

*Project:* DRA FRONT END FILTER PROJECT

*Title:* Specifications of *hide* and *updateState*

*Ref:* DS/FMU/FEF/005      *Issue: Revision : 4.4*      *Date:* 5 December 2009

*Status:* Draft      *Type:* Specification

*Keywords:*

*Author:*

<i>Name</i>	<i>Location</i>	<i>Signature</i>	<i>Date</i>
G. M. Prout	WIN01		

*Authorisation for Issue:*

<i>Name</i>	<i>Function</i>	<i>Signature</i>	<i>Date</i>
R.B. Jones	HAT Manager		

*Abstract:* The formal specifications of the functions *hide* and *updateState* for the DRA front end filter project RSRE 1C/6130.

*Distribution:* HAT FEF File  
Simon Wiseman

## 0 DOCUMENT CONTROL

### 0.1 Contents List

<b>0</b>	<b>DOCUMENT CONTROL</b>	<b>2</b>
0.1	Contents List . . . . .	2
0.2	Document Cross References . . . . .	2
0.3	Changes History . . . . .	3
0.4	Changes Forecast . . . . .	3
<b>1</b>	<b>GENERAL</b>	<b>4</b>
1.1	Scope . . . . .	4
1.2	Introduction . . . . .	4
<b>2</b>	<b>PRELIMINARIES</b>	<b>4</b>
2.1	Representation and Abstraction Functions for <i>State : Exp</i> . . . . .	4
<b>3</b>	<b>THE FUNCTION <i>hide</i></b>	<b>5</b>
3.1	Cleaning a Table . . . . .	5
3.1.1	Clean Column Constraints . . . . .	5
3.1.2	Filter all Data in a Row . . . . .	6
3.1.3	Replace Data in a Row with Dummy Data . . . . .	6
3.1.4	Clean all Data in a Row . . . . .	7
3.1.5	Clean all Rows in a Table . . . . .	7
3.1.6	The function <i>cleanTable</i> . . . . .	8
3.2	Cleaning a Directory . . . . .	8
3.3	Cleaning a State . . . . .	9
<b>4</b>	<b>THE FUNCTION <i>updateState</i></b>	<b>9</b>
4.1	Auxiliary Functions . . . . .	10
4.2	Insert . . . . .	11
4.3	Delete . . . . .	12
4.4	Update . . . . .	13
4.5	Specification of <i>updateState</i> . . . . .	15
<b>5</b>	<b>CLOSING DOWN</b>	<b>17</b>
<b>6</b>	<b>THE THEORY fef005</b>	<b>18</b>
6.1	Parents . . . . .	18
6.2	Children . . . . .	18
6.3	Constants . . . . .	18
6.4	Definitions . . . . .	20
<b>7</b>	<b>INDEX</b>	<b>26</b>

### 0.2 Document Cross References

- [1] DS/FMU/017. *Secure Database Technical Proposal*. High Assurance Team, ICL Secure Systems, WIN01, 21st January 1992.

[2] DS/FMU/FEF/004. *Specification of SSQL Semantics I*. G.M. Prout, ICL Secure Systems, WIN01.

[3] *Invitation to Tender RSRE 1c/6130*. DRA, December 1991.

### 0.3 Changes History

**Issue** *Revision : 4.4 (5 December 2009)* postfix.

**Issue 4.4** Removed dependency on ICL logo font

### 0.4 Changes Forecast

None.

# 1 GENERAL

## 1.1 Scope

This document gives a formal specification of the functions *hide* and *updateState*. It constitutes deliverable D2 of work package 1a, as given in section 7 of the Secure Database Technical Proposal, [1].

## 1.2 Introduction

In the introduction to [2] we stated that the formalisation of the semantics of SSQL is to be presented in three main parts. Two of these parts, the functions *hide* and *updateState*, are concerned with the security aspects of the SSQL semantics. The functions *hide* and *updateState* are formalised in this document.

# 2 PRELIMINARIES

The following ProofPower instructions set up the new theory *fef005* and set the context for the proof tools.

SML

```
|open_theory "fef004";
|force_delete_theory "fef005" handle _ => ();
|new_theory "fef005";
|push_pc "hol";
```

## 2.1 Representation and Abstraction Functions for *State : Exp*

We define the representation function, *repState*, and abstraction function, *absState*, for the type *State : Exp*.

HOL Constant

```
|
|   repState : State → StateR;
|   absState : StateR → State
|
|-----
|
|   (∀ a • absState (repState a) = a)
| ∧   (∀ r • isState r ⇒ repState (absState r) = r)
| ∧   (∀ a1 a2 • repState a1 = repState a2 ⇔ a1 = a2)
| ∧   (∀ r1 r2 • (isState r1 ∧ isState r2) ⇒
|       (absState r1 = absState r2 ⇔ r1 = r2))
| ∧   (∀ s • isState (repState s))
```

### 3 THE FUNCTION *hide*

The purpose of the *hide* function is to return a version of the database that does not include any information the existence of which the user is not cleared to know, nor does it contain the true value of any field which the user is not cleared to see. This is achieved as follows :

**Section 3.1** A function *cleanTable* is defined which takes a table and a clearance and returns a table which has been cleaned in the following way: A table that the user is not permitted to access is replaced by an ‘empty’ table; otherwise, all the rows and columns about whose existence the user is not cleared to know are removed, and all the data the user is not cleared to see is replaced by dummy data.

**Section 3.2** A function *cleanDirectory* is defined which takes a directory and a clearance and returns a directory which has been cleaned in the following way: if the user is not permitted to know of the existence of the tables in the directory then a directory with no tables in it is returned; otherwise the tables in the directory are cleaned by *cleanTable*.

**Section 3.3** Finally, the function *hide* is defined which takes a state and a clearance and returns a state which has been cleaned in the following way: all directories about whose existence the user is not cleared to know are removed and the rest are cleaned by *cleanDirectory*.

#### 3.1 Cleaning a Table

In the case where the user is permitted to know of the existence of a table, but is not permitted to access it, i.e. his clearance does not dominate the table class, an ‘empty’ table is left in the directory in place of that table.

