



ORACLE®

Oracle Database 11g: Новые возможности в PL/SQL

Игорь Мельников
Консультант по базам данных

План

- Увеличение быстродействия
- Новые синтаксические конструкции
- Новые возможности в динамическом SQL
- Новое в триггерах
- Технология Result Cache
- Новые инструменты для разработчиков
- Заключение





Увеличение быстродействия

Отслеживание изменений объектов

До PL/SQL 11g

```
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'
/
STATUS
-----
INVALID
```

- Представление V в недействительном состоянии, потому-что структура таблицы изменилась. Отслеживание зависимостей на уровне всего объекта!

Отслеживание изменений объектов

До PL/SQL 11g

```
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'
/
```

STATUS

INVALID

- Тоже самое для процедуры P

Fine Grained Dependency Tracking

В PL/SQL 11g

- В 11.1 зависимости отслеживаются на уровне элементов объектов
 - СУБД “знает” какие изменения приведут к изменению объекта, а какие нет
- Преимущества:
 - Плавный “накат” патчей на приложение
 - Нет необходимости в ненужных перекомпиляциях
- *Внимание: Ошибка “ORA-4068” остается; возникает по другой причине: перекомпиляция тела пакета который использовался в другой сессии*

PL/SQL Native Compilation

До PL/SQL 11g

- Начиная с 9.2, можно компилировать PL/SQL-процедуры в исполняемый код платформы (DLL/SO/SL) вместо компиляции в MC-код PL/SQL VM
- СУБД переводит исходный код PL/SQL в C-код, а затем с помощью C-компилятора получает выполняемую библиотеку
- Нужно было устанавливать на сервер C-компилятор и настраивать СУБД
- На многих платформах C-компилятор нужно дополнительно лицензировать!

Real Native Compilation

В PL/SQL 11g

- В 11.1, СУБД Oracle при компиляции PL/SQL напрямую может генерировать выполняемый код DLL программно-аппаратной платформы
- С-компилятор не нужен
- Возможность встроена в СУБД, не нужно никакой дополнительной настройки
- Только один параметр: включить/выключить, *PLSQL_CODE_TYPE*
 - *INTERPRETED*
 - *NATIVE*

Преимущества Real Native Compilation

В PL/SQL 11g

- Увеличение быстродействия:
 - Компиляция в 2 раза быстрее, чем с переводом на C-исходный код
 - Тест Whetstone показывает увеличение быстродействия в 2.5 раза по сравнению с вариантом с использованием C-компилятора (нет ненужного runtime-library и оптимизация)
 - На тестах – выполнение в 20 раз быстрее, чем при использовании режима интерпретации PVM

Новый тип данных SIMPLE_INTEGER

В 11g: новый тип для целых чисел

- Является подтипом типа PLS_INTEGER
- Но в отличие от PLS_INTEGER, для переменных типа SIMPLE_INTEGER НЕ генерируется код проверки на переполнение (overflow checking), и код проверки на NULL
- Повышение быстродействия в среднем 20-30% по сравнению с PLS_INTEGER
- Области применения: операции с целочисленной арифметикой, где заранее известен диапазон (напр: счетчики циклов)

НОВЫЙ тип данных SIMPLE_INTEGER

В 11g: новый тип для целых чисел

- Переменные типа SIMPLE_INTEGER имеют неявное ограничение NOT NULL !

```
declare
  xCount simple_integer;
begin
  null;
end;
/
  xCount simple_integer;
  *
```

ERROR at line 2:

ORA-06550: line 2, column 10:

PLS-00218: a variable declared NOT NULL must have an
initialization assignment

Последовательности в PL/SQL

До PL/SQL 11g

```
create or replace trigger Trg
  before insert on My_Table for each row
declare
  s number;
begin
  -- Автоматическая генерация первичного ключа
  select My_Seq.Nextval into s from dual;
  :New.Id := s;
end;
```

Последовательности в PL/SQL

В PL/SQL 11g

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

- Обращение к последовательности прямо в выражении - проще и короче код
- Повышение производительности (не нужно обрабатывать курсор), используется прямой и более быстрый механизм доступа – НЕ препроцессинг в *select from dual* !
- Переключение контекста, как таковое, все равно нужно: последовательность лежит в SGA

