
Project: DRA FRONT END FILTER PROJECT

Title: Specification of Query Transformations in SML (I)

Ref: DS/FMU/FEF/019 *Issue: Revision : 2.2* *Date:* 5 December 2009

Status: Approved *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: Preliminaries to a Standard ML specification of the SSQL Query Transformations for the DRA front end filter project RSRE 1C/6130.

Distribution: HAT FEF File
Simon Wiseman

0 DOCUMENT CONTROL

0.1 Contents List

0	DOCUMENT CONTROL	2
0.1	Contents List	2
0.2	Document Cross References	2
0.3	Changes History	2
0.4	Changes Forecast	3
1	GENERAL	3
1.1	Scope	3
1.2	Introduction	3
2	UNIVERSAL TYPES	3
3	SSQL ABSTRACT SYNTAX	5
3.1	Sorts	5
4	TSQL ABSTRACT SYNTAX	9
4.1	Sorts	9
5	SSQL TRANSFORMATIONS	13
5.1	Generic Functions and Primitive Routines	13
5.2	Transformation Notation	16
5.3	Types	17
5.4	The Symbol Table	19
6	INDEX	20

0.2 Document Cross References

- [1] L.Paulson. *ML for the Working Programmer*. Cambridge University Press, 1991.
- [2] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. MIT Press, 1990.
- [3] DS/FMU/FEF/018. *Proposal for Phase 2*. G.M. Prout, ICL Secure Systems, WIN01.
- [4] DS/FMU/FEF/020. *Specification of Query Transformations in SML (II)*. G.M. Prout, ICL Secure Systems, WIN01.
- [5] *SSQL Transformations*. Simon Wiseman, DRA, 14th January 1993.

0.3 Changes History

- Issue 1.1 (27 January 1993)** First draft.
- Issue 1.2 (28 January 1993)** SSQL and TSQL specs separated from transformations.

Issue Revision : 2.2 (5 December 2009) Final approved version.

Issue 2.2 Removed dependency on ICL logo font

0.4 Changes Forecast

None.

1 GENERAL

1.1 Scope

This document gives specifications in Standard ML ([2], [1]) of the SSQL and TSQL abstract syntax and preliminary specifications from [5] to support a formal specification in Standard ML of the SSQL query transformations of [5].

It constitutes part of deliverable D8 of work package 3, as given in the Proposal for Phase 2, [3].

1.2 Introduction

We proposed in [3] to formalise the SSQL query transformations of [5] in Standard ML. Here we provide Standard ML specifications of the SSQL and TSQL datatypes together with specifications of the transformation notation, generic functions, primitive routines, transformation types and symbol table from [5]. Standard ML specifications of the transformations may be found in [4].

In order to avoid overloading of Standard ML identifiers, subscripts have been used to distinguish between constructors of SSQL types and TSQL types. Subscripts or abbreviations have also been used in the case where an identifier used in [5] is a reserved word in Standard ML (e.g. *val*).

2 UNIVERSAL TYPES

We will use SML type *int*, *bool*, *unit*, *list* and *string* where appropriate. At this stage, the rest of the types are defined each with a single constructor.

SML

```
datatype Fixed = mk_fixed;
datatype Floating = mk_floating;
datatype Enum = mk_enum;
datatype Time = mk_time;
datatype Interval = mk_interval;
datatype Class = mk_class;
datatype Code = mk_code;
```

We define a datatype *Op*, rather than use type *Integer*.