All other tables are cleaned in the following way

- Remove from the column constraints those groups of columns about whose existence the user is not cleared to know.
- Remove any columns about whose existence the user is not cleared to know.
- Remove any rows about whose existence the user is not cleared to know.
- Remove data from columns of remaining rows about whose existence the user is not cleared to know.
- Replace data that the user is not cleared to see by dummy values.

##### 3.1.1 Clean Column Constraints

A function that takes a clearance and a table and returns a pair: the column constraints for columns whose existence is dominated by the clearance, and the corresponding set of columns.

HOL Constant

---

**cleanColCons** : *Class* → *TableSpec* → ((*Num* ↔ *ColCon*) × *ColSpec* ℙ)

---

 $\forall$  *clear* : *Class*; *ts* : *TableSpec*

- *cleanColCons clear ts*

=

 $let \quad ccs = TS\_cons \ ts \triangleright \{cc \mid clear \text{ dominates } (CC\_exist \ cc)\}$ 

in

 $(ccs, \{col : ColSpec \mid (CS\_consGroup \ col) \in Dom \ ccs\})$ 

### 3.1.2 Filter all Data in a Row

Filter out the data in columns about whose existence the user is cleared to know.

HOL Constant

---

**filterRow** : *ColSpec* ℙ → (*Num* ↔ *Data*) → (*Num* ↔ *Data*)

---

 $\forall$  *cols* : *ColSpec* ℙ; *ds* : *Num* ↔ *Data*

- *filterRow cols ds*

=

 $\{n \mid \exists c \bullet c \in cols \wedge CS\_posn \ c = n\} \triangleleft ds$ 

### 3.1.3 Replace Data in a Row with Dummy Data

Data consists of a classification and an item of data. An item of data that a user is not cleared to see is replaced by a dummy string *Hidden* (its classification remains).

HOL Constant

---

**Hidden** : *String*


---

*T*

HOL Constant

---

**dummyVal** : *Val*


---

*dummyVal* = *StringVal Hidden*

A function to replace the value of a piece of data that the user is not cleared to see by a dummy value.

HOL Constant

---

**replaceData** : *Class* → *Data* → *Data*


---

 $\forall$  *clear* : *Class*; *d* : *Data*

- *replaceData clear d*

=

if  $\neg(\text{clear dominates } (\text{Dat\_class } d))$ then *MkData* (*Dat\\_class d*)( *ValuedItemItem* (*MkValuedItem worthless dummyVal*))else *d*

### 3.1.4 Clean all Data in a Row

Filter out the data in columns about whose existence the user is cleared to know. Replace data in fields that the user is not cleared to see by dummy values.

HOL Constant

---

**cleanRow** : *Class* → *ColSpec*  $\mathbb{P}$  → *Row* → *Row*


---

 $\forall$  *clear* : *Class*; *cols* : *ColSpec*  $\mathbb{P}$ ; *r*: *Row*

- *cleanRow clear cols r*

=

*MkRow* (*R\\_exist r*)((*filterRow cols* (*R\\_data r*)) % *Graph*(*replaceData clear*))

### 3.1.5 Clean all Rows in a Table

Delete all rows about whose existence the user is not cleared to know, and then clean remaining rows.

HOL Constant

---

**cleanRows** : *Class* → *ColSpec*  $\mathbb{P}$  → *Row LIST* → *Row LIST*


---

 $\forall$  *clear* : *Class*; *cols* : *ColSpec*  $\mathbb{P}$ ; *rs*: *Row LIST*

- *cleanRows clear cols rs*

=

*Map* (*cleanRow clear cols*) (*rs*  $\uparrow$  {*r*:*Row*|*clear dominates* (*R\\_exist r*)})

### 3.1.6 The function *cleanTable*

The function *cleanTable* either returns an ‘empty’ table or a table where the rows and columns that the user is not cleared to see have been removed and fields that the user is not cleared to see have been overwritten by dummy fields.

HOL Constant

```

cleanTable : Class → TableSpec → TableSpec

```

---

```

∀ clear : Class; ts:TableSpec
•   cleanTable clear ts
    =
    if   ¬(clear dominates (TS_class ts))
    then MkTableSpec
          (TS_class ts)
          (TS_class ts)
          {}{}[]
    else let (ccs,cols) = cleanColCons clear ts
          in
          MkTableSpec
            (TS_class ts)
            (TS_maxRow ts)
            cols
            ccs
            (cleanRows clear cols (TS_rows ts))

```

## 3.2 Cleaning a Directory

Any tables in a directory about whose existence the user is not cleared to know can be removed from that directory. These are all the tables in a directory whose *class* component is not dominated by the user’s clearance (i.e. the user is cleared to know the existence of the directory, but is not cleared to see the structure of the directory). To clean a directory, the tables in that directory will either be hidden or modified by *cleanTable*.

HOL Constant

---

**cleanDirectory** :  $Class \rightarrow Directory \rightarrow Directory$ 


---

 $\forall clear : Class; dir : Directory$ 

- $cleanDirectory\ clear\ dir$

=

 $MkDirectory$  $(if\ (clear\ dominates\ (Dir\_class\ dir))$  $then\ (Dir\_tables\ dir\ ;\ Graph(cleanTable\ clear))$  $else\ \{\})$  $(Dir\_exist\ dir)$  $(Dir\_class\ dir)$ 

### 3.3 Cleaning a State

Finally, any directories about whose existence the user is not cleared to know can be removed from the state. All remaining directories will be cleaned by *cleanDirectory*. First, we define *hideR* on the representation state.

HOL Constant

---

**hideR** :  $Class \times StateR \rightarrow StateR$ 


---

 $\forall clear : Class; s : StateR$ 

- $hideR(clear,s)$

=

 $(s \triangleright \{dir \mid clear\ dominates\ (Dir\_exist\ dir)\}) ; Graph(cleanDirectory\ clear)$ 

Now the function *hide* on  $State : Exp$ .

HOL Constant

---

**hide** :  $Class \times State \rightarrow State$ 


---

 $\forall clear : Class; s : State$ 

- $hide(clear,s) = absState(hideR(clear, repState\ s))$

## 4 THE FUNCTION *updateState*

In [2] the function *processQuery* is defined. This function takes a query, a user's classification and a state and returns the result of the query, i.e., the output to be seen by the user, an update to be applied to the data base and some error messages. The effect of a query depends on the type of the query, which may be an *Insert*, a *Delete* an *UpDate* or a *Select*.

