Project: DRA FRONT END FILTER PROJECT

Title: Specification of TSQL

Ref: DS/FMU/FEF/021 *Issue: Revision* : *2.2* *Date:* 5 June 2016

Status: Approved *Type:* Specification

Keywords:

Author:

Name	*Location*	*Signature*	*Date*
G. M. Prout	WIN01		

Authorisation for Issue:

Name	*Function*	*Signature*	*Date*
R.B. Jones	HAT Manager		

Abstract: The formal specifications of TSQL for the DRA front end filter project RSRE 1C/6130.

Distribution: HAT FEF File
Simon Wiseman

0 DOCUMENT CONTROL

0.1 Contents List

0.2 Document Cross References

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

[2] DS/FMU/FEF/005. *Specifications of hide and updateState.* G.M. Prout, ICL Secure Systems, WIN01.

[3] DS/FMU/FEF/010. *Proof of Security (IIa).* G.M. Prout, ICL Secure Systems, WIN01.

[4] DS/FMU/FEF/014. *Specification of SSQL Semantics II.* G.M. Prout, ICL Secure Systems, WIN01.

[5] DS/FMU/FEF/018. *Proposal for Phase 2.* G.M. Prout, ICL Secure Systems, WIN01.

[6] DS/FMU/FEF/022. *SWORD Front End Architectural Model.* R.D. Arthan, ICL Secure Systems, WIN01.

0.3 Changes History

Issue 1.1 (5 February 1993) First draft.

Issue 1.4 (23 February 1993) New parent *fef021*. *lattice_bottom* moved to DS/FMU/FEF/003.

Issue *Revision* : *2.2* **(5 June 2016)** Final approved version.

Issue 2.2 Removed dependency on ICL logo font

Issue 2.3 Stopped it reusing variable names from another document which was causing some make targets to fail.

0.4 Changes Forecast

Changes may be necessary as a result of issues raised during Phase 2.

1 GENERAL

1.1 Scope

This document gives a formal specification of the TSQL semantics. It constitutes deliverable D9 of work package 3, as given in the Proposal for Phase 2, [5].

1.2 Introduction

We propose defining the semantics of TSQL as a subset of SSQL with all classes in the state, except those that are actually stored as data, set at the lowest possible classification, *lattice_bottom*, and all worths *sterling*. This means that the *hide* function of [2] will have no effect on a TSQL state. A TSQL transition function will be comprised of two components $processQuery_t$ and $updateState_t$. We will define the function $processQuery_t$, which captures the functionality of the semantics of TSQL, to be the same as *processQuery*, specified in [4], except that it will be defined on TSQL states and will only return *Effect*s that have all classifications at *lattice_bottom* and all worths *sterling*. The function $updateState_t$ operates on the output from the function $processQuery_t$ and essentially determines whether or not the query should succeed.

A formal specification of the SWORD front end architectural model is given in [6]. The TSQL transition function specified here is described in [6] as the TSQL query processor of the conventional DBMS.

2 PRELIMINARIES

The following ProofPower instructions set up the new theory *fef021* and set the context for the proof tools.

```
SML
open_theory "fef024";
(force_delete_theory "fef021" handle _ => ());
new_theory"fef021";
push_pc "hol";
```

2.1 Consistency Proof for *repState* and *absState*

We have included here the consistency proofs for *repState* and *absState* which were defined in Phase 1 in [3].

```
SML
push_consistency_goal⌜repState⌝;
a(strip_asm_tac (simple_⇒_match_mp_rule type_lemmas_thm state_type_def));
a(∃_tac⌜(rep,abs)⌝ THEN asm_rewrite_tac[]);
a ∧_tac;
```

SML

```
(* *** Goal "1" *** *)
a(REPEAT strip_tac THEN_TRY asm_rewrite_tac[]);
a(POP_ASM_T (ante_tac o app_fun_rule⌜abs⌝) THEN asm_rewrite_tac[]);
(* *** Goal "2" *** *)
a(REPEAT ∀_tac THEN ⇒_tac);
a(REPEAT strip_tac THEN_TRY asm_rewrite_tac[]);
a(POP_ASM_T (ante_tac o app_fun_rule⌜rep⌝) THEN asm_rewrite_tac[]);
save_consistency_thm⌜repState⌝(pop_thm());
val fef021_repState_absState_def = get_spec⌜repState⌝;
```

We simplify one of the properties of the representation and abstraction functions.

SML

```
val fef021_isState_lemma1 = all_∀_intro(nth 1 (strip_∧_rule(all_∀_elim fef021_repState_absState_def)));
```

HOL output

```
fef021_isState_lemma1 = ⊢ isState r ⇒ repState (absState r) = r
```

2.2 Auxiliary Functions

A function that takes an *Item* and sets its worth to *sterling*.

HOL Constant

```
item_sterling : Item → Item
─────────────────────────────────────────
∀ i • item_sterling i = if isNullItem i then i
                        else
                        let v = destValuedItem i
                        in ValuedItemItem(MkValuedItem sterling (VI_val v))
```

A function that takes a piece of data and sets its class to bottom and its worth to sterling.