SML

```
datatype Op =  
    not_op  
  | definitely_op  
  | possibly_op  
  | minus_op  
  | ord_op  
  | char_op  
  | upper_op  
  | lower_op  
  | plus_op  
  | minusd_op  
  | times_op  
  | divide_op  
  | concat_op  
  | and_op  
  | or_op  
  | less_than_op  
  | less_or_equal_op  
  | greater_or_equal_op  
  | greater_than_op  
  | equal_op  
  | not_equal_op  
  | lub_op  
  | glb_op  
  | dom_op  
  | dom_by_op  
  | liked_op  
  | maximum_op  
  | minimum_op  
  | between_op  
  | liket_op;
```

3 SSQL ABSTRACT SYNTAX

3.1 Sorts

SML

```

datatype Col_specssql =
  | denote_col_specs of string
  | absolute_col_specs of string list * string * string
  | default_col_specs of int * string list * string * string;

datatype Table_specssql =
  | absolute_table_specs of string list * string
  | default_table_specs of int * string list * string;

datatype Typessql =
  | monolean_types
  | boolean_types
  | string_types of int * int
  | fixed_types of int * int
  | floating_types of int * int * int
  | enum_types of int * Table_specssql
  | time_types of string
  | interval_types of string
  | class_types
  | code_types
  | any_types;

```

SML

```

datatype Clausessql =
  | where_clauses
  | group_by_clauses
  | having_clauses
  | set_clauses
  | select_clauses
  | constraint_clauses;

```

SML

```
datatype Constant_valuessql =  
    | denote_nulls  
    | denote_voids  
    | denote_trues  
    | denote_falses  
    | denote_strings of string * Typessql  
    | denote_fixeds of Fixed * Typessql  
    | denote_floatings of Floating * Typessql  
    | denote_enums of Enum * Typessql  
    | denote_times of Time * Typessql  
    | denote_intervals of Interval * Typessql  
    | denote_classs of Class  
    | denote_codes of Code;
```

SML

```

datatype Valuessql =
  | denote_constants of Constant_valuessql
  | monops of Op * Valuessql
  | binops of Op * (Valuessql * Valuessql)
  | triops of Op * (Valuessql * Valuessql * Valuessql)
  | converts of Valuessql * Typessql
  | convert_domains of Valuessql * Table_specssql * Typessql
  | make_sterlings of Valuessql
  | make_dinarys of Valuessql
  | declares of string * (Valuessql * Valuessql)
  | caseVals of Valuessql * Valuessql list * Valuessql list * Valuessql
  | cases of Valuessql list * Valuessql list * Valuessql
  | set_func_alls of Op * Valuessql
  | set_func_distincts of Op * Valuessql
  | count_non_nulls of Valuessql * Typessql
  | count_distincts of Valuessql * Typessql
  | count_alls of Typessql
  | all_binops of Op * (Valuessql * Tuple_listssql)
  | some_binops of Op * (Valuessql * Tuple_listssql)
  | all_binop_lists of Op * (Valuessql * Valuessql list)
  | some_binop_lists of Op * (Valuessql * Valuessql list)
  | exists_tupless of Tuple_listssql
  | single_values of Tuple_listssql
  | contentss of Col_specssql
  | sterling_contentss of Col_specssql
  | dinary_contentss of Col_specssql
  | classifications of Col_specssql
  | row_existences of Table_specssql
  | joined_row_existences
  | classifys of Valuessql * Valuessql
  | classify_defaults of Valuessql
  | observeds of Col_specssql * Clausessql
  | modifieds of Col_specssql
  | contexts of string
  | parameters of string

```

SML

```

and From_specssql =      froms of Tuple_listssql
                        | correlate_froms of string * Tuple_listssql
and Tuple_listssql =    table_contentss of Table_specssql
                        | all_tupless of Select_listssql * From_specssql list * Valuessql
                        * Col_specssql list * Col_specssql list * Col_specssql list * Valuessql
                        | distinct_tupless of Select_listssql * From_specssql list * Valuessql
                        * Col_specssql list * Col_specssql list * Col_specssql list * Valuessql
                        | evaluates of Select_listssql * From_specssql list * Valuessql
                        * Col_specssql list * Col_specssql list * Col_specssql list * Valuessql
                        | tuples of Valuessql list
                        | unions of Tuple_listssql list
                        | name_columnss of string list * Tuple_listssql
and Select_listssql =    all_columnss
                        | select_valuess of Select_valuessql list
and Select_valuessql =  anonymous_values of Valuessql
                        | named_values of string * Valuessql
                        | anonymous_pairs of Valuessql * Valuessql
                        | named_pairs of string * (Valuessql * Valuessql);