We have not yet determined whether it is necessary to have as a state invariant that column existence classifications are in ascending order.

## 4.1 Auxiliary Functions

When a user makes an update request on a row, the position of the row in a table will be its position after rows about whose existence the user is not cleared to know have been hidden. We require a function that reveals the true row position given the hidden row position.

HOL Constant

$$\mathbf{revealRow} : \mathit{Class} \rightarrow \mathit{TableSpec} \rightarrow \mathit{Num} \leftrightarrow \mathit{Num}$$


---


$$\forall \mathit{clear} : \mathit{Class}; \mathit{ts} : \mathit{TableSpec}$$

- $\mathit{revealRow} \ \mathit{clear} \ \mathit{ts}$   
=

$$\mathit{let} \quad \mathit{visiblerows} = \{r : \mathit{Row} \mid \mathit{clear} \ \mathit{dominates} \ (\mathit{R\_exist} \ r)\}$$

$$\mathit{in}$$

$$\mathit{Squash}(\mathit{Id}(\mathit{Dom}((\mathit{ListRel}(\mathit{TS\_rows} \ \mathit{ts})) \triangleright \mathit{visiblerows})))$$

We provide the auxiliary functions *replaceRows* and *changeSpec* from Annex 2 *SSQL Abstract Machine* of the ITT [3].

HOL Constant

$$\mathbf{replaceRows} : \mathit{TableSpec} \rightarrow \mathit{Row} \ \mathit{LIST} \rightarrow \mathit{TableSpec}$$


---


$$\forall \mathit{ts} : \mathit{TableSpec}; \mathit{rs} : \mathit{Row} \ \mathit{LIST}$$

- $\mathit{replaceRows} \ \mathit{ts} \ \mathit{rs}$   
=

$$\mathit{MkTableSpec}(\mathit{TS\_class} \ \mathit{ts})(\mathit{TS\_maxRow} \ \mathit{ts})(\mathit{TS\_colspecs} \ \mathit{ts})(\mathit{TS\_cons} \ \mathit{ts}) \ \mathit{rs}$$

The operator @ is the partial function application operator.

HOL Constant

$$\mathbf{changeSpec} : \mathit{Tab} \rightarrow \mathit{TableSpec} \rightarrow \mathit{StateR} \rightarrow \mathit{StateR}$$


---


$$\forall \mathit{i} : \mathit{Tab}; \mathit{ts} : \mathit{TableSpec}; \mathit{s} : \mathit{StateR}$$

- $\mathit{changeSpec} \ \mathit{i} \ \mathit{ts} \ \mathit{s}$   
=

$$\mathit{let} \quad \mathit{dir} = \mathit{s} \ @ \ \mathit{Front} \ \mathit{i}$$

$$\mathit{in}$$

$$\mathit{let} \quad \mathit{dir}' = \mathit{MkDirectory}(\mathit{Dir\_tables} \ \mathit{dir} \oplus \{(Last \ \mathit{i}, \mathit{ts})\})$$

$$\quad \quad \quad (\mathit{Dir\_exist} \ \mathit{dir})(\mathit{Dir\_class} \ \mathit{dir})$$

$$\mathit{in}$$

$$\mathit{s} \oplus \{(\mathit{Front} \ \mathit{i}, \mathit{dir}')\}$$

A function that returns the visible columns of a table.

HOL Constant

$$\mathbf{visibleCols} : Class \rightarrow TableSpec \rightarrow ColSpec \mathbb{P}$$

$$\forall clear : Class; ts: TableSpec$$

- $visibleCols\ clear\ ts$

$$=$$

$$Snd(cleanColCons\ clear\ ts)$$

A function *tabExists* is given that determines whether or not a table name exists at a particular clearance in a particular state.

HOL Constant

$$\mathbf{tabExists} : Class \rightarrow Tab \rightarrow StateR \rightarrow Bool$$

$$\forall c : Class; i : Tab; s:StateR$$

- $tabExists\ c\ i\ s$

$$\Leftrightarrow$$

$$Front\ i \in Dom\ s$$

$$\wedge$$

$$c\ dominates\ Dir\_exist\ (s\ @\ Front\ i)$$

$$\wedge$$

$$c\ dominates\ Dir\_class\ (s\ @\ Front\ i)$$

$$\wedge$$

$$Last\ i \in Dom\ (Dir\_tables\ (s\ @\ Front\ i))$$

The function *getTable* returns a table given its full table name. This will only be applied in the case where the table name exists.

HOL Constant

$$\mathbf{getTable} : Tab \rightarrow StateR \rightarrow TableSpec$$

$$\forall i : Tab; s:StateR$$

- 

$$getTable\ i\ s = (Dir\_tables(s\ @\ (Front\ i)))\ @\ (Last\ i)$$

## 4.2 Insert

The function *colDefaults* takes a partial function from column number to data and a table and adds the appropriate column number - default value pairs which correspond to the columns that were hidden to the user. The default values are obtained from the table information.

HOL Constant

$$\mathbf{colDefaults} : \text{Class} \rightarrow \text{TableSpec} \rightarrow (\text{Num} \leftrightarrow \text{Data}) \rightarrow (\text{Num} \leftrightarrow \text{Data})$$

$$\forall \text{clear} : \text{Class}; \text{ts} : \text{TableSpec}; \text{ds} : \text{Num} \leftrightarrow \text{Data}$$

- $\text{colDefaults clear ts ds}$

$$=$$

$$\text{ds} \oplus \{(n,d) \mid \exists c \bullet \neg c \in \text{visibleCols clear ts} \\ \wedge n = \text{CS\_posn } c \wedge d = \text{CS\_default } c\}$$

An *ambiguousColumn* error is returned if the data in a row to be inserted is not a partial function.

