40.12. 从Oracle PL/SQL进行移植

本节解释了PostgreSQL的PL/pgSQL 和Oracle的PL/SQL语言之间的差别， 希望能对那些从Oracle®向 PostgreSQL移植应用的人有所帮助。

PL/pgSQL与PL/SQL在许多方面都非常类似。 它是一种块结构的，祈使语气(命令性)的语言并且必须声明所有变量。 赋值、循环、条件等都很类似。在从PL/SQL向 PL/pgSQL移植的时候必须记住一些事情：

- 如果一个SQL命令中使用的名字是一个表中的列名，或者是一个函数中变量的引用， 那么PL/SQL会将它当作一个变量名。 这对应的是PL/pgSQL的`plpgsql.variable_conflict = use_column` 动作（不是默认动作）， 参考第 40.10.1 节中的描述。 首先，最好是避免这种模糊的方式，但如果不得不移植一个依赖于该动作的大量的代码， 那么设置`variable_conflict`是个不错的主意。
- 在PostgreSQL里，函数体必须写成字符串文本， 因此你需要使用美元符界定或者转义函数体里面的单引号(见第 40.11.1 节)。
- 应该用模式把函数组织成不同的组，而不是用包。
- 因为没有包，所以也没有包级别的变量。这一点有时候挺讨厌。 你可以在临时表里保存会话级别的状态。
- 带有`REVERSE`的整数的`FOR`循环的工作模式是不一样的： PL/SQL中是从第二个数向第一个数倒计， 而PL/pgSQL是从第一个数想第二个数倒计， 因此在移植时，需要交换循环边界。 不幸的是这种不兼容性是不太可能改变的（参阅第 40.6.3.5 节）。
- 遍历查询的`FOR`循环（而不是循环游标）同样有不同的工作模式： 必须已经声明了目标变量，在这一点上PL/SQL通常是隐式的声明。这样做的优点是，在退出循环后，仍然可以获得变量值。
- 在使用游标变量方面，存在一些记数法差异。

40.12.1. 移植样例

例 40-8演示了 如何从PL/SQL向PL/pgSQL移植一个简单的函数。

例 40-8. 从PL/SQL向PL/pgSQL移植一个简单的函数

下面是一个Oracle PL/SQL函数:

```
CREATE OR REPLACE FUNCTION cs_fmt_browser_version(v_name varchar,
                                                  v_version varchar)
RETURN varchar IS
BEGIN
    IF v_version IS NULL THEN
        RETURN v_name;
    END IF;
    RETURN v_name || '/' || v_version;
END;
/
show errors;
```

让我们读一遍这个函数然后看PL/pgSQL与之的不同:

- 在函数原型里的RETURN(不是函数体里的)关键字到了PostgreSQL里就 是RETURNS。还有， IS变成AS， 并且你还需要增加一个LANGUAGE子句， 因为PL/pgSQL并非唯一可用的函数语言。
- 在PostgreSQL里，函数体被认为是一个字符串文本， 所以你需要使用单引号或者美元符界定它， 这个包围符代替了Oracle 最后的那个/。
- 在PostgreSQL里没有show errors命令， 不需要这个命令是因为错误是自动报告的。

下面是这个函数移植到PostgreSQL之后的样子:

```
CREATE OR REPLACE FUNCTION cs_fmt_browser_version(v_name varchar,
                                                  v_version varchar)
RETURNS varchar AS $$
BEGIN
    IF v_version IS NULL THEN
```

```
        RETURN v_name;
    END IF;
    RETURN v_name || '/' || v_version;
END;
$$ LANGUAGE plpgsql;
```

例 40-9演示了如何移植一个创建另外一个函数的函数的方法， 以及演示了如何处理引号转义的问题。

例 40-9. 从PL/SQL向PL/pgSQL移植一个创建其它函数的函数

下面的过程从一个SELECT语句中抓取若干行， 然后为了提高效率，又用IF语句中的结果制作了一个巨大的函数。

这是Oracle版本:

```
CREATE OR REPLACE PROCEDURE cs_update_referrer_type_proc IS
    CURSOR referrer_keys IS
        SELECT * FROM cs_referrer_keys
        ORDER BY try_order;
    func_cmd VARCHAR(4000);
BEGIN
    func_cmd := 'CREATE OR REPLACE FUNCTION cs_find_referrer_type(v_host IN VARCHAR,
                 v_domain IN VARCHAR, v_url IN VARCHAR) RETURN VARCHAR IS BEGIN';

    FOR referrer_key IN referrer_keys LOOP
        func_cmd := func_cmd ||
          ' IF v_' || referrer_key.kind
          || ' LIKE ''' || referrer_key.key_string
          || ''' THEN RETURN ''' || referrer_key.referrer_type
          || '''; END IF;';
    END LOOP;

    func_cmd := func_cmd || ' RETURN NULL; END;';

    EXECUTE IMMEDIATE func_cmd;
END;
/
show errors;
```

