
Zhipeng Wang¹, Hong Zhu¹, Meiyi Xie¹

¹Computer School of Sci & Tec, Huazhong University of Sci & Tec, Wuhan, Hubei, China, 430074
{hkdwzp, whzhuhong}@gmail.com, xiemeiyi@sina.com

Abstract. Formal specification and verification is required for high security level DBMS in the top level specification design. The specification and verification towards SQL operations is important especially. In this paper, we propose a novel approach to solve the specification and verification issues towards SQL operations. Firstly, we formally define the SQL operations in FTLS; then, we give the definitions of the simple SQL operations and propose a method to verify them; finally, we transform the verification of the SQL operations in FTLS to the verification of the component simple SQL operations. The process of verification shows that our approach makes a comprehensive specification of SQL operations and simplifies the verification procedure.

Keywords: Formal Top Level Specification; SQL Operation; Formal Specification and Verification

1 Introduction

When developing a high security level (rated as B2 and above in TCSEC [1], or EAL5 and above in CC [2]) system, formal specification and verification are needed. And as the criterions require, when developing such a high security level DBMS, we need to make formal specification and verification in both Security Model and Formal Top Level Specification (FTLS). A secure DBMS is an extension of the traditional DBMS. The FTLS of the secure DBMS includes the formal specification and verification of SQL operations, which is important for verifying whether the implementation is consistent with security requirements. Nowadays, the SQL statements are more and more complex, and the specification and verification of SQL operations are therefore more difficult. So it is of great significance to propose an approach for the specification and verification of SQL operations in FTLS.

Teresa F.Lunt et al. have done a series of research from security model, FTLS to verification in SeaView project [3, 4]. However, because of the database technology limitations, they only researched simple SQL operations. For example, one can use a select statement to create a new table. They didn’t consider this situation, but only considered the simple create statement without any complex clauses. Nowadays, there is some research for specification and verification of SQL operations. But they
all have a main problem: no sufficient research on the specification and verification of complex SQL operations, which are very common in modern DBMS.

1.1 The Problems

The FTLS is designed on top of the security model. There are following problems in formal specification and verification of SQL operations in FTLS of a secure DBMS.

Firstly, SQL statements in modern DBMS are more complex, which increases the difficulty of verification. For example, a query may include multi-table join, nested subqueries, etc, which can access the database and change the database states. We need to record all these accesses in the specification. The complexity of the operations increases the complexity of specification and verification.

Secondly, as FTLS is designed on top of model, a trivial way for verification is to make a mapping between the model and the FTLS. Unfortunately, there exists the situation that one rule in the model maps multiple SQL operations. So it is a problem to make a comprehensive and clear specification of the SQL operations in FTLS.

Thirdly, most proof tools and languages, such as Gallina, Z, Isabella, PVS, etc. are not competent for complex structures [5-9]. We take the nested structures of different types as an example. Struct A includes struct B and struct C, and C includes A. This structure is hard for proof tools mentioned above to express for verification. However, it is common in complex SQL statements. The limitations of the proof tools make the verification problems more difficult.

1.2 Our Contributions

Focusing on the problems above, our work is to make a comprehensive and clear specification of the SQL operations in FTLS, and make it easier for verification using proof tools. Our contributions are as follows:

• We propose an approach for the specification of SQL operations in FTLS.
• On top of the specification, we give the definitions of the simple SQL operations, and propose a method to verify those simple SQL operations. Then we transform the verification of the SQL operations in FTLS to the verification of the component simple SQL operations.

The remainder of this paper is organized as follows. In Section 2, we introduce the FTLS and formally specify the SQL operations in FTLS. In Section 3, we verify the SQL operations in detail. In Section 4, we review the related work of the research. Finally, in Section 5 we conclude the paper.

2 Formal Top Level Specification for Secure DBMS

The Formal Top Level Specification (FTLS) for secure DBMS is specified in formal languages (Gallina, Z, etc.), which consists of three parts: System State, Safety Properties and SQL Operations.
2.1 System State

The DBMS is abstracted as a state machine [11]. The system state in FTLS includes the details of the system implementation, e.g. how to store access rights, present integrity constraint, etc. There are four elements in the system state \( \mathcal{VF} \): object set, data dictionary, user data and access set, i.e. \( \mathcal{VF} = (s_{\text{ObjectSet}}, s_{\text{DD}}, s_{\text{UD}}, s_{\text{B}}) \).

