DSM SQL Rules V2

 | Updated today at 12:15 PM by David A. Wood | Tags: *None*

The DSM Toolkit provides a DSMEngine class that receives tuples and then processes them through a set of rules to develop new tuples, some of which may be sent to other tuple applications as defined by the type and contents of the tuples. A rule evaluation sequence is performed for every new tuple additions provided to the engine. Tuple state changes may be provided directly through DSMEngine instance method calls or via tuple sent from other applications over an ITupleSender/ITupleReceiver pair. The rules language defines 3 types of tuples as follows:

- Input - these are non-persistent tuples that are not manipulated by the rules, but which can be used as input to rules that change other tuples..
- Persistent - these are persistent tuples that are set through evaluation of rules and stored across evaluations.
- Transport - these are temporary tuples that are sent to other tuple applications after a rule evaluation sequence. The first tuple entry in a tuple must be an application instance identifier which specifies the application to which the tuple will be sent.

All tuples are operated on through database tables. As such the rule language provides thre types of statements: table creation, initialization SQL, and rule clauses. Table creation creates the tables through SQL CREATE statements. Initialization statements executed before the first evaluation request. Rule clauses create rows of tuples from existing tables using SQL SELECT statements.

Rule Syntax Overview

The rule language uses rule sections to distinguish rule attributes, the creation of and types of tables, initialization, and the rule clauses. Each section is identified by one of the section-beginning tokens

- ATTRIBUTES - this sections contains name value pairs that may be used to modify behaviors.
- INPUT- this section contains statements that create Input tables.
- PERSISTENT- this section contains statements that create Persistent tables.
- TRANSPORT - this section contains statements that create Transport tables.
- INITIALIZE - lists SQL statements(create or insert) that are to executed before the first rule evaluation.
- CLAUSE - this section contains statements that define table modifications (adds or

deletes).

The ATTRIBUTES sections contains property-like name value pairs, which the value terminated by a ';'. The following are supported attributes

- **version.major** : a value of 1 uses Version 1 rule evaluation semantics. Version 2 uses the evaluation semantics listed below.

The rules, initialization and declarations sections contain one or more statements of the form

- <ID><MODIFIERS> : <SQL/DDL>

where the ID is the name of the table being created (in the case of table creation or initialization statements) or the table being modified (in the case of initialization or clause statements). In the case of a clause statement, the ID may not be that of an Input table. The SQL statement is either a table CREATE statement (for Input, Persistent, Transport section statements) or a SELECT statement (for Clause section statements). For example,

- `LINK: CREATE TEMPORARY TABLE LINK (src varchar(128), dest varchar(128));`
- `PATH: SELECT DISTINCT * FROM LINK;`

Table Declarations

Table creation statements provide and SQL statement to establish a table within the database schema. For example,

```
PERSISTENT
LINK: CREATE TEMPORARY TABLE LINK (src varchar(128), dest
varchar(128));
```

creates a persistent table named LINK containing columns src and dest. Notice that the ID of the statement is also LINK, as required. Declarations may include the following modifiers:

- `PRIMARY=<identifier>`
- `PREVIOUS=<identifier>`
- `AUTOMATIC`

The PRIMARY and PREVIOUS modifiers are used together in persistent table declaration to link two tables to each other. One of the tables is the primary table over which rules are generally written, and the PREVIOUS table is used by the evaluation algorithm to hold values from the PRIMARY table during evaluation. Both tables must have the same schema. The value of the PRIMARY modifier is the name of the table with a PREVIOUS modifier having a value that names the primary table. For example,

```
....
PERSISTENT
```

```
myprimary[PRIMARY=secondary] : CREATE TEMPORARY TABLE myprimary
...
secondary[AUTOMATIC,PREVIOUS=myprimary] : CREATE TEMPORARY TABLE
secondary ...
....
```

The AUTOMATIC modifier marks a table as having been generated by an automated system such as a compiler. Automatic tables are not included in results of the engine Java API calls, such as getTupleState().