下面是这个函数在PostgreSQL里面的样子:

```
CREATE OR REPLACE FUNCTION cs_update_referrer_type_proc() RETURNS void AS $func$
DECLARE
    referrer_keys CURSOR IS
```

```
        SELECT * FROM cs_referrer_keys
        ORDER BY try_order;
    func_body text;
    func_cmd text;
BEGIN
    func_body := 'BEGIN';

    FOR referrer_key IN referrer_keys LOOP
        func_body := func_body ||
          ' IF v_' || referrer_key.kind
          || ' LIKE ' || quote_literal(referrer_key.key_string)
          || ' THEN RETURN ' || quote_literal(referrer_key.referrer_type)
          || '; END IF;' ;
    END LOOP;

    func_body := func_body || ' RETURN NULL; END;';

    func_cmd :=
      'CREATE OR REPLACE FUNCTION cs_find_referrer_type(v_host varchar,
                                                        v_domain varchar,
                                                        v_url varchar)
        RETURNS varchar AS '
      || quote_literal(func_body)
      || ' LANGUAGE plpgsql;' ;

    EXECUTE func_cmd;
END;
$func$ LANGUAGE plpgsql;
```

请注意函数体是如何独立制作并且传递给quote_literal，对其中的单引号复制双份的。 需要这个技巧是因为无法使用美元符界定定义新函数： 没法保证referrer_key.key_string字段过来的字符串会解析成什么样子。 可以假设referrer_key.kind是只有host, domain或者 url，但是referrer_key.key_string可能是任何东西，特别是它可能包含美元符。 这个函数实际上是对原来 Oracle 版本的一个改进， 因为如果在referrer_key.key_string或者 referrer_key.referrer_type包含单引号的时候， 它不会生成有毛病的代码。

例 40-10演示了如何移植一个带有OUT参数和字符串处理的函数。 PostgreSQL里面没有内置instr函数，但是你可以用其它函数的组合来绕开它。 在第 40.12.3 节里有一个PL/pgSQL 的instr实现，你可以用它让你的移植变得更简单些。

例 40-10. 从PL/SQL向 PL/pgSQL移植一个字符串操作和OUT参数的过程

下面的Oracle PL/SQL过程用于分析一个URL并且返回若干个元素(主机、路径、命令)。

下面是Oracle版本:

```
CREATE OR REPLACE PROCEDURE cs_parse_url(
    v_url IN VARCHAR,
v_host OUT VARCHAR,  -- 这个变量是要传回的
    v_path OUT VARCHAR,  -- 这个也是
    v_query OUT VARCHAR) -- 还有这个
IS
    a_pos1 INTEGER;
    a_pos2 INTEGER;
BEGIN
    v_host := NULL;
    v_path := NULL;
    v_query := NULL;
    a_pos1 := instr(v_url, '//');

    IF a_pos1 = 0 THEN
        RETURN;
    END IF;
    a_pos2 := instr(v_url, '/', a_pos1 + 2);
    IF a_pos2 = 0 THEN
        v_host := substr(v_url, a_pos1 + 2);
        v_path := '/';
        RETURN;
    END IF;

    v_host := substr(v_url, a_pos1 + 2, a_pos2 - a_pos1 - 2);
    a_pos1 := instr(v_url, '?', a_pos2 + 1);

    IF a_pos1 = 0 THEN
        v_path := substr(v_url, a_pos2);
        RETURN;
    END IF;

    v_path := substr(v_url, a_pos2, a_pos1 - a_pos2);
    v_query := substr(v_url, a_pos1 + 1);
END;
/
show errors;
```

下面就是把这个过程翻译成PL/pgSQL可能的样子:

```
CREATE OR REPLACE FUNCTION cs_parse_url(
    v_url IN VARCHAR,
v_host OUT VARCHAR,  -- 这个将被传回
    v_path OUT VARCHAR,  -- 这个也传回
    v_query OUT VARCHAR) -- 还有这个
AS $$
DECLARE
```

```
    a_pos1 INTEGER;
    a_pos2 INTEGER;
BEGIN
    v_host := NULL;
    v_path := NULL;
    v_query := NULL;
    a_pos1 := instr(v_url, '//');

    IF a_pos1 = 0 THEN
        RETURN;
    END IF;
    a_pos2 := instr(v_url, '/', a_pos1 + 2);
    IF a_pos2 = 0 THEN
        v_host := substr(v_url, a_pos1 + 2);
        v_path := '/';
        RETURN;
    END IF;

    v_host := substr(v_url, a_pos1 + 2, a_pos2 - a_pos1 - 2);
    a_pos1 := instr(v_url, '?', a_pos2 + 1);

    IF a_pos1 = 0 THEN
        v_path := substr(v_url, a_pos2);
        RETURN;
    END IF;

    v_path := substr(v_url, a_pos2, a_pos1 - a_pos2);
    v_query := substr(v_url, a_pos1 + 1);
END;
$$ LANGUAGE plpgsql;
```