```

SML

```

datatype Col_namessql =  denote_col_names of string;
datatype Target_specssql = targets of Table_specssql
                        | correlate_targets of string * Table_specssql;

```

SML

```

datatype Set_clausessql = set_values of Col_namessql * Valuessql
                        | set_classs of Col_namessql * Valuessql
                        | set_class_and_values of Col_namessql * Valuessql * Valuessql;

```

SML

```

datatype Queryssql = inserts of Table_specssql * Col_namessql list * Tuple_listssql
| deletes of Target_specssql * Valuessql * Col_specssql list
  * Col_specssql list * Col_specssql list * Valuessql
| updates of Target_specssql * Set_clausessql list * Valuessql
  * Col_specssql list * Col_specssql list * Col_specssql list
  * Valuessql
| selects of Tuple_listssql
| positioned_deletes of Target_specssql * Select_listssql
  * Valuessql * Col_specssql list * Col_specssql list
  * Col_specssql list * Valuessql
| positioned_updates of Target_specssql * Col_namessql list
  * Col_namessql list * Select_listssql * Valuessql
  * Col_specssql list * Col_specssql list * Col_specssql list
  * Valuessql
| commit
| rollback;

```

SML

```

datatype BoundQueryssql = binds of string list * Class list * Constant_valuessql list
  * Queryssql

```

4 TSQL ABSTRACT SYNTAX

4.1 Sorts

SML

```

datatype Col_spectsql = denote_col_spect of string
| absolute_col_spect of string list * string * string;

datatype Table_spectsql = absolute_table_spect of string list * string;

```

SML

```
datatype Typetsql =  monolean_type_t
                    | boolean_type_t
                    | string_type_t of int * int
                    | fixed_type_t of int * int
                    | floating_type_t of int * int * int
                    | enum_type_t of int * string
                    | time_type_t of string
                    | interval_type_t of string
                    | class_type_t
                    | code_type_t
                    | any_type_t;
```

SML

```

datatype Valuetsql =
    denote_nullt
  | denote_voidt
  | denote_truet
  | denote_falset
  | denote_stringt of string * Typetsql
  | denote_fixedt of Fixed * Typetsql
  | denote_floatingt of Floating * Typetsql
  | denote_enumt of Enum * Typetsql
  | denote_timet of Time * Typetsql
  | denote_intervalt of Interval * Typetsql
  | denote_classt of Class
  | denote_codet of Code
  | monopt of Op * Valuetsql
  | binopt of Op * (Valuetsql * Valuetsql)
  | triopt of Op * (Valuetsql * Valuetsql * Valuetsql)
  | convertt of Valuetsql * Typetsql
  | convert_domaint of string * (Valuetsql * Typetsql)
  | declaret of string * (Valuetsql * Valuetsql)
  | caseValt of Valuetsql * Valuetsql list * Valuetsql list * Valuetsql
  | caset of Valuetsql list * Valuetsql list * Valuetsql
  | set_func_allt of Op * Valuetsql
  | set_func_distinctt of Op * Valuetsql
  | count_non_nullt of Valuetsql * Typetsql
  | count_distinctt of Valuetsql * Typetsql
  | count_allt of Typetsql
  | all_binopt of Op * (Valuetsql * Tuple_listtsql)
  | some_binopt of Op * (Valuetsql * Tuple_listtsql)
  | all_binop_listt of Op * (Valuetsql * Valuetsql list)
  | some_binop_listt of Op * (Valuetsql * Valuetsql list)
  | exists_tuplest of Tuple_listtsql
  | single_valuet of Tuple_listtsql
  | contentst of Col_spectsql

