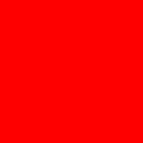


ORACLE

Oracle 数据库 11g 中的 PL/SQL 增强功能

Bryn Llewellyn

Oracle 总部数据库服务器技术产品经理



以下内容旨在概述我们产品总的发展方向。此信息仅供参考，不会纳入到任何合同中。此信息不承诺提供任何资料、代码或功能，并且不应该作为制定购买决策的依据。

此处所述有关 **Oracle** 产品的任何特性或功能的开发、发布以及相应的日程安排均由 **Oracle** 自行决定。

Oracle 数据库 11g 中的 PL/SQL 增强功能

- Oracle 数据库的每个新的主版本都在以下方面提供 PL/SQL 增强功能
 - 透明且“按钮控制的”性能改进
 - 用于在程序中获得更高性能的新语言特性
 - 您早前无法实现或只能靠繁琐的变通方法实现的新语言特性
 - 用 PL/SQL 提高编程可用性的新语言特性

透明的性能:

细粒度的依赖项跟踪



挑战

```
create table t(a number)
/  
create view v as select a from t  
/  
alter table t add(Unheard_Of number)  
/  
select status from User_Objects  
  where Object_Name = 'V'  
/
```

- 视图 `v` 在 10.2 中无效，因为它的依赖父类已更改（就整个对象的层面而言）

挑战

```
create package Pkg is
  procedure p1;
end Pkg;
/
create procedure p is begin Pkg.p1(); end;
/
create or replace package Pkg is
  procedure p1;
  procedure Unheard_Of;
end Pkg;
/
select status from User_Objects
  where Object_Name = 'P'
/
```

- 过程 *p* 也是一样

细粒度的依赖项跟踪

- 在 11.1 中，我们在 *单元内元素级别* 级别跟踪依赖项
 - 因此我们知道这些更改没有结果
- 我将其归类为透明的性能改进
 - 它确实是透明的！
 - 无用的重编译确实消耗 CPU
- 别忘了，“4068”系列错误有一个不同的原因：*至少重编译一个已由第二个并发会话实例化的会话状态程序包主体*

性能“按钮”：

真正的本地编译



挑战

- 通过 10.2，编译为本地 DLL 的 PL/SQL 比 PVM 针对解释编译的 PL/SQL 快得多
- Oracle 将 PL/SQL 源翻译为 C 代码并将最后一步留给第三方 C 编译器
- 完成然而，一些客户不愿意使用附带的 C 编译器！
- 此外，其他一些客户在已经付费取得 Oracle 数据库使用许可之后，不愿意付费取得 C 编译器使用许可！

真正的本地编译

- 在 11.1 中，Oracle 将 PL/SQL 源 *直接* 翻译为当前硬件的 DLL
- 此外，Oracle 进行链接和加载，以便不再需要文件系统目录
- 因此 PL/SQL 本地编译将以随取随用方式工作 — 不会破坏您的习惯
- 只保留一个参数：on/off 开关 — *PLSQL_Code_Type*

真正的本地编译

- 好处在于它更快！
 - 真正的本地编译要比 C 编译快两倍
 - 根据 Whetstone 基准计算，真正的本地编译比 C 编译快 2.5 倍以上
 - 经过设计的测试已经提高了 20%
- 新的 PL/SQL 数据类型 *simple_integer* 具有与硬件的整数操作完全匹配的语法
 - 有非空约束
 - 换行而非溢出

因此它比 *pls_integer* 快

性能“按钮”：

单元内内联



挑战

- 使用帮助子程序（如 Steven Feuerstein 介绍的）提高可理解性
- 通常，它们比较短
- 程序员有时受困于两难境地：

可读性/正确性/可维护性
对比
性能

挑战

```
procedure p(Input_String varchar2) is
    ...
    function Found_Another_Word(w out varchar2)
        return boolean is ...;
    function Is_Article(w in varchar2)
        return boolean is ...;
begin
    while Found_Another_Word(Word) loop
        if Is_Article(Word) then
            Article_Count := Article_Count + 1;
        end if;
    end loop;
end p;
```