这个函数可以这么用:

```
SELECT * FROM cs_parse_url('http://foobar.com/query.cgi?baz');
```

例 40-11演示了如何一个使用各种Oracle专有特性的过程。

例 40-11. 从PL/SQL向PL/pgSQL移植一个过程

Oracle版本:

```
CREATE OR REPLACE PROCEDURE cs_create_job(v_job_id IN INTEGER) IS
    a_running_job_count INTEGER;
    PRAGMA AUTONOMOUS_TRANSACTION;(1)
```

```
BEGIN
    LOCK TABLE cs_jobs IN EXCLUSIVE MODE;(2)

    SELECT count(*) INTO a_running_job_count FROM cs_jobs WHERE end_stamp IS NULL;

    IF a_running_job_count > 0 THEN
        COMMIT; -- free lock(3)
        raise_application_error(-20000,
                'Unable to create a new job: a job is currently running.');
    END IF;

    DELETE FROM cs_active_job;
    INSERT INTO cs_active_job(job_id) VALUES (v_job_id);

    BEGIN
        INSERT INTO cs_jobs (job_id, start_stamp) VALUES (v_job_id, sysdate);
    EXCEPTION
        WHEN dup_val_on_index THEN NULL; -- don't worry if it already exists
    END;
    COMMIT;
END;
/
show errors
```

像这样的过程可以很容易用返回void的函数移植到PostgreSQL里。 对这个过程特别感兴趣是因为它可以教一些东西:

(1)
在PostgreSQL里没有PRAGMA语句。

(2)
如果你在PL/pgSQL里做一个LOCK TABLE, 那么这个锁在调用该命令的事务完成之前将不会释放。

(3)
你不能在PL/pgSQL函数里发出COMMIT。 函数是在外层的事务里运行的, 因此COMMIT蕴涵着结束函数的执行。 不过, 在这个特殊场合下, 这是不必要的了, 因为LOCK TABLE获取的锁将在抛出错误的时候释放。

下面是把这个过程移植到PL/pgSQL里的一种方法:

```
CREATE OR REPLACE FUNCTION cs_create_job(v_job_id integer) RETURNS void AS $$
DECLARE
    a_running_job_count integer;
BEGIN
    LOCK TABLE cs_jobs IN EXCLUSIVE MODE;

    SELECT count(*) INTO a_running_job_count FROM cs_jobs WHERE end_stamp IS NULL;

    IF a_running_job_count > 0 THEN
```

```
        RAISE EXCEPTION 'Unable to create a new job: a job is currently running';(1)
    END IF;

    DELETE FROM cs_active_job;
    INSERT INTO cs_active_job(job_id) VALUES (v_job_id);

    BEGIN
        INSERT INTO cs_jobs (job_id, start_stamp) VALUES (v_job_id, now());
    EXCEPTION
        WHEN unique_violation THEN (2)
            -- don't worry if it already exists
    END;
END;
$$ LANGUAGE plpgsql;
```

(1)

RAISE的语法和Oracle的类似语句差别相当明显。 尽管RAISE*exception_name*运行的基本情况相似。

(2)

PL/pgSQL里支持的异常的名字和Oracle的不同。 内置的异常名要大的多(参阅附录 A)。 目前还不能声明用户定义的异常名。

整个过程和Oracle的等效的主要的功能型差别是， 在cs_jobs上持有的排他锁将保持到调用的事务结束。 同样，如果调用者后来退出(比如说因为错误)，这个过程的效果将被回滚掉。

40.12.2. 其它注意事项

本节解释几个从Oracle PL/SQL函数向PostgreSQL 移植的几个其它方面的事情。

40.12.2.1. 异常后的隐含回滚

在PL/pgSQL里，如果一个异常被EXCEPTION子句捕获， 那么所有自这个块的BEGIN以来的数据库改变都会被自动回滚。 也就是说，这个行为等于你在Oracle里的：

```
BEGIN
    SAVEPOINT s1;
    ... code here ...
EXCEPTION
    WHEN ... THEN
        ROLLBACK TO s1;
        ... code here ...
    WHEN ... THEN
```

```
        ROLLBACK TO s1;
        ... code here ...
END;
```

