
m! = Column Already Exists

ProtectedMatrixAddColumn

ΔProDB
GetToken
MatrixAddColumn
FreeToken

AddColumn ≜ MatrixAddColumn ∨ MatrixAddColumnUserNotExists ∨ MatrixAddColumnTableDoesNotExists ∨ MatrixAddColumnInvalidRights ∨ MatrixAddColumnAlreadyExists ∨ ProtectedMatrixAddColumn

MatrixDeleteColumn

ΔSys_State
u?:users; t?: ℙmatautho; t?:Tablist; col?:cols; rec?:ℙrecs; m!=Message

u? ∈ Auhte
u? ∈ dom UserAutho
t?: ∈ Tablist
u? = owner (token t?)
Table ∈ Matrix(u?)
col? ∈ dom TabDetails(t?)
TabDetails(t?)′ = TabDetails(t) ⊖ {col?, ℙ rec}’
m! = Column Deleted Successfully to the table
MatrixDeleteColumnUserNotExists

\[ \exists \text{Sys\_State} \]
\[ u?:\text{users}; r?:\text{Pmatautho}; t?:\text{Tablist}; \text{col}?:\text{cols}; \text{rec}?:\text{Pres}; \text{m!}=\text{Message} \]
\[ u\notin \text{Authe} \lor u\notin \text{dom UserAutho} \]
\[ \text{m!} = \text{User Does Not Exists} \]

MatrixDeleteColumnTableDoesNotExists

\[ \exists \text{Sys\_State} \]
\[ u?:\text{users}; r?:\text{Pmatautho}; t?:\text{Tablist}; \text{col}?:\text{cols}; \text{rec}?:\text{Pres}; \text{m!}=\text{Message} \]
\[ t\notin \text{Tablist} \]
\[ \text{m!} = \text{Table Does Not Exists} \]

MatrixDeleteColumnInvalidRights

\[ \exists \text{Sys\_State} \]
\[ u?:\text{users}; r?:\text{Pmatautho}; t?:\text{Tablist}; \text{col}?:\text{cols}; \text{rec}?:\text{Pres}; \text{m!}=\text{Message} \]
\[ \text{Table} \notin \text{Matrix}(u?) \]
\[ \text{m!} = \text{Invalid Rights} \]

MatrixDeleteColumnDoesNotExists

\[ \exists \text{Sys\_State} \]
\[ u?:\text{users}; r?:\text{Pmatautho}; t?:\text{Tablist}; \text{col}?:\text{cols}; \text{rec}?:\text{Pres}; \text{m!}=\text{Message} \]
\[ \text{col}\notin \text{dom TabDetails}(t?) \]
\[ \text{m!} = \text{Column Does Not Exists} \]

ProtectedMatrixDeleteColumn

\[ \Delta \text{ProDB} \]
\[ \text{GetToken} \]
\[ \text{MatrixDeleteColumn} \]
\[ \text{FreeToken} \]

\[ \text{DeleteColumn} \triangleq \text{MatrixDeleteColumn} \lor \text{MatrixDeleteColumnUserNotExists} \lor \text{MatrixDeleteColumnTableDoesNotExists} \lor \text{MatrixDeleteColumnInvalidRights} \lor \text{MatrixDeleteColumnDoesNotExists} \lor \text{ProtectedMatrixDeleteColumn} \]
MatrixDeleteTable

| ΔSys_State
| u?:users; rights?:ℙmatautho; t?:Tablist; m!=Message

| u? ∈ Auth
| u? ∈ dom UserAutho
| t?: ∈ Tablist
| u? = owner (token t?)
| Table ∈ Matrix(u?)
| Tablist = Tablist’ ∪ {t?}

MatrixDeleteTableUserNotExists

| ΞSys_State
| u?:users; rights?:ℙmatautho; t?:Tablist; m!=Message

| u? ∉ Auth \lor u? ∉ dom UserAutho
| m! = User Does Not Exists

MatrixDeleteTableDoesNotExists

| ΞSys_State
| u?:users; rights?:ℙmatautho; t?:Tablist; m!=Message

| t? ∉ Tablist
| m! = Table Does Not Exists

MatrixDeleteTableInvalidRights

| ΞSys_State
| u?:users; rights?:ℙmatautho; t?:Tablist; m!=Message

| Table ∉ Matrix(u?)
| m! = Invalid Rights

ProtectedMatrixDeleteTable

| ΔProDB
| GetToken
| MatrixDeleteTable
| FreeToken

DeleteTable ≜ MatrixDeleteTable \lor MatrixDeleteTableUserNotExists \lor MatrixDeleteTableDoesNotExists \lor MatrixDeleteTableInvalidRights \lor ProtectedMatrixDeleteTable
APPENDIX B