挑战

```
function Found_Another_Word(w out varchar2)
  return boolean is
begin
  End_Pos := Instr(v, Space, Start_Pos);
  if End_Pos > 0 then
    w := Substr(v, Start_Pos, (End_Pos-
  Start_Pos));
    Start_Pos := End_Pos + 1;
    while Substr(v, Start_Pos, 1) = Space loop
      Start_Pos := Start_Pos + 1;
    end loop;
    return true;
  else
    return false;
  end if;
end Found_Another_Word;
```

单元内内联

```
alter procedure p compile
  PLSQL_Optimize_Level = 2
  reuse settings
/
begin p(:Big_Doc); end;
/
alter procedure p compile
  PLSQL_Optimize_Level = 3 -- New in 11.1
  reuse settings
/
begin p(:Big_Doc); end;
/
```

- 级别 2 约为 700 毫秒
- 级别 3 约为 400 毫秒

单元内内联

- 您获得的结果可能有所不同！
- 使用电子商务套件的一个测试
 - “弹性域”
 - 纯 PL/SQL 数据整理
 - 具有大量帮助子程序的大型程序包
 - 速度提高了 20%
- 使用 PL/SQL 小组的基准测试套件
 - 一些测试没有内联机会
 - 平均速度提高了 10%

性能语言特性:

SQL 和 PL/SQL 结果缓存



挑战

- 找到总人口中按州分组的最大平均收入值 — 或某个类似的量度
- 大量行合并为几行或一行
- 该数据更改得十分慢（例如，每小时），但该查询经常重复（例如，每秒）

挑战

```
function f1 return t1%rowtype is
  r t1%rowtype;
begin
  select a, m
  into r.a, r.b
  from (
    select a, sb m from (
      select a, Sum(b) sb from t1
      group by a)
    order by m desc)
  where Rownum = 1;
  return r;
end f1;
```

- 对于每个新调用约为 1,000 毫秒

SQL 查询结果缓存

```
function f1 return t1%rowtype is
  r t1%rowtype;
begin
  select /*+ result_cache */ a, m
  into r.a, r.b
  from (
    select a, sb m from (
      select a, Sum(b) sb from t1
      group by a)
    order by m desc)
  where Rownum = 1;
  return r;
end f1;
```

- 对于每个新调用约为 0 毫秒

挑战

- 计算更复杂的派生量度 — 例如，总人口中按州分组的中上等收入和中下等收入的比率
- 现在我们需要一个 **PL/SQL** 函数
- 同样，该数据更改得十分慢（例如，每小时），但该查询经常重复（例如，每秒）

挑战

```
function f2 return t1%rowtype
is
    ...
begin
    select a, m into r1.a, r1.b from ...;

    select a, m into r2.a, r2.b from ...;

    r.a := r1.a + r2.a;
    r.b := r1.b + r2.b;
    return r;
end f2;
```

- 对于每个新调用约为 2,000 毫秒

PL/SQL 函数结果缓存

```
function f2 return t1%rowtype
  result_cache relies_on(t1, t2)
is
  ...
begin
  select a, m into r1.a, r1.b from ...;

  select a, m into r2.a, r2.b from ...;

  r.a := r1.a + r2.a;
  r.b := r1.b + r2.b;
  return r;
end f2;
```

- 对于每个新调用约为 0 毫秒

SQL 和 PL/SQL 结果缓存

- 它们都是跨会话的、RAC 可互操作的
- 它们构建于同一基础架构之上
 - 相同的 *Result_Cache_Size*,... 初始化参数
 - 相同的 *DBMS_Result_Cache* 管理程序包
 - 相同的 *v\$Result_Cache_** 性能视图

性能语言特性:

复合触发器



挑战

- 每当员工薪水改变时向单独的审计表中添加一行
- 通常，大量员工行由单个更新更改
- 寻找针对审计行使用批量插入的方式
- 通过 10.2，程序员已经使用了“辅助程序包范例”
 - 在“在语句之前”中初始化程序包全局变量；在“在每一行之前”中进行批处理和刷新；在“在语句之后”中进行最终刷新