如果你在翻译使用SAVEPOINT和ROLLBACK TO的Oracle过程， 那么你的活儿很好干：只要省略SAVEPOINT和 ROLLBACK TO即可。 如果你要翻译的过程使用了不同的SAVEPOINT和ROLLBACK TO，那么就需要想想了。

40.12.2.2. EXECUTE

PL/pgSQL版本的EXECUTE类似PL/SQL运转， 不过你必须记住要像第 40.5.4 节里描述的那样 用quote_literal和quote_ident。 如果你不用这些函数，那么像EXECUTE 'SELECT * FROM $1';这样的构造是不会运转的。

40.12.2.3. 优化PL/pgSQL函数

PostgreSQL给你两个创建函数的修饰词用来优化执行："volatility" (易变的，在给出的参数相同时， 函数总是返回相同结果)和"strictness" (严格的，如果任何参数是NULL，那么函数返回NULL)。 参考CREATE FUNCTION的手册获取细节。

如果要使用这些优化属性，那么你的CREATE FUNCTION语句可能看起来像这样：

```
CREATE FUNCTION foo(...) RETURNS integer AS $$
...
$$ LANGUAGE plpgsql STRICT IMMUTABLE;
```

40.12.3. 附录

本节包含Oracle兼容的instr函数，你可以用它简化你的移植过程。

```
--
-- 模拟 Oracle 概念的 instr 函数
-- 语法: instr(string1, string2, [n], [m])  这里的 [] 表示可选参数
--
-- 从 string1 的第 n 个字符开始寻找 string2 的第 m 个出现。
```

```
-- 如果 n 是负数，则从后向前着。如果没有传递 m ，假定为 1(从第一个字符开始找)。
--

CREATE FUNCTION instr(varchar, varchar) RETURNS integer AS $$
DECLARE
    pos integer;
BEGIN
    pos:= instr($1, $2, 1);
    RETURN pos;
END;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;


CREATE FUNCTION instr(string varchar, string_to_search varchar, beg_index integer)
RETURNS integer AS $$
DECLARE
    pos integer NOT NULL DEFAULT 0;
    temp_str varchar;
    beg integer;
    length integer;
    ss_length integer;
BEGIN
    IF beg_index > 0 THEN
        temp_str := substring(string FROM beg_index);
        pos := position(string_to_search IN temp_str);

        IF pos = 0 THEN
            RETURN 0;
        ELSE
            RETURN pos + beg_index - 1;
        END IF;
    ELSIF beg_index < 0 THEN
        ss_length := char_length(string_to_search);
        length := char_length(string);
        beg := length + beg_index - ss_length + 2;

        WHILE beg > 0 LOOP
            temp_str := substring(string FROM beg FOR ss_length);
            pos := position(string_to_search IN temp_str);

            IF pos > 0 THEN
                RETURN beg;
            END IF;

            beg := beg - 1;
        END LOOP;

        RETURN 0;
    ELSE
        RETURN 0;
    END IF;
END;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;
```

```
CREATE FUNCTION instr(string varchar, string_to_search varchar,
                      beg_index integer, occur_index integer)
RETURNS integer AS $$
DECLARE
    pos integer NOT NULL DEFAULT 0;
    occur_number integer NOT NULL DEFAULT 0;
    temp_str varchar;
    beg integer;
    i integer;
    length integer;
    ss_length integer;
BEGIN
    IF beg_index > 0 THEN
        beg := beg_index;
        temp_str := substring(string FROM beg_index);

        FOR i IN 1..occur_index LOOP
            pos := position(string_to_search IN temp_str);

            IF i = 1 THEN
                beg := beg + pos - 1;
            ELSE
                beg := beg + pos;
            END IF;

            temp_str := substring(string FROM beg + 1);
        END LOOP;

        IF pos = 0 THEN
            RETURN 0;
        ELSE
            RETURN beg;
        END IF;
    ELSIF beg_index < 0 THEN
        ss_length := char_length(string_to_search);
        length := char_length(string);
        beg := length + beg_index - ss_length + 2;

        WHILE beg > 0 LOOP
            temp_str := substring(string FROM beg FOR ss_length);
            pos := position(string_to_search IN temp_str);

            IF pos > 0 THEN
                occur_number := occur_number + 1;

                IF occur_number = occur_index THEN
                    RETURN beg;
                END IF;
            END IF;

            beg := beg - 1;
        END LOOP;

        RETURN 0;
```

```
    ELSE
        RETURN 0;
    END IF;
END;
$$ LANGUAGE plpgsql STRICT IMMUTABLE;
```

上一页 开发PL/pgSQL的一些提示
起始页 上一级
下一页 PL/Tcl - Tcl 过程语言