HOL Constant

$$\mathbf{insertQuery} : (\text{Class} \times \text{Insert} \times \text{StateR} \times \text{TableSpec}) \rightarrow \text{StateR} \times \text{Errors}$$

$$\forall \text{clear} : \text{Class}; i : \text{Tab}; \text{ds} : (\text{Num} \leftrightarrow \text{Data}) \text{ LIST}; s : \text{StateR}; \text{ts} : \text{TableSpec}$$

- $\text{insertQuery (clear, (i, ds), s, ts)}$

$$=$$

$$\text{let } rl = \text{Map } ((\text{MkRow clear}) \circ (\text{colDefaults clear ts})) \text{ ds}$$

$$\text{in}$$

$$\text{if } \neg \text{Elms } rl \subseteq \text{RowS}$$

$$\text{then } (s, [\text{ambiguousColumn}])$$

$$\text{else}$$

$$((\text{changeSpec } i (\text{replaceRows ts } ((\text{TS\_rows } ts) \hat{\ } rl)) s), [])$$

### 4.3 Delete

Delete all rows specified by *DeleteEffect*. The row numbers supplied by *DeleteEffect* correspond to the users view of the hidden state of the data base. In order to delete the correct rows from the true state of the data base, it is necessary to take into account those rows about whose existence the user is not cleared to know. If the user were not permitted access to the table then the structure of the table would have been hidden from him and therefore the request would have failed.

HOL Constant

---

**deleteQuery** :  $(Class \times Delete \times StateR \times TableSpec) \rightarrow StateR$ 


---

 $\forall clear:Class; i:Tab; ns : Num \mathbb{P}; e:Errors; s:StateR; ts:TableSpec$ 

- $deleteQuery(clear,(i,ns),s,ts)$

=

 $let \quad ns' = (revealRow \ clear \ ts) \ Image \ ns$ 

in

 $let \ ns'' = ns' \cap \{i|R\_exist(Nth(TS\_rows \ ts)i) = clear\}$ 
 $in \quad let \ rs = Extract \ (1 \ .. \ Length \ (TS\_rows \ ts) \ \ ns'')(TS\_rows \ ts)$ 

in

 $(changeSpec \ i \ (replaceRows \ ts \ rs) \ s)$ 

#### 4.4 Update

Perform the updates specified by *UpdateEffect*, provided that they do not violate the security policy. A user may change the value in a field (classification stays the same) provided the field's classification dominates the user's clearance. A user may downgrade the classification of a field providing that the value in the field is overwritten.

Firstly, consider an update request on a single field of data. This will either yield a new piece of data or an error message.

HOL Constant

---

**updateField** :  $Class \rightarrow Class \rightarrow (Update \times Data) \rightarrow (Data + Error)$ 


---

 $\forall clear \ table\_class:Class; table\_d:Data; u:Update$ 

- $updateField \ clear \ table\_class \ (u,table\_d)$

=

 $if \quad clear = table\_class$ 
 $then \quad if \quad isItem \ u$ 
 $then \quad giveVal(MkData \ (Dat\_class \ table\_d)(destItem \ u))$ 
 $else \ if \ isClass \ u$ 
 $then \quad if \ (destClass \ u) \ dominates \ (Dat\_class \ table\_d)$ 
 $then \quad giveVal(MkData \ (destClass \ u) \ (Dat\_item \ table\_d))$ 
 $else \quad giveError \ downgrade$ 
 $else \quad giveVal(destData \ u) \quad (* \ must \ be \ Data \ *)$ 
 $else \ if \ isItem \ u$ 
 $then \quad if \quad (Dat\_class \ table\_d) \ dominates \ clear$ 
 $then \quad giveVal(MkData \ (Dat\_class \ table\_d) \ (destItem \ u))$ 
 $else \quad giveError \ underClassified$ 
 $else \quad giveError \ classChange$

Now the effect of an update on a row.

Updates for a row must be functional otherwise an *ambiguousUpdate* error is returned. A new row is only returned if all updates are secure; otherwise a sequence of errors is returned.

HOL Constant

$$\mathbf{updateRow} : \text{Class} \rightarrow \text{Class} \rightarrow ((\text{Num} \leftrightarrow \text{Update}) \times \text{Row}) \rightarrow (\text{Row} + \text{Errors})$$

$$\forall \text{clear table\_class}:\text{Class}; \text{us}:\text{Num} \leftrightarrow \text{Update}; \text{r}:\text{Row}$$

- $\text{updateRow clear table\_class (us,r)}$

=

if  $\neg \text{us} \in \text{Functional}$

then  $\text{giveError [ambiguousUpdate]}$

else

let  $\text{us}' = \text{RelCombine us (R\_data r)} \% \text{Graph(updateField clear table\_class)}$

in

let  $\text{es} = (\text{us}' \triangleright \{x | \text{isError } x\}) \% \text{Graph destError}$

in

if  $\text{es} = \{\}$

then  $\text{giveVal(MkRow (R\_exist r)((R\_data r) \oplus (\text{us}' \% \text{Graph destVal})))$

else  $\text{giveError (RelList(Squash es))}$

Finally, the *updateQuery* function. Updates must be functional otherwise an *ambiguousUpdate* error is returned. A check is made to prevent access to hidden columns.

HOL Constant

$$\mathbf{updateQuery} : (Class \times UpDate \times StateR \times TableSpec) \rightarrow StateR \times Errors$$

$$\begin{aligned} & \forall clear:Class; i:Tab; us : Num \leftrightarrow (Num \leftrightarrow Update); s:StateR; ts : TableSpec \\ & \bullet \quad updateQuery (clear,(i,us),s,ts) \\ & \quad = \\ & \quad let \ colnums = \{n \mid \exists c \bullet c \in visibleCols \ clear \ ts \wedge CS\_posn \ c = n\} \\ & \quad in \\ & \quad if \quad \neg us \in Functional \\ & \quad then \ (s,[ambiguousUpdate]) \\ & \quad else if \ \neg((Dom (\bigcup (Ran \ us))) \subseteq colnums) \\ & \quad then \ (s,[noSuchColumn]) \\ & \quad else \ let \ us' = (revealRow \ clear \ ts) \sim \ ; \ us \\ & \quad \quad in \\ & \quad \quad let \ pr = RelCombine \ us' (ListRel(TS\_rows \ ts)) \\ & \quad \quad \quad \ ; \ Graph(updateRow \ clear (TS\_class \ ts)) \\ & \quad \quad in \\ & \quad \quad let \ es = (pr \triangleright \{x \mid isError \ x\}) \ ; \ Graph \ destError \\ & \quad \quad in \ if \ es = \{\} \\ & \quad \quad \quad then \ let \ rs = RelList(ListRel(TS\_rows \ ts)) \\ & \quad \quad \quad \quad \oplus (pr \ ; \ Graph \ destVal)) \\ & \quad \quad \quad \quad in \ (changeSpec \ i(replaceRows \ ts \ rs) \ s,[]) \\ & \quad \quad else \ (s,Flat(RelList(Squash \ es))) \end{aligned}$$