Complete Specifications for Table Manipulation Operations for Table Level Security

\[
\begin{align*}
\text{[USER]} & \quad \text{[TABLE]} \\
\text{[COLUMN]} & \quad \text{[RECORD]} \\
\text{[Authenticated]} & \quad \text{[Message]} \\
\text{[TOKEN]} & \quad \text{MODE ::= Read | Edit | Insert | Delete} \\
\text{UAutho ::= Add | Delete} \\
\text{MatAutho ::= Mat | Table | User} \\
\text{token : Tablist → TOKEN}
\end{align*}
\]
Sys_State

users: PUSER
mode: PMODE
Autho: PUSER
Tablist: PTABLE
cols: PCOLUMN
recs: P RECORD
mode: PMODE
matautho: PMatAutho
userautho: PUAutho
TabCols: cols → Precs
TabDetail: Tablist → TabCols
UserAutho: users → PAuthenticate
Matrix: users → PMatAutho
AccTabAutho: Tablist → Pmode
AccTab: users → AccTabAutho
AccUserAutho: users → Puserautho
AccUser: users → AccUserAutho

dom UserAutho ⊆ users
dom Matrix ⊆ users
dom TabDetails ⊆ Tablist
dom TabCols ⊆ cols
dom AccTabAutho ⊆ Tablist
dom AccTab ⊆ users
dom AccUser ⊆ users
dom AccUserAutho ⊆ users

ProDB

Sys_State

owner: TOKEN → users

GetToken

ΔProDB

u?: users; t?: TOKEN

u?: ∈ dom owner

owner' = owner ⊕ {t?, u?}
\[\text{FreeToken}\]
\[\Delta \text{ProDB}\]
\[u^? : \text{users}; t^? : \text{TOKEN}\]
\[u^? = \text{owner } t^?\]
\[\text{owner } = \{t^?\} \oplus \text{owner}\]

\[\text{ReadSuccessful}\]
\[\Xi \text{Sys\_State}\]
\[u^? : \text{users}; t^? : \text{Tablist}; m^!= \text{Message}\]
\[u^? \in \text{Authe}\]
\[u^? \in \text{UserAutho}\]
\[t^? \in \text{Tablist}\]
\[u^? = \text{owner (token } t^?)\]
\[\{t^?, \text{Read}\} \in \text{AccTab}(u^?)\]
\[\text{TabDetails}' = \text{TabDetails}\]
\[\text{Tablist}' = \text{Tablist}\]
\[\text{AccTab}' = \text{AccTab}\]
\[\text{Authe}' = \text{Authe}\]
\[\text{UserAutho}' = \text{UserAutho}\]
\[m^! = \text{Read Successful}\]

\[\text{UserDoesNotExists}\]
\[\Xi \text{Sys\_State}\]
\[u^? : \text{users}; t^? : \text{Tablist}; m^!= \text{Message}\]
\[u^? \notin \text{Authe} \lor u^? \notin \text{UserAutho}\]
\[m^! = \text{User Does Not Exists}\]

\[\text{TableNotExists}\]
\[\Xi \text{Sys\_State}\]
\[u^? : \text{users}; t^? : \text{Tablist}; m^!= \text{Message}\]
\[t^? \notin \text{Tablist}\]
\[m^! = \text{Table Does Not Exists}\]
InvalidRights

\[ \exists \text{Sys} \_\text{State} \]
\[ u?: \text{users}; t?: \text{Tablist}; m!=\text{Message} \]

\[ \{t?, \text{Read}\} \notin \text{AccTab}(u?) \]
\[ m! = \text{Invalid Rights} \]

ProtectedReadSuccessful

\[ \Delta \text{ProDB} \]
\[ \text{GetToken} \]
\[ \text{ReadSuccessful} \]
\[ \text{FreeToken} \]

\[ \text{ReadTable} \equiv \text{ReadSuccessful} \lor \text{UserDoesNotExists} \lor \text{TableDoesNotExists} \lor \text{InvalidRights} \lor \text{ProtectedReadSuccessful} \]

