Операторы трансляции динамического SQL

В данном руководстве приводится описание синтаксиса и семантики операторов и выражений, реализованных для трансляции конструкций динамического SQL, задаваемых в коде PL/SQL. Ключевые слова приведены в верхнем регистре, однако они являются регистронезависимыми, как и все остальные ключевые слова PL/SQL.

Применение любых описанных ниже конструкций, выполняющих подстановку ? в блоках динамического PL/SQL кода, интерпретируется конвертером как подстановка на самый верхний уровень динамического блока. Соответственно знаки ? для таких конструкций не экранируются, а аргументы-подстановки добавляются в список аргументов самой старшей форматной строки.

Оглавление

1. Выражения для динамического SQL

Следующие 2 команды имеют семантику “интерпретировать заданные переменные как нетипизированные динамические поля в динамических выражениях”.

1.1. _DYN_EXPR – создать динамически генерируемое подвыражение

_DYN_EXPR(<выражение>) – транслировать выражение в динамически формируемую строку таким образом, чтобы его вычисление находилось в аргументе вызова makestr (т.е. в контексте процедурного языка, а не динамического запроса).

При подстановке в строку динамического запроса (форматную строку makestr) – учитывать тип самого выражения. Т.е. например, для выражения с типом VARCHAR2 – будет подставлен ’?’, для выражения с типом DATE – to_date(’?’), для выражения с типом INTEGER – просто ? и т.д. Кроме того, при необходимости, тип подстановки будет приведен соответственно контексту.

1.2. _DYN_FIELD – создать динамическое поле с возможностью типизации

_DYN_FIELD(<выражение>[, <динамический тип>]) – транслировать выражение в динамически формируемую строку таким образом, чтобы его вычисление находилось в аргументе makestr (т.е. в контексте процедурного языка, а не динамического запроса). В строку динамического запроса (форматную строку makestr) – подставлять символ ?, без кавычек. При задании необязательного аргумента “<динамический тип>”, тип подстановки будет приведен соответственно контексту через общий механизм приведения типов.

Аргумент “<тип>” имеет синтаксис типов PL/SQL. Допускаются конструкции “<foo>%TYPE” и “<foo>%ROWTYPE”.

```
<динамический тип> ::=
   <тип PL/SQL, включая %TYPE и %ROWTYPE>
 | _DYN_SUBQUERY  (<имя в кавычках>)
 ;
```

1.3. _DYN_SUBQUERY – сформировать пустую конструкцию, тип которой будет рассматриваться как тип ранее определенного подзапроса.

_DYN_SUBQUERY(<имя в кавычках>)

При трансляции подставить пустую строку и рассматривать ее тип, как тип запроса, ранее определенного в операторе _CONSTRUCT_EXPR с помощью опций global_cursor(<имя

в кавычках>) или save_cursor(<имя в кавычках>). Поиск определения запроса будет выполнен вначале среди операторов CONSTRUCT_EXPR (в области видимости объемлющей функции), заданных с опцией save_cursor, а затем – в глобальном контексте среди всех запросов, определенных с помощью global_cursor в операторах CONSTRUCT_EXPR.

Следует обратить внимание, что с помощью опции save_cursor – можно отобразить именованный идентификатор только на один запрос, а с помощью опции global_cursor – можно отобразить именованный идентификатор на несколько разных запросов (задав эту опцию с одним и тем же идентификатором для разных запросов). При этом, разные запросы, заданные с одним идентификатором – будут обрабатываться как если бы они находились в общем UNION.

1.4. __BOOL_TAIL__ – сформировать пустую конструкцию булевого типа

__BOOL_TAIL__ – подставить пустую строку и рассматривать ее тип, как BOOLEAN.

1.5. __NUM_TAIL__ – сформировать пустую конструкцию числового типа

__NUM_TAIL__ – подставить пустую строку и рассматривать ее тип, как NUMERIC.

1.6. __STR_TAIL__ – сформировать пустую конструкцию строкового типа

__STR_TAIL__ – подставить пустую строку и рассматривать ее тип, как VARCHAR2.

1.7. __DYN_LEN__ – подставить длину динамически сгенерированной строки

__DYN_LEN__(<выражение>) – выполнить трансляцию конструкции <выражение>, но подставить не саму оттранслированную конструкцию, а ее длину.

1.8. _DYN_TR_CALL_SIGNATURE – транслировать в рантайме полную сигнатуру вызова

_DYN_TR_CALL_SIGNATURE(<выражение>)

Обернуть <выражение> в вызов специально определенной хранимой процедуры, выполняющей динамическую трансляцию полной сигнатуры вызова (со скобками)[1]. Результат выполнения вызова данной хранимой процедуры-транслятора использовать в качестве подстановки.