#### 4.5 Specification of *updateState*

The function *updateState* operates on the output from the function *processQuery*, defined in [2]. This output is a pair consisting of the effect of the update query, which depends on whether the query was a select, insert, delete or update, and a sequence of errors messages. For a select query, data is returned to the user, together with any error messages. The state of the database remains unchanged. For an insert, delete or update query, if a table name is returned by *processQuery* that does not exist, then the state of the database remains unchanged and an appropriate error message is returned. If errors were generated by *processQuery* then the state of the database remains unchanged and the errors are returned. If a valid table name and no errors are returned by *processQuery*, then if the clearance of the user does not dominate the classification of the table, again the state of the database remains unchanged and an appropriate error message is returned. Otherwise, the appropriate update to the state of the database, given by *insertQuery*, *deleteQuery* and *updateQuery*, is carried out after the security checks have been made. In the case of an insert or update query, if the security checks identified that the output of *processQuery* would result in an insecure update to the state of the database, then the state of the database remains unchanged and error messages generated by the security checks are returned to the user.

First, we define *updateStateR* on the representation state.

HOL Constant

$$\mathbf{updateStateR} : \text{Class} \times (\text{Effect} \times \text{Errors}) \times \text{StateR} \rightarrow$$

$$\text{StateR} \times (\text{Class} \times (\text{Data LIST LIST} \times \text{Errors}))$$

$$\forall \text{clear}:\text{Class}; \text{ef} : \text{Effect}; \text{es} : \text{Errors}; \text{s} : \text{StateR}$$

- $\text{updateStateR}(\text{clear},(\text{ef},\text{es}),\text{s})$   
 $=$   
*if*  $\text{isSelect ef}$   
*then*  $(\text{s},(\text{clear},(\text{destSelect ef},\text{es})))$   
*else if*  $\neg(\text{es} = [])$  (\* delete,update and insert only valid if no errors \*)  
*then*  $(\text{s},(\text{clear},([],\text{es})))$   
*else let*  $i = \text{tabFromEffect ef}$   
*in*  
*if*  $\text{tabExists clear } i \text{ s}$   
*then let*  $\text{ts} = \text{getTable } i \text{ s}$   
*in*  
*if*  $\neg(\text{clear dominates TS\_class ts})$   
*then*  $(\text{s},(\text{clear},([],[\text{accessDenied}])))$   
*else*  
*let*  $(\text{s}',\text{es}')$   
 $=$   
*if*  $\text{isInsert ef}$   
*then*  $\text{insertQuery}(\text{clear},\text{destInsert ef},\text{s},\text{ts})$   
*else if*  $\text{isDelete ef}$   
*then*  $(\text{deleteQuery}(\text{clear},\text{destDelete ef},\text{s},\text{ts}),[])$   
*else*  $\text{updateQuery}(\text{clear},\text{destUpdate ef},\text{s},\text{ts})$   
*in*  
 $(\text{s}',(\text{clear},([],\text{es}')))$   
*else*  $(\text{s},(\text{clear},([],[\text{noSuchTable}])))$  (\* Invalid table name \*)

Now the function *updateState* on *State* : *Exp*.

HOL Constant

$$\mathbf{updateState} : \text{Class} \times (\text{Effect} \times \text{Errors}) \times \text{State} \rightarrow$$

$$\text{State} \times (\text{Class} \times (\text{Data LIST LIST} \times \text{Errors}))$$

$$\forall \text{clear}:\text{Class}; \text{ef} : \text{Effect}; \text{es} : \text{Errors}; \text{s} : \text{State}$$

- $\text{updateState}(\text{clear},(\text{ef},\text{es}),\text{s})$   
 $=$   
*let*  $(\text{s}_r,\text{out}) = \text{updateStateR}(\text{clear},(\text{ef},\text{es}),\text{repState } \text{s})$   
*in*  $(\text{absState } \text{s}_r,\text{out})$

## 5 CLOSING DOWN

The following ProofPower instruction restores the previous proof context.

SML

```
|pop-pc();
```

## 6 THE THEORY fef005

### 6.1 Parents

*fef004*

### 6.2 Children

*fef014*

### 6.3 Constants

<b>absState</b>	$(CHAR\ LIST\ LIST \times Directory)\ SET \rightarrow State$
<b>repState</b>	$State \rightarrow (CHAR\ LIST\ LIST \times Directory)\ SET$
<b>cleanColCons</b>	$Class \rightarrow TableSpec \rightarrow (\mathbb{N} \times ColCon)\ SET \times ColSpec\ SET$
<b>filterRow</b>	$ColSpec\ SET \rightarrow (\mathbb{N} \times Data)\ SET \rightarrow (\mathbb{N} \times Data)\ SET$
<b>Hidden</b>	$CHAR\ LIST$
<b>dummyVal</b>	$ONE$ $+\ BOOL$ $+ CHAR\ LIST$ $+ Int$ $+ Float$ $+ Time$ $+ Interval$ $+ Code$ $+ Class$ $+ ONE$
<b>replaceData</b>	$Class \rightarrow Data \rightarrow Data$
<b>cleanRow</b>	$Class \rightarrow ColSpec\ SET \rightarrow Row \rightarrow Row$
<b>cleanRows</b>	$Class \rightarrow ColSpec\ SET \rightarrow Row\ LIST \rightarrow Row\ LIST$
<b>cleanTable</b>	$Class \rightarrow TableSpec \rightarrow TableSpec$
<b>cleanDirectory</b>	$Class \rightarrow Directory \rightarrow Directory$
<b>hideR</b>	$Class \times (CHAR\ LIST\ LIST \times Directory)\ SET$ $\rightarrow (CHAR\ LIST\ LIST \times Directory)\ SET$
<b>hide</b>	$Class \times State \rightarrow State$
<b>revealRow</b>	$Class \rightarrow TableSpec \rightarrow (\mathbb{N} \times \mathbb{N})\ SET$
<b>replaceRows</b>	$TableSpec \rightarrow Row\ LIST \rightarrow TableSpec$
<b>changeSpec</b>	$CHAR\ LIST\ LIST$ $\rightarrow TableSpec$ $\rightarrow (CHAR\ LIST\ LIST \times Directory)\ SET$ $\rightarrow (CHAR\ LIST\ LIST \times Directory)\ SET$
<b>visibleCols</b>	$Class \rightarrow TableSpec \rightarrow ColSpec\ SET$
<b>tabExists</b>	$Class$ $\rightarrow CHAR\ LIST\ LIST$ $\rightarrow (CHAR\ LIST\ LIST \times Directory)\ SET$ $\rightarrow BOOL$