The object set \( s_{\text{ObjectSet}} \) represents the set of objects. \( o \in s_{\text{ObjectSet}} \) is an object, and can be a database, a schema, a table, a tuple, etc. The minimal granularity of the object for security level is tuple. The hierarchy of the objects is: database, schema, table, tuple make up a tree. \( s_{\text{DD}} \) is the description of the data dictionary, which stores the access rights, security level of subjects and objects, etc. \( s_{\text{UD}} \) records the values of user data, and is used for integrity constraint verification. Because the minimal granularity of the object is tuple, data items are not in the object set. \( s_{\text{B}} \) represents the records of accesses which are important for security analysis.

2.2 Safety Properties

The BLP model defines three security properties: Discretionary-Security Property, Simple-Security Property and Star-Security Property [10]. In addition, according to the secure DBMS features, there are three integrity constraints:

Object-Compatibility Property: a state \( v \) satisfies Object-Compatibility Property, if and only if in \( v \), if object \( o_2 \) is the father of object \( o_1 \), it must satisfy \( f_s(o_1) \geq f_s(o_2) \).

\( f_s(o) \) is the security level of the object \( o \). This property means the security level of an object must dominate the security level of its father. For example, the security levels of the tuples in a table must be equal or greater than the security level of this table. To a subject, if he/she can not access a table because of the low security level, he/she can neither access the tuples in this table.

Entity-Integrity Property: a state \( v \) satisfies Entity-Integrity Property, if and only if in \( v \), for any tuples \( o_1 \) and \( o_2 \) in an arbitrary table, their primary keys can not be NULL, and either \( f_s(o_1) \neq f_s(o_2) \), or their primary keys are different.

Here the security level is added into the traditional entity integrity. Subjects with different security levels can use the same primary key. This can avoid the covert channel caused by the primary key and ensure the safety of the information flow.

Reference-Integrity Property: a state \( v \) satisfies Reference-Integrity Property, if and only if in \( v \), for any tuples \( o_1 \) and \( o_2 \), if \( o_1 \) is referenced by \( o_2 \), then either \( o_2 \)'s foreign key is NULL, or its value is the same with \( o_1 \)'s primary key's value and \( f_s(o_1) = f_s(o_2) \).

The Reference-Integrity Property only allows the reference between tuples with the same security level. It is similar to the meaning of Entity-Integrity Property.

2.3 Security Definitions

**Definition 1. Safe State.** A safe state is a state \( v \) that satisfies all safety properties.

**Definition 2. Safe Operation.** If the state \( v_i \) is the pre-state of an operation and \( v_{i+1} \) is the post-state of the operation, and they are both safe states, then the operation is a safe operation.
**Definition 3. Safe System.** A safe system is a system in which all the states are safe, which means the initial state of the system $v_0$ is safe, and if any state $v_i$ that can be transited from $v_0$ is safe, after arbitrary operation, the post-state $v_{i+1}$ is also safe.

According to Definition 3, if we want to verify that the system is safe, we should first verify that the initial state of the system $v_0$ is safe. In FTLS, the initial state $v_0 = (O_0, DD_0, \Phi, \Phi)$, in which the user data and the access set are $\Phi$. Because the initial access set $s_B$ is $\Phi$, Simple-Security Property, Star-Security Property and Discretionary-Security Property are satisfied obviously. Because the initial user data $s_{UD}$ is $\Phi$, Entity-Integrity Property and Reference-Integrity Property are also satisfied. Because the only object in the initial objects set is the new created database, Object-Compatibility Property is satisfied. So the initial state of the system $v_0$ is safe.

### 2.4 SQL Operations

The operation set in FTLS is $OP = \{select\_op, insert\_op, update\_op, delete\_op, create\_op, alter\_op, drop\_op, grant\_op, revoke\_op, execute\_op\}$, in which there are all the SQL operations. Compared to the transition rules in security model [11], the SQL operations in FTLS are more detailed and closer to the DBMS implementation.

In the base of system state specification and initial state verification, we can verify the whole FTLS. The key step is to verify whether arbitrary SQL operation is safe, so it is important to specify the SQL operations. In this paper, considering the standard SQL syntax and the implementation of DBMS, we take $select$ operation as an example to specify the SQL operations in FTLS in the view of security. First, this operation is normal and representative for everyday use. Second, it is also typical from the view of verification.