Inline-подстановка в PL/SQL

PL/SQL 11g: Подстановка тела функции вместо вызова

```
begin
  PRAGMA INLINE (my_func, 'YES'); -- включаем подстановку

  for f in (select * from employees)
  loop
    x := my_func (f.Name, f.amount) + 17; -- не вызов, а тело
                                           -- функции!
  end loop;

  PRAGMA INLINE (my_func, 'NO'); -- выключаем подстановку
  ...
end;
```

- Увеличение скорости выполнения: вместо передачи параметров, возврата управления и результатов
- Включение/выключение подстановки в коде

Inline-подстановка в PL/SQL

Новый уровень оптимизации (level 3)

```
alter session set plsql_optimize_level=3;
```

- Оптимизатор PL/SQL сам делает подстановку (inline) исходя из:
 - Размера процедуры/функции
 - Предполагаемой частоты вызова (вызов в цикле)
 - Числа и типов параметров
- Проведены тесты на системе E-Business Suite
 - Система состоит из больших PL/SQL-пакетов с небольшими процедурами
 - Получено увеличение быстродействия на ~20%
 - В общем случае выигрыш зависит от структуры PL/SQL-кода

Поиск числа вхождений подстроки

До PL/SQL 11g

```
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);
```

- Как узнать число вхождений ?
 - Делать цикл: находить вхождение, а затем сдвигать позицию с которой начинается поиск
 - Очень неэффективно!

Расширение в поддержке регулярных выражений в SQL и PL/SQL

В PL/SQL 11g

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

- Новая функция *Regexp_Count* – число вхождений подстроки по рег. выражению
- Функции *Regexp_Instr* и *Regexp_Substr* теперь имеют опциональный параметр *Subexpr* - дает возможность получить номер символа по подвыражению шаблона

Потенциальная проблема в безопасности

До PL/SQL 11g

- СУБД Oracle Database имеет встроенные пакеты для работы по протоколу TCP/IP:
 - *UTL_TCP, UTL_SMTP, UTL_HTTP*
- Если вы имеете права на выполнение на эти пакеты, то вы можете работать с любым хостом и по любому порту !
- Необходимо давать права на выполнение напрямую пользователям, и ни в коем случае не через PUBLIC

Гибкий контроль доступа для *UTL_TCP* и пакетов на его основе

В PL/SQL 11g

- Возможность создания списков доступа - Access Control List (ACL) которые включают в себя роли и пользователей
- ACL содержит хост и диапазон портов
- ACL управляются с помощью XDB

Новые синтаксические конструкции



Работа с циклами в PL/SQL

До 11g: Переход на следующий шаг цикла

```
begin
  for f in (select * from employees)
  loop
    if f.Salary < 1000 then
      goto next_iteration;
    end if;

    ...

    <<next_iteration>>
    null;
  end loop;
end;
```

Новый оператор “continue” в PL/SQL

PL/SQL 11g: Переход на следующий шаг цикла

```
begin
  for f in (select * from employees)
  loop
    if f.Salary < 1000 then
      continue;
    end if;

    ...

  end loop;
end;
```

Новый оператор “continue” в PL/SQL

PL/SQL 11g: Переход на следующий шаг внешнего цикла

```
<<outer>>for i in 1..10 loop
  ...

  <<inner>>for j in 1..Data.Count() loop

    if Data(j).Uninteresting then
      continue outer;
    end if

    ...

  end loop;
end loop;
```

Новый оператор “continue” в PL/SQL

PL/SQL 11g: Переход на следующий шаг внешнего цикла по условию

```
<<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;
```

Вызов метода super-класса в PL/SQL

До 11g: Вызов перекрытого метода в классе-потомке

```
create or replace type TObject as object
(
  ...
  member function getName return varchar2,
  ...
)
not final;
```

-- Перегружаем метод getName в типе-потомке

```
create or replace type TMyObject under TObject
(
  ...
  overriding member function getName return varchar2,
  ...
);
```

Вызов метода super-класса в PL/SQL

До 11g: Невозможно напрямую вызвать перекрытый метод в классе-потомке

```
create or replace type body TMyObject is
...

overriding member function getName return varchar2 is
begin
  return '_' || self.getName; //Как вызвать метод предка ???
end;                                //Super – в Java
                                    //Inherited в Pascal

...
end;
```