EditSuccessful

\[ \Delta \text{Sys} \_\text{State} \]
\[ u?: \text{users}; t?: \text{Tablist}; m!=\text{Message}; \ col?: \text{cols}; \ rec?: \text{recs} \]

\[ u? \in \text{Authe} \]
\[ u? \in \text{UserAutho} \]
\[ t? \in \text{Tablist} \]
\[ u? = \text{owner (token t?)} \]
\[ \{t?, \text{Edit}\} \in \text{AccTab}(u?) \]
\[ \{\text{col?}, \text{rec}\} \in \text{TabDetails}(t?) \]
\[ \text{TabDetails}' = \text{TabDetails}(t?) \oplus \{\text{col?}, \text{rec}\} \]
\[ \text{Tablist}' = \text{Tablist} \]
\[ \text{AccTab}' = \text{AccTab} \]
\[ \text{Authe}' = \text{Authe} \]
\[ \text{UserAutho}' = \text{UserAutho} \]
\[ m! = \text{Read Successful} \]

UserDoesNotExists

\[ \exists \text{Sys} \_\text{State} \]
\[ u?: \text{users}; t?: \text{Tablist}; m!=\text{Message}; \ col?: \text{cols}; \ rec?: \text{recs} \]

\[ u? \notin \text{Authe} \lor u? \notin \text{UserAutho} \]
\[ m! = \text{User Does Not Exists} \]
TableDoesNotExists

\[ \Xi \text{Sys\_State} \\
\text{users} \vdash \text{message}; \text{cols} \vdash \text{recs} \\
\text{tablist} \not\in \text{tablist} \\
m! = \text{Table Does Not Exists} \]

InvalidRights

\[ \Xi \text{Sys\_State} \\
\text{users} \vdash \text{message}; \text{cols} \vdash \text{recs} \\
\{t?\',\text{Edit}\} \not\in \text{AccTab}(u?) \\
m! = \text{Invalid Rights} \]

ColRecNotFound

\[ \Xi \text{Sys\_State} \\
\text{users} \vdash \text{message}; \text{cols} \vdash \text{recs} \\
\{\text{col?},\text{rec}\} \not\in \text{TabDetails}(t?) \\
m! = \text{Column Record Not Found} \]

ProtectedEditSuccessful

\[ \Delta \text{ProDB} \\
\text{GetToken} \\
\text{EditSuccessful} \\
\text{FreeToken} \]

EditUpdateTable \(=\) \text{EditSuccessful} \(\lor\) \text{UserDoesNotExist} \(\lor\) \text{TableDoesNotExist} \(\lor\) \text{InvalidRights} \(\lor\) \text{ColRecNotFound} \(\lor\) \text{ProtectedEditSuccessful}
\textbf{InsertSuccessful}

$\Delta \text{Sys\_State}$

$\exists \, u^? \in \text{users} , \, t^? \in \text{Tablist} \implies m^!=\text{Message} , \, \text{col}^?:\text{cols} , \, \text{rec}^?:\text{RECORD}$

$u^? \in \text{UserAutho} \implies m^!=\text{Insert Successful}$

\textbf{UserDoesNotExists}

$\Xi \text{Sys\_State}$

$\exists \, u^? \in \text{users} , \, t^? \in \text{Tablist} \implies m^!=\text{Message} , \, \text{col}^?:\text{cols} , \, \text{rec}^?:\text{RECORD}$

$u^? \notin \text{UserAutho} \lor u^? \notin \text{UserAutho} \implies m^!=\text{User Does Not Exists}$

\textbf{TableDoesNotExists}

$\Xi \text{Sys\_State}$

$\exists \, u^? \in \text{users} , \, t^? \in \text{Tablist} \implies m^!=\text{Message} , \, \text{col}^?:\text{cols} , \, \text{rec}^?:\text{RECORD}$

$t^? \notin \text{Tablist} \implies m^!=\text{Table Does Not Exists}$

\textbf{InvalidRights}

$\Xi \text{Sys\_State}$

$\exists \, u^? \in \text{users} , \, t^? \in \text{Tablist} \implies m^!=\text{Message} , \, \text{col}^?:\text{cols} , \, \text{rec}^?:\text{RECORD}$

$\{t^?, \text{Insert}\} \notin \text{AccTab}(u^?) \implies m^!=\text{Invalid Rights}$
\textbf{ColumnNotFound} \\
\[ \Xi_{\text{Sys\_State}} \]
\[ u?::\text{users}; t?::\text{Tablist}; m!=\text{Message}; \text{col}?::\text{cols}; \text{rec}?::\text{RECORD} \]
\[ \text{col}? \notin \text{dom} \text{TabDetails}(t?) \]
\[ m! = \text{Column Does Not Exists} \]