**Definition 4. $s_{tc}$.** A first-in-last-out list whose elements are 2-tuple of $<t, c>$, which mean $<table, where\_clause>$. The list records all the tables and their where clauses for filtering tuples in select operation.

**Definition 5. select operation.** A select operation is specified by this 3-tuple of $<v, s, s_{tc}>$, in which: $v \in VF$ is a subject, whose information can be got from $s_{DD}$; $s_{tc}$ is as specified in Definition 4.

In the above definition, $v, s$ are single element and $s_{tc}$ is a linear list. This is easy to deal with for proof tools. But the SQL statements are often nested and complex themselves. It is a problem to transform the complex SQL statements to the specification as in Definition 5. However, because the transformation is not the key point of this paper, we won’t show the rules for transformation here.

### 3 Analysis and Verification for SQL Operations

As mentioned above, the specification as in Definition 5 only including linear list and single element is much easier to deal with for proof tools than the original nested and complex SQL statements. However, that’s not enough. As the verification process is very complex, we will do some further simplification. First, we give the definitions of simple SQL operations, and propose a method to verify those simple SQL operations. Then we transform the verification of the SQL operations in FTLS to the verification
of those component simple SQL operations. We also give the proof of the correctness of this approach. The same as above, we use \textit{select} operation as an example for the analysis and verification.

### 3.1 Analysis and Verification for Simple Select Operation

**Definition 6. simple select operation.** A simple select operation is specified by this 4-tuple of \(<v, s, t, c>\), in which: \(v\); \(s\) is a subject, whose information can be got from \(s\_DD\) in \(v\); \(t \in s\_ObjectSet\) whose type is table; \(c\) represents the \textit{where} clause for filtering tuples which includes only simple expression (contains no clause).

Figure 1 shows the specification of simple select operation rule.

Figure 1. Specification of Simple Select Operation Rule

\[\begin{align*}
&\text{if } \text{objType}(t) = \text{table} \land \text{HavePriv}(v, (s, t, \text{select}_opr)) = \text{true} \land \\
&f_r(v, x) \geq f_s(x, t) \\
&\text{then } \forall (o \in \text{includedby}(t) \land c(o) = \text{true} \land f_r(v, x) \geq f_s(x, o)), \\
&v^* = (s\_ObjectSet(v), s\_DD(v), s\_UD(v), \text{select}_addB(o)) \\
&\text{else } v^* = v
\end{align*}\]

\(v\) and \(v^*\) represent the pre-state and post-state of the operation respectively. After the operation succeeds, the first three parts of \(v^*\) are the same as in \(v\). \textit{select}_addB(o) means for the qualified tuple \(o\), add \((s, o, \text{select}_opr)\) to the \(s\_B\). For readability, the specifications and verifications are all written improved on the original COQ codes.

According to Definition 2, the object for verifying an operation is that on the premise pre-state of the operation is safe, the post-state of the operation is also safe. We will verify each safety property. As an example, Figure 2 shows the lemma of Simple-Security Property for simple select operation.

Figure 2. Lemma of Simple-Security Property for Simple Select Operation

\[
\text{Lemma selectSP : if BasicProperties}(v) = \text{true} \land \text{SimpleSecurity}(v) = \text{true} \land \text{select}(v, s, t, c) = v^* \land \text{SimpleSecurity}(v^*) = \text{true}
\]

\textit{BasicProperties} includes some basic properties relevant to the database, such as a table and its tuples belong to the same database, a table and its tuples are father and sons, etc. These properties seem obvious, but the verification will not succeed without them, which shows the rigor of the proof tool. \textit{select} is as specified in Figure 1.

This lemma shows: if the pre-state satisfies the Simple-Security Property, after the simple select operation, the post-state also satisfies the Simple-Security Property. The verification of this lemma is the verification of the Simple-Security Property for the simple select operation.