Initialization Section

Statements may be defined which are executed prior to the first rule evaluation. Statements are executed in the order they are defined and may be either SQL INSERT or CREATE statements. The AUTOMATIC modifier may also be present. For example,

```
...
Initialize
DUMMY_INTERNAL_TABLE_[AUTOMATIC]: CREATE TABLE
DUMMY_INTERNAL_TABLE_ (X int);
DUMMY_INTERNAL_TABLE_[AUTOMATIC]: INSERT INTO
DUMMY_INTERNAL_TABLE_ VALUES (1);
...
```

These initialization statements serve to create a table and place a single row into it prior to the first rule evaluation.

Clause Section

Clause statements are used to assign new rows into the table identified by the statement ID. The SQL is a SELECT statement which must produce a set of rows with the same schema as the table being assigned to, which is identified by the statement ID. For example,

```
PATH: SELECT * FROM PATH2;
```

copies all rows from the PATH2 table into the PATH table. Both tables have the same schema. More complex statements can be defined, the only requirement is that the SQL result set as the same schema (i.e. columns number and types) as the destination.

Clause statements may used one or more of the following modifiers:

- `MSEC=<N>`
- `CHOOSE=<N> [, CHOOSE_MIN=<M>]`
- `AUTOMATIC`

The MSEC modifier turns the statement into a clause that is only executed during a rule evaluation if N milliseconds have expired since the last evaluation of the statement.

The CHOOSE modifier indicates that a random selection of N tuples should be chosen from

the SELECT or DELETE results. If CHOOSE_MIN is specified, then a minimum of M results must be result, otherwise no results will be produced from the statement.

Some valid examples are as follow:

- `LINK[MSEC=250]: SELECT * FROM CLINK;`
- `LINK[CHOOSE=5,CHOOSE_MIN=2,,MSEC=250]: SELECT * FROM CLINK;`

In some cases, CLAUSE statements may be dependent on others CLAUSE statements making the order of rule evaluation important. To accommodate this, CLAUSEs may be grouped in blocks using the BLOCK keyword. In general, the BLOCK keyword may be used anywhere within the CLAUSE section, as follows:

```
CLAUSE
BLOCK
<ID> : <SQL>
....
BLOCK
<ID>: <SQL>
....
BLOCK
...
```

Examples

The following was generated from higher level DSM Rule language...

```
ATTRIBUTES
version.major=2;  // Must be present to get version 2 evaluation algorithm.
INPUT
// input input1(int X);
input1: CREATE TABLE input1 (X int);

// input input2(int X);
input2: CREATE TABLE input2 (X int);

PERSISTENT
// persistent output2(int X);
output2[PREVIOUS=prev_output2]: CREATE TABLE output2 (X int,
TOC__INTERNAL timestamp);

// persistent output1(int X);
output1[PREVIOUS=prev_output1]: CREATE TABLE output1 (X int,
```

```
TOC__INTERNAL timestamp);

// Automatically inserted declaration(s):
// persistent prev_output2(int X);
prev_output2[AUTOMATIC,PRIMARY=output2]: CREATE TABLE
prev_output2 (X int, TOC__INTERNAL timestamp);

// Automatically inserted declaration(s):
// persistent prev_output1(int X);
prev_output1[AUTOMATIC,PRIMARY=output1]: CREATE TABLE
prev_output1 (X int, TOC__INTERNAL timestamp);

// Automatically inserted initialization(s):
INITIALIZE
DUMMY_INTERNAL_TABLE_[AUTOMATIC]: CREATE TABLE
DUMMY_INTERNAL_TABLE_ (X int);
DUMMY_INTERNAL_TABLE_[AUTOMATIC]: INSERT INTO
DUMMY_INTERNAL_TABLE_ VALUES (1);

CLAUSE
// output1(0) if  not input1(*);
output1: SELECT 0 as X, CURRENT_TIMESTAMP as TOC__INTERNAL FROM
DUMMY_INTERNAL_TABLE_
    WHERE (NOT EXISTS (SELECT * FROM input1));

// output2(x) if  prev output1(x);
output2: SELECT prev_output1.X as X, CURRENT_TIMESTAMP as
TOC__INTERNAL FROM prev_output1;
```