HOL Constant

```
class_bottom : Data → Data
─────────────────────────────────────────
∀ d • class_bottom d =      let i = Dat_item d
                            in MkData lattice_bottom (item_sterling i)
```

Now a function that takes a relation between column number and data and sets all classifications to *lattice_bottom* and worths to *sterling*.

HOL Constant

```
set_bottom_d : (Num ↔ Data) → (Num ↔ Data)
─────────────────────────────────────────
∀ nd • set_bottom_d nd = {(n,d)|n ∈ Dom nd ∧ d = class_bottom (nd @ n)}
```

A similar function that takes a relation between column number and update and sets all classifications to *lattice_bottom* and worths to *sterling*.

HOL Constant

$$\mathbf{set\_bottom_u} : (Num \leftrightarrow Update) \rightarrow (Num \leftrightarrow Update)$$

$$\forall\ nu \bullet set\_bottom_u\ nu =$$
$$\{(n,u) | n \in Dom\ nu \wedge u =$$
$$\quad let\ u' = nu\ @\ n$$
$$\quad in$$
$$\quad if\ isItem\ u'\ then\ ItemUpdate\ (item\_sterling(destItem\ u'))$$
$$\quad else\ if\ isClass\ u'\ then\ ClassUpdate\ lattice\_bottom$$
$$\quad else\ DataUpdate(class\_bottom\ (destData\ u'))\}$$

3 TSQL STATE

We define the TSQL state $State_t$ as a subset of the SSQL state, *State* : *Exp* specified in [1], where all classifications in the state are set at *lattice_bottom* except for those classifications that are stored as data. All worths in the state are set at *sterling*.

The constant $State_{tS}$ is the set of everything of type *State* : *Exp* where the required classifications are at *lattice_bottom* and worths at *sterling*.

HOL Constant

$$\mathbf{State_{tS}} : State\ \mathbb{P}$$

$$\begin{aligned}
State_{tS} = \{st \mid & \forall dir \bullet \\
& dir \in Ran\ (repState\ st) \\
& \Rightarrow Dir_exist\ dir = lattice_bottom \\
& \wedge\ Dir_class\ dir = lattice_bottom \\
& \wedge\ \forall\ tab \bullet \\
& \quad tab \in Ran(Dir_tables\ dir) \\
& \quad \Rightarrow TS_class\ tab = lattice_bottom \\
& \quad \wedge\ TS_maxRow\ tab = lattice_bottom \\
& \quad \wedge\ \forall\ col \bullet \\
& \quad\quad col \in TS_colspecs\ tab \\
& \quad\quad \Rightarrow CS_min\ col = lattice_bottom \\
& \quad\quad \wedge\ CS_max\ col = lattice_bottom \\
& \quad \wedge\ \forall\ cc \bullet \\
& \quad\quad cc \in Ran(TS_cons\ tab) \\
& \quad\quad \Rightarrow CC_exist\ cc = lattice_bottom \\
& \quad \wedge\ \forall\ row \bullet \\
& \quad\quad row \in Elems\ (TS_rows\ tab) \\
& \quad\quad \Rightarrow R_exist\ row = lattice_bottom \\
& \quad\quad \wedge\ \forall\ data \bullet \\
& \quad\quad\quad data \in Ran(R_data\ row) \\
& \quad\quad\quad \Rightarrow Dat_class\ data = lattice_bottom \\
& \quad\quad\quad \wedge\ isValuedItem\ (Dat_item\ data) \\
& \quad\quad\quad \Rightarrow VI_worth(destValuedItem(Dat_item\ data)) \\
& \quad\quad\quad = sterling\}
\end{aligned}$$

We define the property required on the representation state.

HOL Constant

$$\mathbf{isState_t} : State \rightarrow Bool$$

$$\forall\ s \bullet isState_t\ s \Leftrightarrow s \in State_{tS}$$

We demonstrate that the new type will be non-empty.

SML

```
push_goal([],⌜∃ s : State • isState_t s⌝);
a(rewrite_tac[get_spec⌜isState_t⌝]);
a(∃_tac⌜absState{}⌝);
a(rewrite_tac[get_spec⌜State_tS⌝]);
a(lemma_tac⌜isState{}⌝);
(* *** Goal "1" *** *)
a(rewrite_tac[get_spec⌜isState⌝,get_spec⌜StateS⌝,↦_def,functional_def,∩_def,↔_def]);
(* *** Goal "2" *** *)
a(asm_fc_tac[fef021_isState_lemma1]);
a(asm_rewrite_tac[get_spec⌜Ran⌝]);
```

Now the new type $State_t$ is defined.

SML

```
val state_t_type_def = new_type_defn(["state_t_type_def"],"State_t",[],pop_thm());
```

4 FUNCTIONALITY OF THE TSQL QUERY PROCESSING

The functionality of the semantics of TSQL will be captured by the function $processQuery_t$ which is similar to *processQuery*, specified in [4] except that it works on TSQL states, of type $State_t$, with user clearance at *lattice_bottom*. The effect of $processQuery_t$ is the same as that of *processQuery* with all classifications returned (except for those that are actually stored as data) set at *lattice_bottom*.