### 3.2 Analysis and Verification for Select Operation

Comparing definition 5 with definition 6, we can find that the difference between select operation and simple select operation is: the specification of select operation includes \(s\_tc\), which is a list whose elements are 2-tuple of \(<t, c>\), while there is only...
one \(t\), \(c\) in the specification of simple select operation. Figure 3 shows the specification of select operation rule.

\[
\text{select}(v, s, s_{tc}): \\
case s_{tc} \\
\text{nil} \Rightarrow v \\
ts_{tc} :: ltc \Rightarrow \\
\{
\text{if} \quad \text{objType}(\text{fst } tc) = \text{table} \land \text{HavePriv}(v, (s, (\text{fst } tc), \text{select}_{\text{op}})) = \text{true} \land f_c(v, s) \geq f_c(v, (\text{fst } tc)) \\
\text{then} \quad \forall (o @ \text{includedby}(\text{fst } tc) \land (\text{snd } tc)(o) = \text{true} \land f_c(v, o) \geq f_c(v, o)), \quad v^* = ( (s, \text{ObjectSet } v), (s, \text{DD } v), (s, \text{UD } v), \text{select}_{\text{addB}}(o) ) \cup \text{select}(v, s, ltc) \\
\text{else} \quad v^* = ( (s, \text{ObjectSet } v), (s, \text{DD } v), (s, \text{UD } v), (s, \text{B } v) ) \cup \text{select}(v, s, ltc)
\}
\]

Figure 3. Specification of Select Operation Rule

\("::\) is the symbol for concatenation. \(tc :: ltc\) represents the list by concatenating the list \(ltc\) to the element \(tc\). Other symbols are the same as in the specification of simple select operation rule.

The object for verification of select operation is the same as for simple select operation. The difference is: the post-state of the select operation is more complex to describe. As specified in Figure 3, we need to analyse each element in the list \(s_{tc}\), and add each result for the analysis together to form the final post-state \(v^*\).

As \(s_{tc}\) is a first-in-last-out list, the node taken out prior is the one added into the list later, which represents the deeper select clause in the select statement. So we can simplify the verification of the select operation. First we take out all the nodes \(<t_s, c_s>\) from the list \(s_{tc}\) sequentially, and form the simple select operations \(<v, s, t, c>\) with state \(v\) and subject \(s\). Then we invoke the verification of simple select operation to verify those simple select operations in order, so that we transform the verification of select operation to those of sequential simple select operations.

For example, the transformation, analysis and verification of the select statement:

\[
\text{SELECT ts.sname, tsc.score, (SELECT AVG(score) FROM tsc WHERE cno = '100001') Average FROM ts, tsc WHERE ts.sno=tsc.sno AND tsc.cno = (SELECT cno FROM tc WHERE cname = 'Chinese');}
\]

is like this:

Figure 4. Example for Transformation, Analysis and Verification of Select Operation

As depicted in Figure 4, first we transform the original statement to the specification as in definition 5 like this: \(<v, s, t, c>\), \(<t, c>\),
Then we take out the nodes from the list _s_tc_ in the above specification sequentially, and form the 4 simple select operations: _<v, s, tsc, cno='100001'>_. At last we invoke the verification of simple select operation to verify these simple select operations in order.

### 3.3 Correctness of the Verification of Select Operation

**Theorem 1.** The approach to transform the verification of select operation to those of sequential simple select operations is correct.

**Proof.** The change of the successful operation on the state is to add (s,o,select_op) to the _s_B_ for those qualified tuple o. And which o is qualified is decided by t, c and the security levels. So the proof of Theorem 1 is the same as the correctness proof of the approach to transform _s_tc_ to each single _<t_i, c_i>_.

There are two relationships among the nodes _<t_i, c_i>_ in _s_tc_: 1) the nested clause and the main clause in different hierarchies; 2) different tables in the same hierarchy for join.

1) As mentioned above, the node taken out prior represents the deeper select clause in the select statement. So when we take out the nodes _<t_i, c_i>_ from the list _s_tc_ sequentially, and form the simple select operations _<v, s, t_i, c_i>_ for verification, we are first verifying the select clause in a lower hierarchy and then the main select clause in a higher hierarchy, which is consistent with the execution of the select statement. So this is obviously correct.

2) For the table join, the proof is by contradiction. Let _Sec(O)_ be the proposition that it is secure to add (s,o,select_op) into _s_B_ for all o_i in the tuple set O. Let _<t_1, c_1>, _<t_2, c_2>_ be the nodes in _s_tc_ where _t_1 and _t_2 are the only tables for join. Let _O_ be the tuple set in which all tuples are qualified for _t_1, _c_1 and the security level, and _O_ be the tuple set in which all the tuples are accessed at last in the original select statement. Then our proof is simplified like this: Sec(_O_1) ∧ Sec(_O_2) → Sec(_O_). Because _c_1, _c_2 are the filtering conditions relevant to each single table _t_1, _t_2 respectively, we have _O_ ⊆ _O_1 ∪ _O_2. Suppose that Sec(_O_) is false, then _∃_ o ∈ _O_, so that ¬Sec(_O_). As _O_ ⊆ _O_1 ∪ _O_2, we have _o_ ∈ _O_1 or _o_ ∈ _O_2. Assume that _o_ ∈ _O_1, then ¬Sec(_O_1), which contradicts the condition Sec(_O_1). So the proposition is true. This proof can also be extended to joins for multiple tables.