---

<b>getTable</b>	<i>CHAR LIST LIST</i> → ( <i>CHAR LIST LIST</i> × <i>Directory</i> ) <i>SET</i> → <i>TableSpec</i>
<b>colDefaults</b>	<i>Class</i> → <i>TableSpec</i> → ( $\mathbb{N}$ × <i>Data</i> ) <i>SET</i> → ( $\mathbb{N}$ × <i>Data</i> ) <i>SET</i>
<b>insertQuery</b>	<i>Class</i> × ( <i>CHAR LIST LIST</i> × ( $\mathbb{N}$ × <i>Data</i> ) <i>SET LIST</i> ) × ( <i>CHAR LIST LIST</i> × <i>Directory</i> ) <i>SET</i> × <i>TableSpec</i> → ( <i>CHAR LIST LIST</i> × <i>Directory</i> ) <i>SET</i> × $\mathbb{N}$ <i>LIST</i>
<b>deleteQuery</b>	<i>Class</i> × ( <i>CHAR LIST LIST</i> × $\mathbb{N}$ <i>SET</i> ) × ( <i>CHAR LIST LIST</i> × <i>Directory</i> ) <i>SET</i> × <i>TableSpec</i> → ( <i>CHAR LIST LIST</i> × <i>Directory</i> ) <i>SET</i>
<b>updateField</b>	<i>Class</i> → <i>Class</i> → ( <i>ValuedItem</i> + <i>ONE</i> ) + <i>Class</i> + <i>Data</i> × <i>Data</i> → <i>Data</i> + $\mathbb{N}$
<b>updateRow</b>	<i>Class</i> → <i>Class</i> → ( $\mathbb{N}$ × ( <i>ValuedItem</i> + <i>ONE</i> ) + <i>Class</i> + <i>Data</i> ) <i>SET</i> × <i>Row</i> → <i>Row</i> + $\mathbb{N}$ <i>LIST</i>
<b>updateQuery</b>	<i>Class</i> × ( <i>CHAR LIST LIST</i> × ( $\mathbb{N}$ × ( $\mathbb{N}$ × ( <i>ValuedItem</i> + <i>ONE</i> ) + <i>Class</i> + <i>Data</i> ) <i>SET</i> ) <i>SET</i> ) × ( <i>CHAR LIST LIST</i> × <i>Directory</i> ) <i>SET</i> × <i>TableSpec</i> → ( <i>CHAR LIST LIST</i> × <i>Directory</i> ) <i>SET</i> × $\mathbb{N}$ <i>LIST</i>
<b>updateStateR</b>	<i>Class</i> × (( <i>CHAR LIST LIST</i> × ( $\mathbb{N}$ × <i>Data</i> ) <i>SET LIST</i> ) + ( <i>CHAR LIST LIST</i> × $\mathbb{N}$ <i>SET</i> ) + ( <i>CHAR LIST LIST</i> × ( $\mathbb{N}$ × ( $\mathbb{N}$ × ( <i>ValuedItem</i> + <i>ONE</i> ) + <i>Class</i> + <i>Data</i> ) <i>SET</i> ) <i>SET</i> ) + <i>Data LIST LIST</i> × $\mathbb{N}$ <i>LIST</i> ) × ( <i>CHAR LIST LIST</i> × <i>Directory</i> ) <i>SET</i> → ( <i>CHAR LIST LIST</i> × <i>Directory</i> ) <i>SET</i> × <i>Class</i> × <i>Data LIST LIST</i> × $\mathbb{N}$ <i>LIST</i>
<b>updateState</b>	<i>Class</i>

---

$$\begin{aligned}
& \times ((CHAR\ LIST\ LIST \times (\mathbb{N} \times Data)\ SET\ LIST) \\
& + (CHAR\ LIST\ LIST \times \mathbb{N}\ SET) \\
& + (CHAR\ LIST\ LIST \\
& \quad \times (\mathbb{N} \\
& \quad \quad \times (\mathbb{N} \times (ValuedItem + ONE) + Class + Data) \\
& \quad \quad \quad SET)\ SET) \\
& + Data\ LIST\ LIST \\
& \times \mathbb{N}\ LIST) \\
& \times State \\
& \rightarrow State \times Class \times Data\ LIST\ LIST \times \mathbb{N}\ LIST
\end{aligned}$$

## 6.4 Definitions

**repState**

**absState**

$$\begin{aligned}
& \vdash ConstSpec \\
& (\lambda (repState', absState') \\
& \quad \bullet (\forall a \bullet absState' (repState' a) = a) \\
& \quad \wedge (\forall r \\
& \quad \quad \bullet isState\ r \Rightarrow repState' (absState' r) = r) \\
& \quad \wedge (\forall a_1\ a_2 \\
& \quad \quad \bullet repState' a_1 = repState' a_2 \Leftrightarrow a_1 = a_2) \\
& \quad \wedge (\forall r_1\ r_2 \\
& \quad \quad \bullet isState\ r_1 \wedge isState\ r_2 \\
& \quad \quad \quad \Rightarrow (absState' r_1 = absState' r_2 \\
& \quad \quad \quad \Leftrightarrow r_1 = r_2)) \\
& \quad \wedge (\forall s \bullet isState (repState' s))) \\
& (repState', absState')
\end{aligned}$$