\textbf{ProtectedInsertSuccessful} \\
\[ \Delta_{\text{ProDB}} \]
\[ \text{GetToken} \]
\[ \text{InsertSuccessful} \]
\[ \text{FreeToken} \]

\[ \text{InsertData} \triangleq \text{InsertSuccessful} \lor \text{UserDoesNotExist} \lor \text{TableDoesNotExist} \lor \text{InvalidRights} \lor \text{ColumnNotFound} \lor \text{ProtectedInsertSuccessful} \]

\textbf{DeleteSuccessful} \\
\[ \Delta_{\text{Sys\_State}} \]
\[ u?::\text{users}; t?::\text{Tablist}; m!=\text{Message}; \text{col}?::\text{cols}; \text{rec}?::\text{recs} \]
\[ u? \in \text{Authe} \]
\[ u? \in \text{UserAutho} \]
\[ t? \in \text{Tablist} \]
\[ u? = \text{owner (token} \ t?) \]
\[ \{ t?, \text{Delete}\} \in \text{AccTab}(u?) \]
\[ \text{col}? \in \text{dom} \text{TabDetails}(t?) \]
\[ \text{rec}? \in \text{ran} \text{TabDetails}(t?) \]
\[ \text{TabDetails}' = \text{TabDetails}(t?) \oplus \{ \text{col}?, \text{rec}? \} \]
\[ \text{Tablist}' = \text{Tablist} \]
\[ \text{AccTab}' = \text{AccTab} \]
\[ \text{Authe}' = \text{Authe} \]
\[ \text{UserAutho}' = \text{UserAutho} \]
\[ m! = \text{Delete Successful} \]
\begin{align*}
\text{UserDoesNotExists} & \quad \forall \text{Sys\_State} \\
& \quad \forall u : \text{users}; t : \text{Tablist}; m = \text{Message}; c : \text{cols}; r : \text{recs} \\
& \quad u \notin \text{Auth} \lor u \notin \text{UserAutho} \\
& \quad m = \text{User Does Not Exists} \\
\text{TableDoesNotExists} & \quad \forall \text{Sys\_State} \\
& \quad \forall u : \text{users}; t : \text{Tablist}; m = \text{Message}; c : \text{cols}; r : \text{recs} \\
& \quad t \notin \text{Tablist} \\
& \quad m = \text{Table Does Not Exists} \\
\text{InvalidRights} & \quad \forall \text{Sys\_State} \\
& \quad \forall u : \text{users}; t : \text{Tablist}; m = \text{Message}; c : \text{cols}; r : \text{recs} \\
& \quad \{t, \text{Delete}\} \not\in \text{AccTab}(u) \\
& \quad m = \text{Invalid Rights} \\
\text{ColumnNotFound} & \quad \forall \text{Sys\_State} \\
& \quad \forall u : \text{users}; t : \text{Tablist}; m = \text{Message}; c : \text{cols}; r : \text{recs} \\
& \quad c : \not\in \text{dom \ TabDetails}(t) \\
& \quad m = \text{Column Does Not Exists} \\
\text{RecordNotFound} & \quad \forall \text{Sys\_State} \\
& \quad \forall u : \text{users}; t : \text{Tablist}; m = \text{Message}; c : \text{cols}; r : \text{recs} \\
& \quad r : \not\in \text{ran \ TabDetails}(t) \\
& \quad m = \text{Record Does Not Exists} \\
\text{ProtectedDeleteSuccessful} & \quad \Delta \text{ProDB} \quad \text{GetToken} \quad \text{DeleteSuccessful} \quad \text{FreeToken} \\
\text{DeleteData} & \equiv \text{DeleteSuccessful} \lor \text{UserDoesNotExist} \lor \text{TableDoesNotExists} \lor \text{InvalidRights} \lor \text{ColumnNotFound} \lor \text{RecordNotFound} \lor \text{ProtectedDeleteSuccessful}
\end{align*}
APPENDIX C