```

SML

```

and From_spectsql =      fromt of Tuple_listtsql
                        | correlate_fromt of string * Tuple_listtsql
and Tuple_listtsql =    table_contentst of Table_spectsql
                        | all_tuplest of Select_listtsql * From_spectsql list * Valuetsql
                          * Col_spectsql list * Valuetsql
                        | distinct_tuplest of Select_listtsql * From_spectsql list * Valuetsql
                          * Col_spectsql list * Valuetsql
                        | evaluatet of Select_listtsql * From_spectsql list * Valuetsql
                          * Col_spectsql list * Valuetsql
                        | tuplet of Valuetsql list
                        | uniont of Tuple_listtsql list
                        | name_columnst of string list * Tuple_listtsql

```

SML

```

and Select_listtsql =    all_columnst
                        | select_valuest of Select_valuetsql list
and Select_valuetsql =  anonymous_valuet of Valuetsql
                        | named_valuet of string * Valuetsql;

```

SML

```

datatype Col_nametsql =  denote_col_namet of string;
datatype Target_spectsql = targett of Table_spectsql
                        | correlate_targett of string * Table_spectsql;

```

SML

```

datatype Set_clausetsql = set_valuet of Col_nametsql * Valuetsql;

```

SML

```

datatype Querytsql =    insertt of Table_spectsql * Col_nametsql list * Tuple_listtsql
                        | deletet of Target_spectsql * Valuetsql * Col_spectsql list
                          * Valuetsql
                        | updatet of Target_spectsql * Set_clausetsql list * Valuetsql
                          * Col_spectsql list * Valuetsql
                        | selectt of Tuple_listtsql
                        | positioned_deletet of Target_spectsql * Select_listtsql
                          * Valuetsql * Col_spectsql list * Valuetsql
                        | positioned_updatet of Target_spectsql * Col_spectsql list
                          * Select_listtsql * Valuetsql * Col_spectsql list * Valuetsql
                        | commit
                        | rollback;

```

5 SSQL TRANSFORMATIONS

5.1 Generic Functions and Primitive Routines

We will use the standard ML functions \wedge for concatenation of strings, *implode* and *explode* for handling strings as lists of characters (and vice versa), *size* for the number of characters in a string, @ for concatenation of lists ($\&$ in [5]), *rev* for reversing a list and *map* to apply a function to all the elements of a list (\ast in [5]).

```

| _ ^ _      : string * string -> string
| implode   : string list -> string
| explode   : string -> string list
| size      : string -> int
| _ @ _      : 'a list * 'a list -> 'a list
| rev       : 'a list -> 'a list
| map       : ('a -> 'b) -> 'a list -> 'b list

```

The function *seq*.

SML

```

| exception negative of string;
| fun (seq : int * 'a -> 'a list) (0,x) = []
|      seq (n,x) = if n > 0 then (x :: (seq (n-1,x))) else raise negative "seq";

```

We will use the function *length* for the length of a list, ($\#$ in [5]) and *fold* to fold a list into a single value ($\&$ in [5]).

SML

```

| fun (length : 'a list -> int) (x :: xs) = 1 + (length xs)
|      length [] = 0;

```

SML

```

| exception emptylist;
| fun (fold : ('a * 'a -> 'a) -> 'a list -> 'a) f [] = raise emptylist
|      fold f [x] = x
|      fold f (h :: t) = f(h,fold f t);

```

We define a function *curry*:

SML

```

| fun (curry : ('a * 'b -> 'c) -> 'a -> 'b -> 'c) f = (
|      fn a => (fn b => f(a,b)));

```

Now a function *combine2* (and *combine3* and *combine4*), which takes a pair (triple,..) of lists and returns a list of pairs (triples,..). An error is raised if the lists are of different lengths.