复合触发器

- 复合触发器允许在单个触发器中针对表的每个 DML 时间点实现操作
- 可以为这些部分定义全局变量
 - 可以“在语句之前”精心修改这些声明
 - 可在“在语句之前”部分提供显式初始化代码
 - 可在“在语句之后”部分提供最终化代码
 - 这些全局变量在激发 SQL 完成时被破坏

复合触发器

```
create trigger My_Compound_Trigger
  for update of Salary on Employees
compound trigger
  -- These variables have firing-statement duration
  Threshold constant pls_integer := 200;

  before statement is
  begin
    ...
  end before statement;

  -- And/or "after each row"
  before each row is
  begin
    null;
  end before each row;

  after statement is
  begin
    null;
  end after statement;
end My_Compound_Trigger;
/
```

复合触发器

```
create trigger My_Compound_Trg
  for update of Salary on Employees
  compound trigger

  Threshold    constant pls_integer := 200;
  type Emps_t  is table of Employee_Salaries%rowtype
    index by pls_integer;
  Emps         Emps_t;
  Idx         pls_integer := 0;

  procedure Flush_Array is
  begin
    forall j in 1..Emps.Count()
      insert into Employee_Salaries values Emps(j);
    Emps.Delete();
    Idx := 0;
  end Flush_Array;

  ...

end My_Compound_Trg;
/
```

复合触发器

```
create trigger My_Compound_Trigger
  for update of Salary on Employees
  compound trigger

  ...

  after each row is
  begin
    Idx := Idx + 1;
    Emps(Idc).Employee_Id      := :New.Employee_Id;
    Emps(Idc).Salary           := :New.Salary;
    Emps(Idc).Effective_Date   := Sysdate();

    if Idc >= Threshold then
      Flush_Array();
    end if;
  end after each row;

  ...

end My_Compound_Trigger;
/
```

复合触发器

```
create trigger My_Compound_Trg
  for update of Salary on Employees
  compound trigger
```

...

```
  after statement is
  begin
    Flush_Array();
  end after statement;

end My_Compound_Trg;
/
```

功能:

动态 SQL 函数完整性



挑战

- 您要生成一个源超过 32K 字符的较大 PL/SQL 单元
- 您只希望通过 PL/SQL 子程序公开数据库；对于具有非绑定结果集的查询，使用 *Ref Cursor*。现在要求变了，直到运行时您才能知道 *where* 字句 — 从而知道绑定数
- 直到运行时才能知道绑定数，但选择列表是固定的；您想使用本地动态 SQL 的批提取

顺便说一句，方法 4 即是如此

- 方法 4 意味着您直到运行时才能知道定义数（即，选择列表）或绑定数
- 因此，需要发现选择列表列的数量和数据类型
- 经过大量争论，我们认为通过一个过程 API 比通过语言语法能更好地表达方法 4 所需步骤的本质
- *DBMS_Sql* 就是一个好例子！

动态 SQL 函数完整性

- *execute immediate* 获取一个 clob
- 相应地, *DBMS_Sql.Parse()* 获取一个 clob
- 可将 Ref Cursor 转换为 *DBMS_Sql* 游标, 反之亦然
- *DBMS_Sql* 支持 ADT

动态 SQL 函数完整性

```
...
Cur_Num number := DBMS_Sql.Open_Cursor();
rc Sys_Refcursor;

cursor e is select Employee_ID, First_Name, Last_Name
              from Employees;
type Emps_t is table of e%rowtype;
Emps Emps_t;
begin
  DBMS_Sql.Parse(
    c=>Cur_Num, Language_Flag=>DBMS_Sql.Native, Statement=>
      'select Employee_ID, First_Name, Last_Name
       from Employees
       where Department_ID = :d and Salary > :s and ...');

  DBMS_Sql.Bind_Variable(Cur_Num, ':d', Department_ID);
  DBMS_Sql.Bind_Variable(Cur_Num, ':s', Salary);
  ...
  Dummy := DBMS_Sql.Execute(Cur_Num);
  -- Switch to ref cursor and native dynamic SQL
  rc := DBMS_Sql.To_Refcursor(Cur_Num);

  fetch rc bulk collect into Emps;
  close rc;
  ...
```