Complete Specifications for Column Level Security

```
[USER]
[TABLE]
[COLUMN]
[RECORD]
[Authenticated]
[Message]
[TOKEN]

MODE ::= Read | Edit | Insert | Delete

UAutho ::= Add | Delete

MatAutho ::= Mat | Table | User

| token : Tablist | → | TOKEN |
```
Sys_State

users: \mathbb{P}\text{USER}
mode: \mathbb{P}\text{MODE}
Authe: \mathbb{P}\text{USER}
Tablist: \mathbb{P}\text{TABLE}
cols: \mathbb{P}\text{COLUMN}
recs: \mathbb{P}\text{RECORD}
userautho: \mathbb{P}\text{UAutho}
Authenticate: \mathbb{P}\text{AUTHENTICATE}
matautho: \mathbb{P}\text{MatAutho}
ColAutho: cols \rightarrow \mathbb{P}\text{mode}
TabColAutho: Tablist \rightarrow \mathbb{P}\text{ColAutho}
AccTabCol: users \rightarrow TabColAutho
TabCols: cols \rightarrow \mathbb{P}\text{recs}
TabDetail: Tablist \rightarrow \mathbb{P}\text{TabCols}
UserAutho: users \rightarrow \mathbb{P}\text{Authenticate}
Matrix: users \rightarrow \mathbb{P}\text{matautho}
AccUserAutho: users \rightarrow \mathbb{P}\text{userautho}
AccUser: users \rightarrow \mathbb{P}\text{AccUserAutho}

\text{dom } UserTabColMode \subseteq \text{users}
\text{dom } TabColMode \subseteq \text{TabList}
\text{dom } UserAutho \subseteq \text{users}
\text{dom } Matrix \subseteq \text{users}
\text{dom } AccTabCol \subseteq \text{users}
\text{dom } TabColAutho \subseteq \text{TabList}
\text{dom } ColAutho \subseteq \text{cols}
\text{dom } TabDetails \subseteq \text{Tablist}
\text{dom } TabCols \subseteq \text{cols}
\text{dom } AccUser \subseteq \text{users}

ProDB

Sys_State

owner: \text{TOKEN} \rightarrow \text{users}
GetToken
\[\upDelta ProDB\]
\[u? : \text{users}; t?:\text{TOKEN}\]
\[u? \in \text{dom owner}\]
\[\text{owner}' = \text{owner} \oplus \{t?, u?\}\]

FreeToken
\[\upDelta ProDB\]
\[u? : \text{USER}; t?:\text{TOKEN}\]
\[u? = \text{owner t?}\]
\[\text{owner} = \{t?\} \boxplus \text{owner}\]

AddTableRights
\[\upDelta \text{Sys\_State}\]
\[u? : \text{users}; \text{client}? : \text{users}; t?:\text{Tablist}; \text{col}? : \text{cols}; \text{rights}? : \mathbb{P}\text{mode}; m!:\text{Message}\]
\[u? \in \text{Auth}\]
\[u? \in \text{dom UserAutho}\]
\[\text{client}? \in \text{Auth}\]
\[\text{client}? \in \text{dom UserAutho}\]
\[t? \in \text{Tablist}\]
\[\text{col}? \in \text{dom TabDetails}(t?)\]
\[\text{Table} \in \text{Matrix}(u?)\]
\[\text{if } t? \in \text{AccTabCol(client?)}\]
\[\text{then } \text{AccTabCol(client?)'} = \text{AccTabCol(client?)} \cup \{t?, (\text{col?}, \mathbb{P}\text{rights?})\}\]
\[\text{else ( if c? \notin \text{AccTabCol(client?, t?)}\]
\[\text{then } \text{AccTabCol(client?, t?)'} = \text{AccTabCol(client?, t?)} \cup \{\text{col?}, \mathbb{P}\text{rights?}\}\]
\[\text{else } \text{AccTabCol(client?, t?, \text{col?})'} = \text{AccTabCol(client?, t?, \text{col?})} \cup \{\mathbb{P}\text{rights?}\}\]
\[\text{TabDetails}' = \text{TabDetails}\]
\[\text{Auth}' = \text{Auth}\]
\[\text{UserAutho}' = \text{UserAutho}\]
\[\text{Tablist}' = \text{Tablist}\]
\[\text{Matrix}' = \text{Matrix}\]
\[m! = \text{Table Rights Given Successfully}\]
UserDoesNotExists

\[ u? \in \text{Auhte} \lor u? \notin \text{dom UserAutho} \]
\[ m! = \text{User Does Not Exists} \]

ClientDoesNotExists

\[ u?: \text{users}; \; \text{client?}:\text{users}; \; t?:\text{Tablist}; \; \text{col?}:\text{cols}; \; \text{rights?}:\mathbb{P}\text{mode}; \; m!:\text{Message} \]
\[ \text{client?} \notin \text{Auhte} \lor \text{client?} \notin \text{UserAutho} \]
\[ m! = \text{Client Does Not Exists} \]