SML

```

|exception diffLengths of string;
|fun   (combine2 : 'a list -> 'b list -> ('a * 'b) list)
|      [] [] = []
|      combine2 (h1::t1)(h2::t2) = (h1,h2)::(combine2 t1 t2)
|      combine2 _ _ = raise diffLengths "combine2";

```

SML

```

|fun   (combine3 : 'a list -> 'b list -> 'c list -> ('a * 'b * 'c) list)
|      [] [] [] = []
|      combine3 (h1::t1)(h2::t2)(h3::t3) = (h1,h2,h3)::(combine3 t1 t2 t3)
|      combine3 _ _ _ = raise diffLengths "combine3";

```

SML

```

|fun   (combine4 : 'a list -> 'b list -> 'c list -> 'd list -> ('a * 'b * 'c * 'd) list)
|      [] [] [] [] = []
|      combine4 (h1::t1)(h2::t2)(h3::t3)(h4::t4) = (h1,h2,h3,h4)::(combine4 t1 t2 t3 t4)
|      combine4 _ _ _ _ = raise diffLengths "combine4";

```

Now the function *at2* (and *at3* and *at4*) (@ in [5]) which turns a function taking a sequence of pairs (triples,..) into one taking a pair (triple,..) of sequences.

SML

```

|fun   (at2 : (('a * 'b ) list -> 'c) -> ('a list * 'b list -> 'c)) f = (
|      fn (a,b) => f(combine2 a b));

```

SML

```

|fun   (at3 : (('a * 'b * 'c ) list -> 'd) -> ('a list * 'b list * 'c list -> 'd)) f = (
|      fn (a,b,c) => f(combine3 a b c));

```

SML

```

|fun   (at4 : (('a * 'b * 'c * 'd ) list -> 'e) ->
|          ('a list * 'b list * 'c list * 'd list -> 'e)) f = (
|      fn (a,b,c,d) => f(combine4 a b c d));

```

Classifications are partially ordered by *dom*. The exception *notDefined* will be used in definitions of functions from [5] which are incompletely specified.

SML

```

|exception notDefined of string;

```

SML

```

|fun   (dom: Class * Class -> bool) (a,b) = raise notDefined "dom";
|infix dom;

```

SML

```
| fun (lub: Class * Class -> Class) (a,b) = raise notDefined "lub";
| infix lub;
```

SML

```
| fun (lattice_top: unit -> Class) () = raise notDefined "lattice_top";
```

SML

```
| fun (lattice_bottom: unit -> Class) () = raise notDefined "lattice_bottom";
```

We define infix functions *max* and *min* on integers.

SML

```
| infix min;
| fun (a : int) min (b : int) = if a < b then a else b;
| infix max;
| fun (a : int) max (b : int) = if a > b then a else b;
```

Head and Tail of a list.

SML

```
| exception Hd;
| fun (hd : 'a list -> 'a)
|   [] = raise Hd
|   hd(h::t) = h;
| exception Tl;
| fun (tl : 'a list -> 'a list)
|   [] = raise Tl
|   tl(h::t) = t;
```

The functions *invert* and *split*.

SML

```
| fun (invert : 'a list list -> 'a list list)
|   ([] :: -) = []
|   invert x = map hd x :: invert (map tl x);
```

SML

```
| fun (split: ('a * 'b) list -> 'a list * 'b list) [] = ([],[])
|   split ((h1,h2):: t) = let val (t1,t2) = split t
|                           in (h1 :: t1,h2 :: t2)
|                           end;
```

SML

```

| fun (split3: ('a * 'b * 'c) list -> 'a list * 'b list * 'c list) [] = ([],[],[])
|   split3 ((h1,h2,h3):: t) = let val (t1,t2,t3) = split3 t
|                               in (h1 :: t1,h2 :: t2,h3 :: t3)
|                               end;

```

SML

```

| fun (split4: ('a * 'b * 'c * 'd) list -> 'a list * 'b list * 'c list * 'd list) [] = ([],[],[],[])
|   split4 ((h1,h2,h3,h4):: t) = let val (t1,t2,t3,t4) = split4 t
|                               in (h1 :: t1,h2 :: t2,h3 :: t3,h4 :: t4)
|                               end;

```

SML

```

| fun (split5: ('a * 'b * 'c * 'd * 'e) list -> 'a list * 'b list * 'c list * 'd list * 'e list) []
|   = ([],[],[],[],[])
|   split5 ((h1,h2,h3,h4,h5):: t) = let val (t1,t2,t3,t4,t5) = split5 t
|                               in (h1 :: t1,h2 :: t2,h3 :: t3,h4 :: t4,h5 :: t5)
|                               end;

```

Finally, the function *or*.