- Приходилось использовать для этого различные *workarounds*
- Например: http://www.citforum.ru/database/oracle/oo_pl_sql/

Вызов метода super-класса в PL/SQL

PL/SQL 11g: Вызов переопределенного метода в классе-потомке

```
overriding member function getName return varchar2 is
begin
  return '_' || (self as TObject).getName;
end;
```

- Выполняется явное преобразование к нужному типу-предку с помощью оператора AS

Вызов функции в SQL - выражениях

До 11g: возможна только позиционная нотация

```
SQL> select
2     get_amount(v_pAccId    => a.Id,
3                 v_pCurrency => 'USD') as Amount
4 from
5     accounts a
6 /
```

```
get_amount(v_pAcclId => a.Id,
            *
            )
```

ERROR at line 2:

ORA-00907: missing right parenthesis

- Это создавало проблемы для перегруженных (overload) функций

Вызов функции в SQL - выражениях

В 11g: возможна нотация по имени, а также смешанная

```
SQL>select
2     get_amount(a.Id,
3                 v_pCurrency => 'USD') as Amount
4   from
5     accounts a
6 /
```

Amount

538

537

536

535

4 rows selected.

Возможное скрытие исключений

До 11g: ошибка - потеря исключения

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

- “*when others then null*” – исключение “проглатывается”
- Это приводит к трудно обнаруживаемым ошибкам

Новые сообщения от компилятора

В PL/SQL 11g: новые предупреждения компилятора

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

- Компилятор выдает предупреждение:

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

- Много других новых предупреждений, например от оптимизатора:

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

Новые возможности в динамическом SQL



Виды динамического SQL в PL/SQL

До PL/SQL 11g: виды динамического SQL в PL/SQL

- Пакет DBMS_SQL
 - Используется, если на этапе компиляции неизвестна структура курсора, число и типы переменных привязки
 - Позволяет выяснить структуру курсора в run-time
 - Сложный синтаксис !
- Native Dynamic SQL, NDS (операторы EXECUTE IMMEDIATE и OPEN FOR)
 - Число и типы переменных привязки должны быть известны в время компиляции
 - Структура курсора должна быть известна !
 - Простой и короткий синтаксис !