TableDoesNotExists

\[ u?: \text{users}; \; \text{client?}:\text{users}; \; t?:\text{Tablist}; \; \text{col?}:\text{cols}; \; \text{rights?}:\mathbb{P}\text{mode}; \; m!:\text{Message} \]
\[ t? \notin \text{Tablist} \]
\[ m! = \text{Table List Does Not Exists} \]

InvalidRights

\[ u?: \text{users}; \; \text{client?}:\text{users}; \; t?:\text{Tablist}; \; \text{col?}:\text{cols}; \; \text{rights?}:\mathbb{P}\text{mode}; \; m!:\text{Message} \]
\[ \text{Table} \notin \text{Matrix}(u?) \]
\[ m! = \text{Invalid Rights} \]

ColumnDoesNotExists

\[ u?: \text{users}; \; \text{client?}:\text{users}; \; t?:\text{Tablist}; \; \text{col?}:\text{cols}; \; \text{rights?}:\mathbb{P}\text{mode}; \; m!:\text{Message} \]
\[ \text{col?} \notin \text{dom TabDetails}(t?) \]
\[ m! = \text{Column Does Not Exists} \]

GiveTableRights \( \triangleq \) AddTableRights \lor UserDoesNotExists \lor ClientDoesNotExists \lor TableDoesNotExists \lor InvalidRights \lor ColumnDoesNotExists
ReadSuccessful

\[ \exists \text{Sys\_State} \]
\[ u?\in \text{users}; \, t?:\text{Tablist}; \, col?:\text{cols}; \, m!=\text{Message} \]
\[ u? \in \text{Authe} \]
\[ u? \in \text{UserAutho} \]
\[ t? \in \text{Tablist} \]
\[ u? = \text{owner (token t?)} \]
\[ \text{col}\in \text{dom TabDetails}(t?) \]
\[ \{\text{col}, \text{Read}\}\in \text{AccTabCol}(u?, t?) \]
\[ \text{AccTabCol}' = \text{AccTabCol} \]
\[ \text{TabDetails}' = \text{TabDetails} \]
\[ \text{Tablist}' = \text{Tablist} \]
\[ \text{AccTab}' = \text{AccTab} \]
\[ \text{Authe}' = \text{Authe} \]
\[ \text{UserAutho}' = \text{UserAutho} \]
\[ m! = \text{Read Successful} \]

UserDoesNotExists

\[ \exists \text{Sys\_State} \]
\[ u?\in \text{users}; \, t?:\text{Tablist}; \, col?:\text{cols}; \, m!=\text{Message} \]
\[ u? \notin \text{Authe} \lor u? \notin \text{UserAutho} \]
\[ m! = \text{USER Does Not Exists} \]

TableNotExists

\[ \exists \text{Sys\_State} \]
\[ u?\in \text{users}; \, t?:\text{Tablist}; \, col?:\text{cols}; \, m!=\text{Message} \]
\[ t? \notin \text{Tablist} \]
\[ m! = \text{Table Does Not Exists} \]

InvalidRights

\[ \exists \text{Sys\_State} \]
\[ u?\in \text{users}; \, t?:\text{Tablist}; \, col?:\text{cols}; \, m!=\text{Message} \]
\[ \{\text{col}, \text{Read}\}\notin \text{AccTabCol}(u?, t?) \]
\[ m! = \text{Invalid Rights} \]
\[ \text{ColumnDoesNotExists} \]
\[ \exists \text{ Sys \_State} \]
\[ u^?: \text{ users}; t^?: \text{ Tablist}; \text{ col}?: \text{ cols}; m^!: \text{ Message} \]
\[ \text{ col}^? \notin \text{ dom TabDetails}(t^?) \]
\[ m^! = \text{ Column Does Not Exists} \]

\[ \text{ProtectedReadSuccessful} \]
\[ \Delta \text{ ProDB} \]
\[ \text{ GetToken} \]
\[ \text{ ReadSuccessful} \]
\[ \text{ FreeToken} \]