SML

```

| fun (or : bool * bool -> bool) (b1,b2) = b1 orelse b2;

```

5.2 Transformation Notation

The primitive types are given in section 2. We will use the standard ML type *unit* in place of *Null*. We give a datatype *Sum* to allow us to form the disjoint sum of two types.

SML

```

| datatype ('a , 'b)Sum = inL of 'a | inR of 'b;

```

Then discriminator and destructor functions for a sum type.

SML

```

| fun (isL: ('a , 'b)Sum -> bool) (inL x) = true
|   isL _ = false;
| fun (isR: ('a , 'b)Sum -> bool) (inR x) = true
|   isR _ = false;
| exception notLeft;
| fun (getL: ('a , 'b)Sum -> 'a) (inL x) = x
|   getL _ = raise notLeft;
| exception notRight;
| fun (getR: ('a , 'b)Sum -> 'b) (inR x) = x
|   getR _ = raise notRight;

```

SML

```

| datatype Monolean = void;

```

5.3 Types

SML

```

datatype TableSpecification =      absolute of string list * string
                                   | default of int * string list * string;

datatype SwordType =              nullType
                                   | monoleanType
                                   | booleanType
                                   | stringType of int * int
                                   | fixedType of int * int
                                   | floatingType of int * int * int
                                   | enumType of int * TableSpecification
                                   | timeType of string
                                   | intervalType of string
                                   | classType
                                   | codeType
                                   | anyType;

```

SML

```

datatype Worth =                  priceless
                                   | worthless
                                   | sterling
                                   | dinary;

```

SML

```

datatype SsqlName =               anons
                                   | names of string;
datatype TsqlName =              nonet
                                   | anont
                                   | namet of string;

```

SML

```

type ColType = SwordType * SwordType;

```

SML

```

datatype BoundInfo =             upb of Class
                                   | constant of Class;

```

SML

```

type SsqlCol =                   { name:SsqlName,
                                   type_field:ColType,
                                   col_exist:Class,
                                   col_class:BoundInfo};

```

SML

```
datatype TsqlClassName = anontc
                        | nametc of string
                        | constanttc of Class;
```

SML

```
type TsqlCol = {sterling_name:TsqlName,
               dinary_name:TsqlName,
               class_name:TsqlClassName};
```

SML

```
type TableInfo = {table_exist_class:Class,
                 table_class:Class,
                 row_class:BoundInfo};
```

SML

```
type ConstraintInfo = {null_allowed:bool list,
                      lwb:Class list,
                      unique:int list list,
                      uniform:int list list,
                      index:int list list};
```

SML

```
datatype ColumnSpecification = anonymous_column of string
                              | specific of TableSpecification * string;
```

SML

```
datatype TsqlRepr = local_identifier of string
                  | column of string * string
                  | constant_class of Class
                  | constant_null;
```

SML

```
type ExpType = SwordType * Worth;
```

SML

```
datatype ExpClass = variable of Valuetsql * Class
                  | constantec of Class
```

SML

```
datatype InternalExpClass = ands of Valuetsql list * ExpClass list
                          | ors of Valuetsql list * ExpClass list
                          | simple of ExpClass;
```

We give a new type for use in entering details of tables to the symbol table.