挑战

- *DBMS_Sql* 游标只是一个普通数字
- *Is_Open()* 函数使恶意的程序员（假设其对另一个用户的定义器的权限单元有 **Execute** 权限）能够使用 *DBMS_Sql* 安全地处理指定的 **SQL**，以便猜测并截获开放的游标以及重绑定和重执行该 **SQL** — 利用安全检查只在分析时进行的事实。

增强的 **DBMS_Sql** 安全性

- 如果 *Is_Open()* 通过无效游标调用，则调用任何 **DBMS_Sql** 子程序的后续尝试都将在该会话的剩余生存期内引起错误。
- *Open_Cursor()* 的一个新过载有一个 *Security_Level* 参数
 - 0: 不检查
 - 1: 进行绑定和执行的用户必须与进行分析的用户一致
 - 2: 所有操作的用户必须一致
- 新的具有下划线的参数 *_dbms_sql_security_level* 在系统范围应用这些选项

功能:

细粒度的访问控制
针对 ***Utl_TCP*** 及其同类



挑战

- Oracle 数据库为 PL/SQL 子程序提供打包的 API，以便使用 TCP/IP 和其他内置在其上的其他协议（SMTP 和 HTTP）访问计算机（由主机和端口指定）。
- *Utl_TCP*、*Utl_SMTP*、*Utl_HTTP*.....
- 如果您有程序包的 *Execute* 权限，则可以访问任何主机端口
- 至于 *Execute* 是通过 *public* 操作还是直接得到授权并不重要

细粒度的访问控制

针对 *Utl_TCP* 及其同类

- 访问控制列表 (ACL) 指定一组用户和角色
- 可将一个 ACL 分配到一个主机和端口范围
- 这些 ACL 由 XDB 管理

功能:

SQL 和 PL/SQL 中的 正则表达式增强功能



挑战

```
p := '(?\\d{3}\\)? ?\\d{3}[-.]\\d{4}';
```

```
Str :=
```

```
'bla bla (123)345-7890 bla bla  
(345)678-9012 bla bla (567)890-1234 bla bla';
```

```
Match_Found := Regexp_Like(Str, p);
```

- 好的，至少有一个匹配。但究竟有多少呢？
- 可以按照 *Str* 逐步发现每个成功的匹配，逐个叠加来自自己计算！

SQL 和 PL/SQL 中的正则表达式增强功能

```
No_Of_Matches := Regexp_Count(Str, p);
```

- *Regexp_Instr* 和 *Regexp_Substr* 现在有一个可选的 *Subexpr* 参数，使您能够针对所评估的正则表达式的特定子字符串。

功能:

对“**super**”的支持



挑战

- *Employee* 父类型有一个计算一般基础信息的可重写的成员函数 *Monthly_Pay()*
- *Salesperson* 子类型专门使用 *Monthly_Pay()* 了解诸如实际销售佣金的概况
- 该自然实现使 *SalespersonMonthly_Pay()* 调用 *Employee.Monthly_Pay()*
- 您知道吗？在 10.2 中，不用繁琐的替代方法是无法作到这一点的

对“super”的支持

- 该 OO 范例指定解决方案
- ANSI 对其进行描述
- 通俗地讲，它就是对“super”的支持
- 11.1 引入了该支持

- 如果您不知道这是什么，那么您不需要它！

功能:

创建一个禁用的触发器
指定触发器过滤顺序
新的 **PLW-06009** 警告



创建一个禁用的触发器

```
create or replace trigger Trg
  before insert on My_Table for each row
  disable
begin
  :New.ID := My_Seq.Nextval;
end;
/
```

- 如果您创建一个主体带有 PL/SQL 编译错误的触发器，该表的 DML 将产生错误“ORA-04098:trigger 'TRG' is invalid and failed re-validation”
- 因此以禁用状态创建它，并仅当您知道它进行了无错编译的情况下进行启用要更加安全

指定触发器过滤顺序

```
create or replace trigger Trg_2
  before insert on My_Table for each row
  follows Trg_1
begin
  ...
end;
/
```

- 在 10.2 中，您可能认为自己知道过滤顺序（通过实验观察），但是您却不能依赖它

挑战

```
create procedure p(i in number) is
begin
  insert into My_Table(n) values(i);
exception
  when others then null;
end p;
/
```