\( \text{ ReadTable} \triangleq \text{ ReadSuccessful} \lor \text{ UserDoesNotExists} \lor \text{ TableDoesNotExists} \lor \text{ InvalidRights} \lor \text{ ColumnDoesNotExists} \lor \text{ ProtectedReadSuccessful} \)

\[ \text{EditSuccessful} \]
\[ \Delta \text{ Sys \_State} \]
\[ u^?: \text{ users}; t^?: \text{ Tablist}; m^! = \text{ Message}; \text{ col}?: \text{ cols}; \text{ rec}?: \text{ recs} \]
\[ u^? \in \text{ Authe} \]
\[ u^? \in \text{ UserAutho} \]
\[ t^? \in \text{ Tablist} \]
\[ u^? = \text{ owner (token } t^?) \]
\[ \{ \text{ col}?: \text{ rec}? \} \in \text{ TabDetails}(t^?) \]
\[ \{ \text{ col}?: \text{ Edit}\} \in \text{ AccTabCol}(u^?, t^?) \]
\[ \text{ TabDetails}' = \text{ TabDetails}(t^?) \oplus \{ \text{ col}?: \text{ rec}? \} \]
\[ \text{ AccTabCol}' = \text{ AccTabCol} \]
\[ \text{ Tablist}' = \text{ Tablist} \]
\[ \text{ AccTab}' = \text{ AccTab} \]
\[ \text{ Authe}' = \text{ Authe} \]
\[ \text{ UserAutho}' = \text{ UserAutho} \]
\[ m^! = \text{ Read Successful} \]

\[ \text{UserDoesNotExists} \]
\[ \exists \text{ Sys \_State} \]
\[ u^?: \text{ users}; t^?: \text{ Tablist}; m^! = \text{ Message}; \text{ col}?: \text{ cols}; \text{ rec}?: \text{ recs} \]
\[ u^? \notin \text{ Authe} \lor u^? \notin \text{ UserAutho} \]
\[ m^! = \text{ User Does Not Exists} \]
TableDoesNotExists

\[\exists \text{Sys}\_\text{State} \quad u?:\text{users}; \ t?:\text{Tablist}; \ m!=\text{Message}; \ col?:\text{cols}; \ rec?:\text{recs} \]
\[t? \notin \text{Tablist} \]
\[m!=\text{Table Does Not Exists} \]

InvalidRights

\[\exists \text{Sys}\_\text{State} \quad u?:\text{users}; \ t?:\text{Tablist}; \ m!=\text{Message}; \ col?:\text{cols}; \ rec?:\text{recs} \]
\[\{\text{col?},\text{Edit}\} \notin \text{AccTabCol}(u?,t?) \]
\[m!=\text{Invalid Rights} \]

ColRecNotFound

\[\exists \text{Sys}\_\text{State} \quad u?:\text{USER}; \ t?:\text{TABLE}; \ m!=\text{Message}; \ col?:\text{cols}; \ rec?:\text{recs} \]
\[\{\text{col?},\text{rec}\} \notin \text{TabDetails}(t?) \]
\[m!=\text{Column Record Not Found} \]

ProtectedEditSuccessful

\[\Delta \text{ProDB} \quad \text{GetToken} \quad \text{EditSuccessful} \quad \text{FreeToken} \]

\[\text{EditUpdateTable} \triangleq \text{EditSuccessful} \lor \text{UserDoesNotExist} \lor \text{TableDoesNotExist} \lor \text{InvalidRights} \lor \text{ColRecNotFound} \lor \text{ProtectedEditSuccessful} \]
InsertSuccessful

\[ \Delta_{\text{Sys\_State}} \]
\[ u'?:\text{users}; t?:\text{Tablist}; m! = \text{Message}; col?:\text{cols}; \text{rec}?:\text{RECORD} \]
\[ u' \in \text{Authe} \]
\[ u' \in \text{UserAutho} \]
\[ t' \in \text{Tablist} \]
\[ u' = \text{owner (token } t)? \]
\[ \text{col}' \in \text{dom } \text{TabDetails}(t') \]
\[ \{\text{col}',\text{Insert}\} \in \text{AccTabCol}(u',t') \]
\[ \text{TabDetails}' = \text{TabDetails}(t') \cup \{\text{col}',\text{rec}'\} \]
\[ \text{AccTabCol}' = \text{AccTabCol} \]
\[ \text{Tablist}' = \text{Tablist} \]
\[ \text{AccTab}' = \text{AccTab} \]
\[ \text{Authe}' = \text{Authe} \]
\[ \text{UserAutho}' = \text{UserAutho} \]
\[ m! = \text{Insert Successful} \]

UserDoesNotExists

\[ \exists_{\text{Sys\_State}} \]
\[ u'?:\text{users}; t?:\text{Tablist}; m! = \text{Message}; col?:\text{cols}; \text{rec}?:\text{RECORD} \]
\[ u' \notin \text{Authe} \lor u' \notin \text{UserAutho} \]
\[ m! = \text{User Does Not Exists} \]