4.1 Representation and Abstraction Functions for $State_t$

First the representation function, $repState_t$, and abstraction function, $absState_t$, for the type $State_t$.

HOL Constant

$$
\begin{array}{|ll}
 & \mathbf{repState_t} : State_t \to State; \\
 & \mathbf{absState_t} : State \to State_t \\
\hline
 & (\forall\ a \bullet absState_t\ (repState_t\ a) = a) \\
\wedge & (\forall r \bullet isState_t\ r \Rightarrow repState_t\ (absState_t\ r) = r) \\
\wedge & (\forall\ a_1\ a_2 \bullet repState_t\ a_1 = repState_t\ a_2 \Leftrightarrow a_1 = a_2) \\
\wedge & (\forall\ r_1\ r_2 \bullet\ (isState_t\ r_1 \wedge isState_t\ r_2) \Rightarrow \\
 & \qquad (absState_t\ r_1 = absState_t\ r_2 \Leftrightarrow r_1 = r_2)) \\
\wedge & (\forall s \bullet isState_t\ (repState_t\ s))
\end{array}
$$

We prove the consistency of $repState_t$ and $absState_t$ and retrieve their definitions with the consistency obligation satisfied.

SML

```
push_consistency_goal⌜repState_t⌝;
a(strip_asm_tac (simple_⇒_match_mp_rule type_lemmas_thm state_t_type_def));
a(∃_tac⌜(rep,abs)⌝ THEN asm_rewrite_tac[]);
a ∧_tac;
```

SML

```
(* *** Goal "1" *** *)
a(REPEAT strip_tac THEN_TRY asm_rewrite_tac[]);
a(POP_ASM_T (ante_tac o app_fun_rule⌜abs⌝) THEN asm_rewrite_tac[]);
(* *** Goal "2" *** *)
a(REPEAT ∀_tac THEN ⇒_tac);
a(REPEAT strip_tac THEN_TRY asm_rewrite_tac[]);
a(POP_ASM_T (ante_tac o app_fun_rule⌜rep⌝) THEN asm_rewrite_tac[]);
save_consistency_thm⌜repState_t⌝(pop_thm());
val repState_t_absState_t_def = get_spec⌜repState_t⌝;
```

4.2 The function $processQuery_t$

HOL Constant

```
processQuery_t : Query × State_t → Effect × Errors
```

```
∀ q s • processQuery_t (q, s) =
        let (ef,es) = processQuery(q,lattice_bottom,repState_t s)
        in
        let ef' =
        if isInsert ef
        then    let (t,ndl) = destInsert ef
                in InsertEffect(t,Map set_bottom_d ndl)
        else if isDelete ef then ef
        else if isUpdate ef
        then    let (t,nnu) = destUpdate ef
                in UpdateEffect(t,{(n,nu)|n ∈ Dom nnu ∧ nu = set_bottom_u(nnu @ n)})
        else    let dll = destSelect ef
                in SelectEffect(Map (Map class_bottom) dll)
        in (ef',es)
```

5 UPDATING A TSQL STATE

HOL Constant

$$\mathbf{updateState_t} : (Effect \times Errors) \times State_t \rightarrow State_t \times (Data\ LIST\ LIST \times Errors)$$

$\forall\ efes : Effect \times Errors;\ s : State_t$

• $updateState_t(efes,s) = let\ (s',(c,out)) = updateState(lattice\_bottom,efes,repState_t\ s)\ in\ (absState_t\ s',out)$

6 TSQL TRANSITION FUNCTION

A TSQL transition function is to be built from the two components $updateState_t$ and $processQuery_t$. We first give abbreviation definitions for the types $Process_t$ of the $processQuery_t$ component and $Ustate_t$ of the $updateState_t$ component.

SML

```
declare_type_abbrev("Process_t",[],⌜: Query × State_t → Effect × Errors⌝);
```

SML

```
declare_type_abbrev("Ustate_t",[],⌜: (Effect × Errors) × State_t
                                  → State_t × (Data LIST LIST × Errors)⌝);
```

We also give an abbreviation definition for the type of TSQL state transition functions, tf_t.

SML

```
declare_type_abbrev("tf_t",[],⌜: Query × State_t → State_t × (Data LIST LIST × Errors)⌝);
```

6.1 Building a Transition Function

We define a function, $MkTf_t$, which builds a transition function from two components: a component of type $Process_t$ and a component of type $Ustate_t$. The resulting transition function updates the original state of the database by using the result of processing a query on the state of the database.

HOL Constant

$$\mathbf{MkTf_t} : Process_t \rightarrow Ustate_t \rightarrow tf_t$$

$\forall p{:}Process_t;\ u{:}Ustate_t;\ q : Query;\ s : State_t$

• $(MkTf_t\ p\ u)\ (q,s) = u(p(q,s),s)$

6.2 The TSQL Transition Function

A TSQL transition function is a transition function built from the two components $processQuery_t$ and $updateState_t$.

HOL Constant

TSQLtf : tf_t

$$TSQLtf = MkTf_t\ processQuery_t\ updateState_t$$

7 INDEX