**cleanColCons**

$$\begin{aligned}
& \vdash \forall clear\ ts \\
& \quad \bullet cleanColCons\ clear\ ts \\
& \quad = (let\ ccs \\
& \quad \quad = TS\_cons\ ts \\
& \quad \quad \triangleright \{cc | clear\ dominates\ CC\_exist\ cc\} \\
& \quad \quad in (ccs, \{col | CS\_consGroup\ col \in Dom\ ccs\}))
\end{aligned}$$

**filterRow**

$$\begin{aligned}
& \vdash \forall cols\ ds \\
& \quad \bullet filterRow\ cols\ ds \\
& \quad = \{n | \exists c \bullet c \in cols \wedge CS\_posn\ c = n\} \triangleleft ds
\end{aligned}$$

**Hidden**

$\vdash T$

**dummyVal**

$\vdash dummyVal = StringVal\ Hidden$

**replaceData**

$$\begin{aligned}
& \vdash \forall clear\ d \\
& \quad \bullet replaceData\ clear\ d \\
& \quad = (if\ \neg\ clear\ dominates\ Dat\_class\ d \\
& \quad \quad then \\
& \quad \quad \quad MkData \\
& \quad \quad \quad (Dat\_class\ d) \\
& \quad \quad \quad (ValuedItemItem \\
& \quad \quad \quad (MkValuedItem\ worthless\ dummyVal)) \\
& \quad \quad else\ d)
\end{aligned}$$

**cleanRow**

$\vdash \forall clear\ cols\ r$

---

- *cleanRow clear cols r*  
 $= \text{MkRow}$   
 $(R\_exist\ r)$   
 $(\text{filterRow}\ cols\ (R\_data\ r))$   
 $\% \text{Graph}\ (\text{replaceData}\ clear)$

**cleanRows**  $\vdash \forall\ clear\ cols\ rs$ 

- *cleanRows clear cols rs*  
 $= \text{Map}$   
 $(\text{cleanRow}\ clear\ cols)$   
 $(rs\ \upharpoonright\ \{r\mid \text{clear}\ \text{dominates}\ R\_exist\ r\})$

**cleanTable**  $\vdash \forall\ clear\ ts$ 

- *cleanTable clear ts*  
 $= (\text{if}\ \neg\ \text{clear}\ \text{dominates}\ TS\_class\ ts$   
 $\text{then}$   
 $\text{MkTableSpec}$   
 $(TS\_class\ ts)$   
 $(TS\_class\ ts)$   
 $\{\}$   
 $\{\}$   
 $\square$   
 $\text{else}$   
 $(\text{let}\ (ccs,\ cols) = \text{cleanColCons}\ clear\ ts$   
 $\text{in}\ \text{MkTableSpec}$   
 $(TS\_class\ ts)$   
 $(TS\_maxRow\ ts)$   
 $cols$   
 $ccs$   
 $(\text{cleanRows}\ clear\ cols\ (TS\_rows\ ts))))$

**cleanDirectory**  $\vdash \forall\ clear\ dir$ 

- *cleanDirectory clear dir*  
 $= \text{MkDirectory}$   
 $(\text{if}\ clear\ \text{dominates}\ Dir\_class\ dir$   
 $\text{then}$   
 $\text{Dir\_tables}\ dir\ \% \text{Graph}\ (\text{cleanTable}\ clear)$   
 $\text{else}\ \{\})$   
 $(Dir\_exist\ dir)$   
 $(Dir\_class\ dir)$

**hideR**  $\vdash \forall\ clear\ s$ 

- *hideR (clear, s)*  
 $= (s\ \triangleright\ \{dir\mid \text{clear}\ \text{dominates}\ Dir\_exist\ dir\})$   
 $\% \text{Graph}\ (\text{cleanDirectory}\ clear)$

**hide**  $\vdash \forall\ clear\ s$ 

- *hide (clear, s)*  
 $= \text{absState}\ (\text{hideR}\ (clear,\ \text{repState}\ s))$

**revealRow**  $\vdash \forall\ clear\ ts$ 

- *revealRow clear ts*  
 $= (\text{let}\ \text{visiblerows}$

---

---

	$= \{r \mid \text{clear dominates } R\_exist \ r\}$ <i>in Squash</i> <i>(Id</i> <i>(Dom</i> <i>(ListRel (TS_rows ts)</i> <i>▷ visiblerows))))</i>
<b>replaceRows</b>	$\vdash \forall ts \ rs$ <ul style="list-style-type: none"> <li>• <i>replaceRows ts rs</i></li> </ul> $= \text{MkTableSpec}$ $(\text{TS\_class } ts)$ $(\text{TS\_maxRow } ts)$ $(\text{TS\_colspecs } ts)$ $(\text{TS\_cons } ts)$ $rs$
<b>changeSpec</b>	$\vdash \forall i \ ts \ s$ <ul style="list-style-type: none"> <li>• <i>changeSpec i ts s</i></li> </ul> $= (\text{let } dir = s \ @ \ \text{Front } i$ $\text{in let } dir'$ $= \text{MkDirectory}$ $(\text{Dir\_tables } dir \oplus \{(Last \ i, \ ts)\})$ $(\text{Dir\_exist } dir)$ $(\text{Dir\_class } dir)$ $\text{in } s \oplus \{(\text{Front } i, \ dir')\})$
<b>visibleCols</b>	$\vdash \forall clear \ ts$ <ul style="list-style-type: none"> <li>• <i>visibleCols clear ts = Snd (cleanColCons clear ts)</i></li> </ul>
<b>tabExists</b>	$\vdash \forall c \ i \ s$ <ul style="list-style-type: none"> <li>• <i>tabExists c i s</i></li> </ul> $\Leftrightarrow \text{Front } i \in \text{Dom } s$ $\wedge c \text{ dominates } \text{Dir\_exist } (s \ @ \ \text{Front } i)$ $\wedge c \text{ dominates } \text{Dir\_class } (s \ @ \ \text{Front } i)$ $\wedge Last \ i \in \text{Dom } (\text{Dir\_tables } (s \ @ \ \text{Front } i))$
<b>getTable</b>	$\vdash \forall i \ s$ <ul style="list-style-type: none"> <li>• <i>getTable i s = Dir_tables (s @ Front i) @ Last i</i></li> </ul>
<b>colDefaults</b>	$\vdash \forall clear \ ts \ ds$ <ul style="list-style-type: none"> <li>• <i>colDefaults clear ts ds</i></li> </ul> $= ds$ $\oplus \{(n, d)$ $  \exists c$ <ul style="list-style-type: none"> <li>• <math>\neg c \in \text{visibleCols clear ts}</math></li> <li><math>\wedge n = \text{CS\_posn } c</math></li> <li><math>\wedge d = \text{CS\_default } c\}</math></li> </ul>
<b>insertQuery</b>	$\vdash \forall clear \ i \ ds \ s \ ts$ <ul style="list-style-type: none"> <li>• <i>insertQuery (clear, (i, ds), s, ts)</i></li> </ul> $= (\text{let } rl$ $= \text{Map}$ $(\text{MkRow clear } o \ \text{colDefaults clear } ts)$ $ds$