- 另外的人编写“*when others then null*”，因为他们只预期了 *Dup_Val_On_Index* 异常 — 但（令人惊异地）想“确保”该程序不会失败。
- 现在，您已经继承了该代码，而且您认为可以避免这些异常

新的 PLW-06009 警告

```
alter procedure p compile
  PLSQL_Warnings = 'enable:all'
  reuse settings
/
```

- 这将产生以下警告:

```
PLW-06009: procedure "P" OTHERS handler
  does not end in RAISE or
  RAISE_APPLICATION_ERROR
```

- 其他新警告, 例如

```
PLW-06006: uncalled procedure "F" is removed.
```

语言可用性:

PL/SQL 表达式中的序列



挑战

```
create or replace trigger Trg
  before insert on My_Table for each row
declare
  s number;
begin
  -- Annoying locution
  select My_Seq.Nextval into s from Dual;
  :New.PK := n;
end;
/
```

- 此处也有一个性能问题

PL/SQL 表达式中的序列

```
create or replace trigger Trg
  before insert on My_Table for each row
```

```
begin
```

```
    :New.ID := My_Seq.Nextval;
end;
/
```

- 值得庆幸的是，对于任何简单的“select... from Dual”，该性能问题得到了解决

语言可用性:

continue 语句



挑战

```
<<Outer>>for i in 1..10 loop  
  
    ...  
  
    <<Inner>>for j in 1..Data.Count() loop  
        if not Data(j).Uninteresting then  
            ...  
        end if;  
    end loop;  
end loop;
```

- 该逻辑比较繁琐，而且遵循从后向前原则.....
- 尤其是当您想（根据您检测的条件）启动一个封装循环的下一迭代时

The *continue* 语句

```
<<Outer>>for i in 1..10 loop  
  
    ...  
  
    <<Inner>>for j in 1..Data.Count() loop  
        continue Outer when Data(j).Uninteresting;  
        ...  
  
    end loop;  
end loop;
```

- 很多算法是使用 *continue* 语句以伪代码描述的

语言可用性:

SQL 中的命名批注和混合批注



挑战

```
create function f(  
  p1 in number default 1,  
  ...,  
  p5 in number default 5) return number  
is  
  v number  
begin  
  ...  
  return v;  
end f;  
/  
select f(p4 => 10) from Dual  
/  
ORA-00907: missing right parenthesis
```

SQL 中的命名批注和混合批注

```
select f(p4 => 10) from Dual
/
  F(P4=>10)
-----
          21
```

工具支持



PL/Scope

```
alter session set  
  PLScope_Settings = 'identifiers:all'
```

```
create or replace...
```

```
select      Name,  
            Type,  
            Usage,  
            Usage_ID,  
            Usage_Context_ID,  
            Signature  
  
from        User_Identifiers  
  
where       Object_Name = ...  
            and Object_Type = ...
```

PL/SQL 分层 Profiler

- 报告动态执行配置文件
- 按子程序调用组织，例如，
 - 对子程序的调用数
 - 花在子程序自身上的时间
 - 花在子程序的子树上的时间
 - 详细的父-子信息
- 对 SQL 和 PL/SQL 分别计算
- 生成超链接的 HTML 报告

总结



总结

- 性能
 - 细粒度的依赖项跟踪
 - 真正的 PL/SQL 本地编译
 - 单元内内联
 - SQL 和 PL/SQL 结果缓存
 - 复合触发器
- 请注意，只需少量工作就可获益于这些特性

总结

- 功能
 - 动态 SQL 函数完整性
 - *DBMS_Sql* 安全性
 - 针对 *Utl_TCP* 等的细粒度访问
 - SQL 和 PL/SQL 中的 *Regexp_Count()* 等
 - 对“super”的支持
 - 将表 t 修改为只读
 - 创建一个禁用的触发器；指定触发器过滤顺序
 - “when others then null”编译时警告

总结

- 可用性
 - PL/SQL 表达式中的序列
 - *continue* 语句
 - SQL 中的命名批注和混合批注

- 工具
 - PL/Scope
 - PL/SQL 分层 Profiler

问&答