Данное выражение может быть использовано в операторе вызова функции в качестве самого вызова.

[1] Реализация самой хранимой процедуры-транслятора еще не выполнена

1.9. _DYN_TR_FUN – транслировать в рантайме имя процедуры без скобок

_DYN_TR_FUN(<выражение для имени>, <возвращаемый тип>[, <список аргументов>])

Обернуть <выражение для имени> в вызов специально определенной хранимой процедуры, выполняющей динамическую трансляцию oracle-имени хранимой процедуры. В качестве имени может быть подставлено не только имя, но и литералы, числовые константы и NULL[2]. При трансляции рассматривать подставленное выражение, как имеющее тип <возвращаемый тип>.

В случае задания необязательного аргумента <список аргументов> – подставить список аргументов в скобках после подстановки знака ? для транслированного имени хранимой процедуры.

Аргумент <выражение для имени> может быть любым допустимым pl/sql выражением.

Аргумент <возвращаемый тип> может быть любым допустимым pl/sql типом.

Необязательный аргумент <список аргументов> может быть любым допустимым списком аргументов вызова функции.

1.10. _DYN_TABLE – создать динамически генерируемое имя таблицы с возможностью типизации

_DYN_TABLE(<строковое sql выражение>[, [<владелец>.]<имя таблицы или вью>]) – выражение, которое можно использовать там, где ожидается имя таблицы, для динамического формирования ее имени. Если задано <имя таблицы или вью> – то контекстный анализатор будет выполнять поиск полей ссылок на поля в этой таблице.

Конструкции, в которых можно использовать _DYN_TABLE:

- Предложение FROM;
- INSERT INTO;
- DELETE FROM;
- UPDATE;

Примеры использования:

```
OPEN C1 FOR
  select ID, ROWID
    from _dyn_table(V_SCH_NAME || '.' || V_TBL_NAME, M2_ALL.VALUTA)
   where _dyn_field(V_WHERE, BOOLEAN);

update _dyn_table(V_SCH_NAME || '.' || V_TBL_NAME)
   set _dyn_field(V_COL_NAME) = S1 where rowid = R;

insert into _dyn_table(CUR_TABLE.TABLE_NAME) (_dyn_field(FIELD_LIST))
     values(_dyn_field(VALUE_LIST));

delete from _dyn_table(CUR_SPEC_OBJ.TABLE_NAME)
      where _dyn_field(CHECK_WHERE);
```

[2] Реализация самой хранимой процедуры-транслятора еще не выполнена

1.11. _DYN_TAIL – создать динамическую вставку в произвольном месте выражения

_DYN_TAIL(<выражение>) – поместить кусок динамической строки в произвольное место выражения, не ограниченное синтаксисом операций. В этом случае, функциональность кастинга будет считать, что тип конструкции _DYN_TAIL совпадает с типом левого разобранного выражения. При этом, левым разобранным выражением может быть произвольное выражение PL/SQL[3].

1.12. _DYN_WHERE – сгенерировать секцию WHERE без ключевого слова WHERE

```
select ...
  from <список from>
_dyn_where(<содержимое динамического where>) ...
```

Пример:

```
 FOR CURSOR_ in select id from supplyPlan#V _dyn_where(INWHERE)
```

транслируется в

```
OPEN CURSOR_ FOR DIRECT makestr("SELECT id
FROM M2_ALL.\"SUPPLYPLAN#V\" ?;", INWHERE);
```

2. Операторы

2.1. _DECL_NAMESPACE – объявить пространство имен для динамического выражения

```
_DECL_NAMESPACE <имя пространства имен> from <предложение from>;

_DECL_NAMESPACE <имя пространства имен>
   declare
     <секция объявлений блока pl/sql>
   end;
```

Объявить пространство имен, в котором будет выполняться поиск определений ссылок внутри оператора _CONSTRUCT_EXPR. Поиск определений будет выполняться по общим правилам контекстного анализа SQL, если оператор имеет синтаксис секции FROM. Если же оператор имеет синтаксис секции DECLARE, то поиск определений будет выполняться по правилам контекстного анализа блоков PL/SQL.

[3] Если нужен больший контроль за тем, что будет разобрано и сгруппировано слева – необходимо отладить парсинг соответствующих конструкций _DYN_TAIL и настроить динамические приоритеты для нетерминалов.