---

```

in if  $\neg$  Elems rl  $\subseteq$  RowS
then (s, [ambiguousColumn])
else
  (changeSpec
   i
   (replaceRows ts (TS_rows ts @ rl))
   s, [])

```

**deleteQuery**

```

 $\vdash \forall$  clear i ns e s ts
• deleteQuery (clear, (i, ns), s, ts)
= (let ns' = revealRow clear ts Image ns
in let ns''
    = ns'
     $\cap$  {i
      | R_exist (Nth (TS_rows ts) i) = clear}
in let rs
    = Extract
      (1 .. # (TS_rows ts) \ ns'')
      (TS_rows ts)
in changeSpec i (replaceRows ts rs) s)

```

**updateField**

```

 $\vdash \forall$  clear table_class table_d u
• updateField clear table_class (u, table_d)
= (if clear = table_class
then
  if isItem u
  then
    giveVal
      (MkData (Dat_class table_d) (destItem u))
  else if isClass u
  then
    if destClass u dominates Dat_class table_d
    then
      giveVal
        (MkData
         (destClass u)
         (Dat_item table_d))
      else giveError downGrade
    else giveVal (destData u)
  else if isItem u
  then
    if Dat_class table_d dominates clear
    then
      giveVal
        (MkData (Dat_class table_d) (destItem u))
      else giveError underClassified
    else giveError classChange)

```

**updateRow**

```

 $\vdash \forall$  clear table_class us r
• updateRow clear table_class (us, r)

```

---

```

= (if ¬ us ∈ Functional
  then giveError [ambiguousUpdate]
  else
    (let us'
      = RelCombine us (R_data r)
      % Graph
      (updateField clear table_class)
    in let es
      = (us' ▷ {x|isError x})
      % Graph destError
    in if es = {}
      then
        giveVal
          (MkRow
            (R_exist r)
            (R_data r ⊕ us' % Graph destVal))
      else giveError (RelList (Squash es))))

```

**updateQuery**

```

⊢ ∀ clear i us s ts
• updateQuery (clear, (i, us), s, ts)
  = (let colnums
    = {n
      | ∃ c
        • c ∈ visibleCols clear ts
          ∧ CS_posn c = n}
    in if ¬ us ∈ Functional
      then (s, [ambiguousUpdate])
      else if ¬ Dom (∪ (Ran us)) ⊆ colnums
        then (s, [noSuchColumn])
      else
        (let us' = revealRow clear ts ~ % us
          in let pr
            = RelCombine
              us'
              (ListRel (TS_rows ts))
            % Graph
            (updateRow clear (TS_class ts))
          in let es
            = (pr ▷ {x|isError x})
            % Graph destError
          in if es = {}
            then
              let rs
                = RelList
                  (ListRel (TS_rows ts)
                    ⊕ pr
                    % Graph destVal)
              in (changeSpec

```

```

                                i
                                (replaceRows ts rs)
                                s, [])
                                else (s, Flat (RelList (Squash es))))))
updateStateR ⊢ ∀ clear ef es s
• updateStateR (clear, (ef, es), s)
  = (if isSelect ef
    then (s, clear, destSelect ef, es)
    else if ¬ es = []
    then (s, clear, [], es)
    else
      (let i = tabFromEffect ef
        in if tabExists clear i s
        then
          let ts = getTable i s
            in if ¬ clear dominates TS_class ts
            then (s, clear, [], [accessDenied])
            else
              (let (s', es')
                = (if isInsert ef
                  then
                    insertQuery
                      (clear, destInsert ef, s,
                      ts)
                  else if isDelete ef
                  then
                    (deleteQuery
                      (clear,
                      destDelete ef, s,
                      ts), [])
                  else
                    updateQuery
                      (clear, destUpdate ef, s,
                      ts))
                in (s', clear, [], es'))
          else (s, clear, [], [noSuchTable])))
updateState ⊢ ∀ clear ef es s
• updateState (clear, (ef, es), s)
  = (let (sr, out)
    = updateStateR
      (clear, (ef, es), repState s)
    in (absState sr, out))

```

## 7 INDEX

<i>absState</i> .....	4
<i>changeSpec</i> .....	10
<i>cleanColCons</i> .....	6
<i>cleanDirectory</i> .....	9
<i>cleanRows</i> .....	7
<i>cleanRow</i> .....	7
<i>cleanTable</i> .....	8
<i>colDefaults</i> .....	12
<i>deleteQuery</i> .....	13
<i>dummyVal</i> .....	6
<i>fef005</i> .....	4
<i>filterRow</i> .....	6
<i>getTable</i> .....	11
<i>Hidden</i> .....	6
<i>hideR</i> .....	9
<i>hide</i> .....	9
<i>insertQuery</i> .....	12
<i>replaceData</i> .....	7
<i>replaceRows</i> .....	10
<i>repState</i> .....	4
<i>revealRow</i> .....	10
<i>tabExists</i> .....	11
<i>updateField</i> .....	13
<i>updateQuery</i> .....	15
<i>updateRow</i> .....	14
<i>updateStateR</i> .....	16
<i>updateState</i> .....	16
<i>visibleCols</i> .....	11