SML

```
| datatype TableName =      anontn
|                            | nametn of TableSpecification;
```

5.4 The Symbol Table

SML

```
| type TableDetail = { tableName : TableName,
|                       corrName : SsqlName,
|                       genCorr : string,
|                       info : TableInfo,
|                       columns : SsqlCol list,
|                       rowClass : TsqlClassName,
|                       implementation : TsqlCol list,
|                       constraints : ConstraintInfo};
```

SML

```
| type IdentDetail = { identName : string,
|                       info : ExpType,
|                       lubid : Class,
|                       vName : string,
|                       cName : TsqlName};
```

SML

```
| type Scope = { tables : TableDetail list,
|                identifiers : IdentDetail list};
```

SML

```
| type ParamInfo = { name : string,
|                       valp : Constant_valuessql,
|                       clasf : Class};
```

SML

```
| val symbolTable = ref ([]:Scope list);
| val parameterTable = ref ([]:ParamInfo list);
```

6 INDEX

<i>at2</i>	14	<i>Query_{ssql}</i>	9
<i>at3</i>	14	<i>Query_{tsql}</i>	12
<i>at4</i>	14	<i>rev</i>	13
<i>BoundInfo</i>	17	<i>Scope</i>	19
<i>BoundQuery_{ssql}</i>	9	<i>Select_list_{ssql}</i>	8
<i>Class</i>	3	<i>Select_list_{tsql}</i>	12
<i>Clause_{ssql}</i>	5	<i>Select_value_{ssql}</i>	8
<i>Code</i>	3	<i>Select_value_{tsql}</i>	12
<i>ColType</i>	17	<i>seq</i>	13
<i>ColumnSpecification</i>	18	<i>Set_clause_{ssql}</i>	8
<i>Col_name_{ssql}</i>	8	<i>Set_clause_{tsql}</i>	12
<i>Col_name_{tsql}</i>	12	<i>size</i>	13
<i>Col_spec_{ssql}</i>	5	<i>split3</i>	16
<i>Col_spec_{tsql}</i>	9	<i>split4</i>	16
<i>combine2</i>	14	<i>split5</i>	16
<i>combine3</i>	14	<i>split</i>	15
<i>combine4</i>	14	<i>SsqlCol</i>	17
<i>Constant_value_{ssql}</i>	6	<i>SsqlName</i>	17
<i>ConstraintInfo</i>	18	<i>Sum</i>	16
<i>curry</i>	13	<i>SwordType</i>	17
<i>dom</i>	14	<i>TableDetail</i>	19
<i>Enum</i>	3	<i>TableInfo</i>	18
<i>ExpClass</i>	18	<i>TableName</i>	19
<i>explode</i>	13	<i>TableSpecification</i>	17
<i>ExpType</i>	18	<i>Table_spec_{ssql}</i>	5
<i>Fixed</i>	3	<i>Table_spec_{tsql}</i>	9
<i>Floating</i>	3	<i>Target_spec_{ssql}</i>	8
<i>fold</i>	13	<i>Target_spec_{tsql}</i>	12
<i>From_spec_{ssql}</i>	8	<i>Time</i>	3
<i>From_spec_{tsql}</i>	12	<i>tl</i>	15
<i>getL</i>	16	<i>TsqlClassName</i>	18
<i>getR</i>	16	<i>TsqlCol</i>	18
<i>hd</i>	15	<i>TsqlName</i>	17
<i>IdentDetail</i>	19	<i>TsqlRepr</i>	18
<i>implode</i>	13	<i>Tuple_list_{ssql}</i>	8
<i>InternalExpClass</i>	18	<i>Tuple_list_{tsql}</i>	12
<i>Interval</i>	3	<i>Type_{ssql}</i>	5
<i>invert</i>	15	<i>Type_{tsql}</i>	10
<i>isL</i>	16	<i>Value_{ssql}</i>	7
<i>isR</i>	16	<i>Value_{tsql}</i>	11
<i>lattice_bottom</i>	15	<i>Worth</i>	17
<i>lattice_top</i>	15	<i>_@_</i>	13
<i>length</i>	13	<i>^_</i>	13
<i>lub</i>	15		
<i>map</i>	13		
<i>max</i>	15		
<i>min</i>	15		
<i>Monolean</i>	16		
<i>Op</i>	4		
<i>or</i>	16		
<i>ParamInfo</i>	19		