2.2. _CONSTRUCT_EXPR – сформировать динамическое выражение

_CONSTRUCT_EXPR {<имя целевой переменной>[[, <идентификатор пространства имен>], <список опций>]} <выражение>

Сформировать makestr строку для динамического выражения SQL и записать ее в переменную, определяемую конструкцией "<имя целевой переменной>". Имя целевой переменной задается без точки[4]. Грамматика нетерминала "<выражение>" соответствует грамматике произвольного SQL-выражения, используемого внутри PL/SQL кода.

<идентификатор пространства имен> – имя без точки, определяющее ссылку на пространство имен, заданное конструкцией _DECL_NAMESPACE в одной из объемлющих областей видимости PL/SQL (верхняя область видимости ограничена самым старшим блоком). В этом пространстве имен будет выполняться поиск определений ссылок, используемых в динамическом выражении.

```
<список опций> ::=
    <опция>
  | <опция> , <список опций>
  ;

<опция> ::=
      concat
    | save_subquery(<имя в кавычках>)
    | global_cursor(<имя в кавычках>)
    ;
```

Описание опций

concat – выполнять конкатенацию целевой переменной со своим предыдущим значением.
Оператор "_CONSTRUCT_EXPR {<foo>, concat} <expr>;" будет перетранслирован в
<foo> := makestr(''? <транслированное expr>'', <foo>, <аргументы expr>);
Оператор "_CONSTRUCT_EXPR {<foo>} <expr>;" будет перетранслирован в
<foo> := makestr(''<транслированное expr>'', <динамические аргументы expr>);

save_subquery(<имя в кавычках>) – сохранить *в контексте объемлющей функции* ссылку с идентификатором <имя в кавычках> на запрос, определяемый нетерминалом <выражение>. Ссылку на этот запрос можно затем использовать в выражении _DYN_SUBQUERY(<имя в кавычках>) (оно формирует пустую часть строки, но заставляет конвертер считать, что там находится запрос <выражение>, и соответствующим образом приводить типы). Кроме того, сохраненную ссылку можно затем использовать для объявления типа выражения _DYN_FIELD, в качестве его второго аргумента.

global_cursor(<имя в кавычках>): – сохранить *в глобальном контексте* ссылку с идентификатором <имя в кавычках> на запрос, определяемый нетерминалом <выражение>. Глобальную ссылку на запрос можно использовать в качестве второго аргумента выражения _DYN_CURSOR_STR, заданного в любой другой функции модели. Применение данной опции полезно например в ситуации, когда запрос генерируется внутри какой-либо функции и возвращается в вызывающую функцию в виде строки, которая затем исполняется.

[4] При необходимости можно доработать, чтобы задавались произвольные идентификаторы

В глобальном контексте для одного идентификатора можно определить несколько ссылок на разные запросы. В этом случае, при кодогенерации данных запросов их типы будут приведены так, как если бы они находились в одном запросе и были бы объединены с помощью UNION.

2.3. _DECLTYPE_CURSOR – объявить тип запроса, текст которого хранится в строковой переменной

`_DECLTYPE_CURSOR <имя строковой переменной> for <запрос>`

Интерпретировать использование переменной `<имя строковой переменной>` в операциях ‘‘`OPEN <курсор> FOR <имя строковой переменной>`’’ как если бы на месте нетерминала `<имя строковой переменной>` стоял бы нетерминал `<запрос>`. Такое объявление необходимо для того, чтобы конвертер мог найти типы полей переменной `<курсор>` и корректно объявить для нее курсорную переменную в выводе кодогенерации.

Данный оператор целесообразно использовать только в тех случаях, когда в операторе `OPEN FOR` невозможно использовать выражения `_DYN_CURSOR_STR(...)` и `_DYN_FIELD(<foo>, _DYN_SUBQUERY(...))`. Например, в случае когда строка с запросом передается как входной аргумент функции, при этом сама функция ни откуда не вызывается, но из контекста можно понять тип выборки.

В качестве нетерминала `<имя строковой переменной>` должно быть либо имя без точки, либо имя псевдозаписи триггера, тоже без точки.

```
<имя строковой переменной> ::=
      <имя без точки>
  | : <имя псевдозаписи триггера без точки>
  ;
```

2.4. _DYN_PLSQL_BLOCK – создать блок динамического процедурного кода

`_DYN_PLSQL_BLOCK [INTO <имя переменной>] <блок PL/SQL>` – транслировать `<блок PL/SQL>` в блок динамического процедурного кода. Внутри такого динамического блока можно использовать любые конструкции PL/SQL[5], но кроме них также можно использовать выражения, определенные в секции “Выражения для динамических SQL и PL/SQL”.

Если задана секция `INTO <имя переменной>`, то результат будет возвращен в указанную переменную.

[5] Включая, вероятно, операторы для формирования динамического SQL. Однако, эта возможность пока не тестировалась.