In conclusion, the approach to transform the verification of select operation to those of sequential simple select operations is correct. □

We wrote twenty thousands lines of COQ codes to analyse and verify a DBMS which includes all SQL operations mentioned in section 2.4. And when we did the verification we meet some problems. Such as how to make sure that the verification for the post-state really needs the precondition in the specification of the operation rules; the FTLS is closer to the implementation, how to simplify its verification due to the details of the implementation, etc. We did the research on these problems and find that it is very useful to specify and verify the FTLS for the design of a secure DBMS.
4 Related Work

Goguen and Meseguer say that building a secure system should be comprised of four stages [12]: 1) Determine the security needs of the system; 2) Express those needs as a formal requirement; 3) Model the system (at least the security relevant components and functions); 4) Verify that this model satisfies the formal requirement.

Maximiliano Cristia specifies and verifies an extension of a secure, compatible UNIX file system [13]. The paper indicates that a secure file system should include subjects and their groups, list of privileges, security levels, etc. The operations should include create, open, close files and folders, etc. The paper also defines the safety properties, and introduces the concept of state machine. The operations in the file system are treated as queries or changes to the state. However, the target object in their work is file system, which differs a lot from the DBMS. The latter is more complex in operations. And the transformation and simplification for verification in our paper is never mentioned before in general systems.

Antonio Coronato et al. propose a method to formally specify and verify the correctness and security of the general application system [14]. They extend the basic formal tools and introduce static and dynamic verification briefly. However, this paper gives the specification so abstractly that doesn’t refer to some practical problems in general application systems.

Hejiao Huang et al. point out that the security policy design is concerned with the composition of components in security systems and interactions among them [15]. The paper uses CPNP to specify and verify security policies in a modular way. They define fundamental policy properties, e.g. completeness, termination, consistency, and confluence in Petri net terminology and get some theoretical results.

In the research of formal specification and verification for DBMS, the classical one is the SeaView model proposed by Teresa F.Lunt et al. [3, 4]. It is a formal security model for multi-level security RDBMS. The object is to design a multi-level secure DBMS with the A1 security level in TCSEC. SeaView security policies include mandatory access control policy, discretionary access control policy, data marker, data consistency, etc. These policies are formalized into two model layers: MAC model which includes a security kernel that supports the A1 security level, and TCB model which is based on the MAC model. The TCB mode includes the integrity constraint, discretionary authorization, and the specification of the operations, etc. However, in SeaView project, only simple SQL operations are considered. For example, the target object of the update operation they specified is only certain tuples whose primary keys are already known, while the where clause that may contain complex expressions is not considered. Actually, the complex clause determines the tuples to be updated and also the objects whose security levels will be compared, so it has much to do with the security. This is normal in current complex update operation.

Maryam Lotfi Shahreza et al. point out that the theory of the relational databases has much in common with the mathematical structures central to the Z notation, and formally specify the applications in DBMS [16]. They first specify the database by UML class diagram in Z. Then they refine the specification until its corresponding database and program is obtained. Finally, they get the SQL operations corresponding to the specifications above. However, they pay more attention on the specification of database applications instead of security.
5 Conclusion

In this paper, we propose a novel approach for the formal specification and verification of the SQL operations in FTLS of a secure DBMS. First, we formally define the SQL operations in FTLS. Then, on top of the specifications, we give the definitions of the simple SQL operations, and propose a method to verify those simple SQL operations. Finally, we transform the verification of the SQL operations in FTLS to the verification of the component simple SQL operations. We also give the proof of the correctness of this approach. From the process of the specification and verification, we can see that our approach makes a comprehensive and clear specification of the SQL operations in FTLS, and also makes an easier verification for proof tool COQ.

References

15. Hejiao Huang, Hélène Kirchner: Formal specification and verification of modular security policy based on colored petri nets. IEEE Transactions on Dependable and Secure Computing. 8(6), 852-865 (2011)