TableDoesNotExists

\[ \exists_{\text{Sys\_State}} \]
\[ u'?:\text{users}; t?:\text{Tablist}; m! = \text{Message}; col?:\text{cols}; \text{rec}?:\text{RECORD} \]
\[ t' \notin \text{Tablist} \]
\[ m! = \text{Table Does Not Exists} \]

InvalidRights

\[ \exists_{\text{Sys\_State}} \]
\[ u'?:\text{users}; t?:\text{Tablist}; m! = \text{Message}; col?:\text{cols}; \text{rec}?:\text{RECORD} \]
\[ \{\text{col}',\text{Insert}\} \in \text{AccTabCol}(u',t') \]
\[ m! = \text{Invalid Rights} \]
ColumnNotFound

\[ \exists \text{Sys\_State} \]
\[ u?:\text{users}; \ t?:\text{Tablist}; \ m!:\text{Message}; \ col?:\text{cols}; \ rec?:\text{RECORD} \]
\[ \text{col}\? \notin \text{dom \ TabDetails}(t?) \]
\[ m! = \text{Column Does Not Exists} \]

ProtectedInsertSuccessful

\[ \Delta \text{ProDB} \]
\[ \text{GetToken} \]
\[ \text{InsertSuccessful} \]
\[ \text{FreeToken} \]

InsertData \triangleq \text{InsertSuccessful} \lor \text{UserDoesNotExist} \lor \text{TableDoesNotExist} \lor \text{InvalidRights} \lor \text{ColumnNotFound} \lor \text{ProtectedInsertSuccessful}

DeleteSuccessful

\[ \Delta \text{Sys\_State} \]
\[ u?:\text{users}; \ t?:\text{Tablist}; \ m!:\text{Message}; \ col?:\text{cols}; \ rec?:\text{recs} \]
\[ u? \in \text{Authe} \]
\[ u? \in \text{UserAutho} \]
\[ t? \in \text{Tablist} \]
\[ u? = \text{owner (token t?)} \]
\[ \{\text{col?}, \text{rec}\}\in \text{TabDetails}(t?) \]
\[ \{\text{col?}, \text{Delete}\}\in \text{AccTabCol}(u?, t?) \]
\[ \text{TabDetails}' = \text{TabDetails}(t?) \oplus \{\text{col?} \odot \text{rec}\}\]
\[ \text{AccTabCol}' = \text{AccTabCol} \]
\[ \text{Tablist}' = \text{Tablist} \]
\[ \text{AccTab}' = \text{AccTab} \]
\[ \text{Authe}' = \text{Authe} \]
\[ \text{UserAutho}' = \text{UserAutho} \]
\[ m! = \text{Delete Successful} \]
UserDoesNotExist$
\forall \text{Sys\_State}$
$u?\!:\text{users}; t?\!:\text{Tablist}; m!\!:\text{Message}; \text{col}?\!:\text{cols}; \text{rec}?\!:\text{recs}$
$u? \not\in \text{Auth}\lor u? \not\in \text{UserAuth}
\text{m!} = \text{User Does Not Exists}$

TableDoesNotExist$
\forall \text{Sys\_State}$
$u?\!:\text{users}; t?\!:\text{Tablist}; m!\!:\text{Message}; \text{col}?\!:\text{cols}; \text{rec}?\!:\text{recs}$
$t? \not\in \text{Tablist}
\text{m!} = \text{Table Does Not Exists}$

InvalidRights$
\forall \text{Sys\_State}$
$u?\!:\text{users}; t?\!:\text{Tablist}; m!\!:\text{Message}; \text{col}?\!:\text{cols}; \text{rec}?\!:\text{recs}$
$\{\text{col}?\!:\text{Delete}\}\not\in \text{AccTabCol}(u?, t?)
\text{m!} = \text{Invalid Rights}$

ColRecNotFound$
\forall \text{Sys\_State}$
$u?\!:\text{users}; t?\!:\text{Tablist}; m!\!:\text{Message}; \text{col}?\!:\text{cols}; \text{rec}?\!:\text{recs}$
$\{\text{col}?\!:\text{rec}?\} \not\in \text{TabDetails}(t?)
\text{m!} = \text{Record Does Not Exists}$

ProtectedDeleteSuccessful
$\Delta \text{ProDB}$
GetToken
DeleteSuccessful
FreeToken

DeleteData \triangleq \text{DeleteSuccessful} \lor \text{UserDoesNotExist} \lor \text{TableDoesNotExist} \lor \text{InvalidRights} \lor \text{ColRecNotFound} \lor \text{ProtectedDeleteSuccessful}$
REFERENCES