Rule Evaluation Algorithm (Version 2)

Upon acceptance of tuple state changes the DSMEngine class performs the following:

1. If this is the first rule evaluation, then execute any initialization statements in the order defined.
2. Make sure changes are only to Input or Transport tuples and that they are in the application name space and instance id of the engine, otherwise throw an exception.
3. Compute time since last rule evaluation.
4. Append new tuples (Input or Transport) to tables.
5. Copy all persistent tuples with PREVIOUS modifiers into the identified previous tables. Call this the previous persistent tuple state.
6. Clear the non-previous persistent tables declared as persistent (i.e. do not include those

created in the initialization section).

7. Iterate over each BLOCK (if no BLOCKs are defined, all rules are in block 0), performing the following
    1. Evaluate persistent append SELECT clauses, but
        1. if this has an MSEC modifier, then execute only once at the end of block evaluation and only if the value has expired.
        2. if this has a CHOOSE modifier, then execute only once at the end of the block evaluation.
    2. Add any new tuples to persistent tables.
8. Goto step 4, if we added any new tuples to the persistent tables.
9. Evaluate transport append SELECT clauses, including those whose MSEC value is less than the time since the last evaluation. Apply choose modifiers, if present. and send the results to their destinations.
10. Clear the Input tables.
11. Clear the transport tables.
12. Clear the previous persistent tables.

Formal Rule Syntax

<DSM> := <SECTION>*
<SECTION> := <ATTRIBUTE SECTION> | <TABLE SECTION> | <INIT_SECTION> | <CLAUSE SECTION> .

<ATTRIBUTE_SECTION> := 'ATTRIBUTES' <attribute list>
<attribute list> := <attribute value>*
<attribute value> := <identifier> '=' <any string> ';'

<TABLE SECTION> := <TABLE TYPE> <newline> <TABLE CREATION>+
<TABLE TYPE> := 'INPUT' | ‘PERSISTENT’ | ‘TRANSPORT’

<TABLE CREATION> := <Identifier> <DECL_MODIFIER_LIST>* ':' <DDL Statement> ‘;’
<DECL_MODIFIER_LIST> := '[' <DECL_MODIFER> (',' <DECL_MODIFIER>)* ']'
<DECL_MODIFER> := <PREVIOUS_MODIFIER> | <PRIMARY_MODIFIER> .
<PREVIOUS_MODIFIER> := 'PREVIOUS=' <identifier>
<PRIMARY_MODIFIER> := 'PRIMARY=' <identifier>

<INIT_SECTION> := 'INITIALIZE' <newline> <INIT_STATEMENT>* .
<INIT_STATEMENT> := (<SQL INSERT> | <DDL Statement>) ';'

<CLAUSE SECTION> := ‘CLAUSE’ <newline> <CLAUSE STATEMENT>*
<CLAUSE STATEMENT> := <RULE STATEMENT> | 'BLOCK'

<RULE STATEMENT> := <Identifier> (<MODIFIER_LIST>)? ‘:’ <SQL SELECT Statement> ‘;’

<MODIFIER_LIST> := '[' <MODIFIER> (',' <MODIFIER>)* ']'
<MODIFIER> := <PERIODIC_MODIFIER> | <CHOOSE_MODIFIER>.
<PERIODIC_MODIFIER> := 'MSEC=' <Integer>
<CHOOSE_MODIFIER>='CHOOSE='<Integer> [, CHOOSE_MIN=<Integer>] .

<Identifier> := [a-zA-z][a-zA-Z0-9_]*

<DDL Statement> := 'CREATE' <any string>

<SQL SELECT statement> := 'SELECT' <any string>

<SQL INSERT statement> := 'INSERT' <any string>

Caveats (feel free to modify the BNF above):

1. The SQL and DDL are NOT parsed by the rule parser and as such they are <any string>, however if there is an SQL parser error it will be identified at rule evaluation time.
2. A clause's <identifer> must refer to an identifier declared in one of the table sections.
3. Clause identifiers can not be that of an Input table.
4. MSEC modifiers can only appear on Persistent or Transport clauses.

Comments

There are no comments.