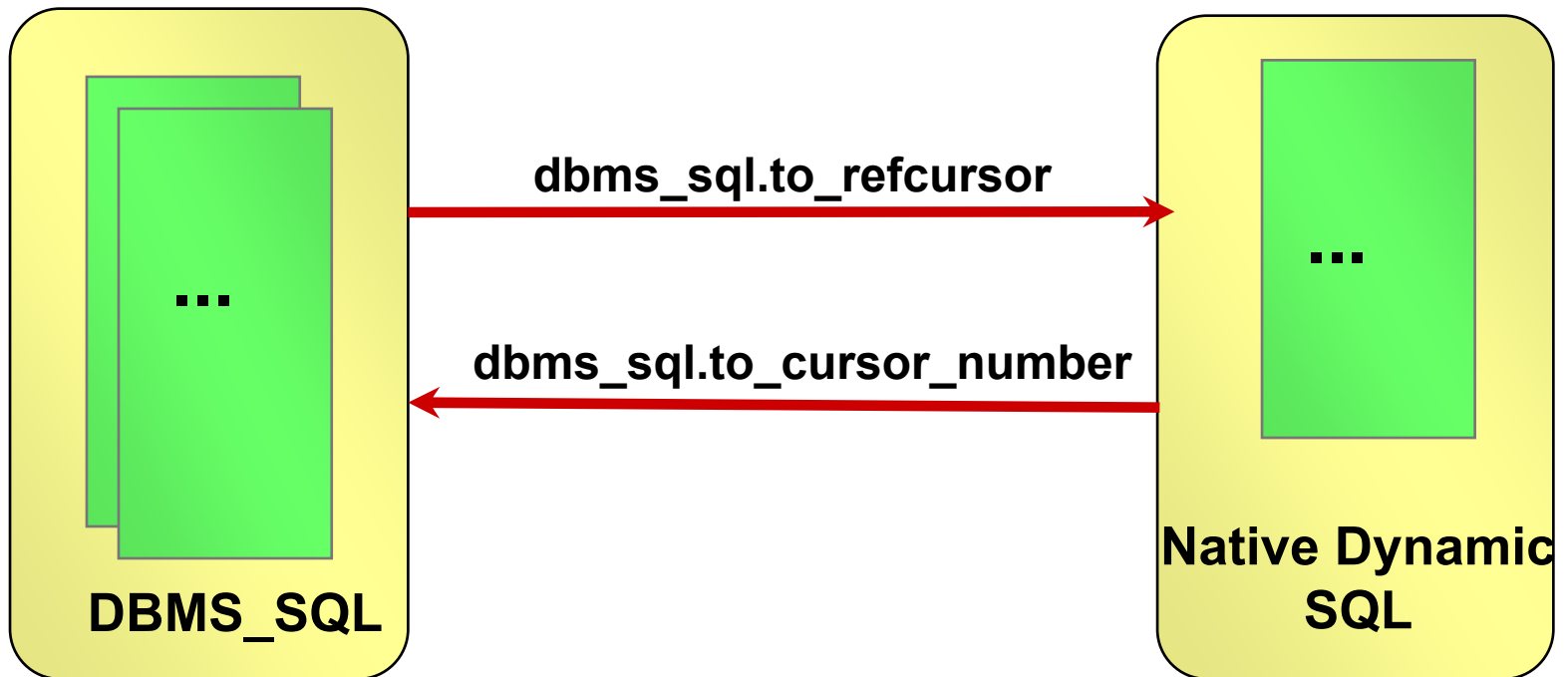
Выбор между видами dynamic-SQL

До 11g: Преимущества NDS по сравнению с DBMS_SQL

```
procedure insert_into_table(pTableName varchar2,  
                           pDeptName  varchar2) is  
  
    stmt_str  varchar2(200);  
    cur_hdl   integer;  
    rows_proc pls_integer;  
begin  
    stmt_str := 'insert into ' || pTableName ||  
                ' values (:dname);';  
    cur_hdl := DBMS_SQL.OPEN_CURSOR;  
    DBMS_SQL.PARSE(cur_hdl, stmt_str, dbms_sql.native);  
    DBMS_SQL.BIND_VARIABLE(cur_hdl, ':dname', pDeptName);  
    rows_proc := DBMS_SQL.EXECUTE(cur_hdl);  
    DBMS_SQL.CLOSE_CURSOR(cur_hdl);  
  
    EXECUTE IMMEDIATE stmt_str USING pDeptName;  
  
end;
```

Переключение между видами dynamic SQL

В PL/SQL 11g: Динамическое переключение



Новое в динамическом SQL

PL/SQL 11g: Переключение из DBMS_SQL в NDS

```
declare
  cur_num number := DBMS_SQL.OPEN_CURSOR();
  cur_ref sys_refcursor;
  type     emps_t is table of employees%rowtype;
  emps     emps_t;
begin
  DBMS_SQL.PARSE(c          => cur_num,
                 Language_Flag => DBMS_SQL.Native,
                 Statement     => 'select * from Employees
                                   where Department_ID = :d and ...');
  ...
  dummy := DBMS_SQL.EXECUTE(cur_num);

-- Переключение в ref cursor и NDS
  cur_ref := DBMS_SQL.TO_REFCURSOR(cur_num);

  fetch cur_ref bulk collect into emps;
  close cur_ref;
```

Новое в динамическом SQL

PL/SQL 11g: Переключение из NDS в DBMS_SQL

```
declare
  cur_num number;
  cur_ref sys_refcursor;
  type     emps_t is table of employees%rowtype;
  emps     emps_t;
begin
  open cur_ref for 'select * from employees
                  where Department_ID = :d and ...';

  -- Переключение из NDS в DBMS_SQL
  cur_num := DBMS_SQL.TO_CURSOR_NUMBER(cur_ref);

  -- Получаем структуру курсора
  DBMS_SQL.DESCRIBE_COLUMNS2(cur_num, col_cnt, col_desc);
  ...
  DBMS_SQL.CLOSE_CURSOR(cur_num);
end;
```

Новое в динамическом SQL

PL/SQL 11g: Замечания по переключению

- Переключение однонаправлено (нельзя возвратиться в исходный вид):
 - DBMS_SQL -> NDS
 - NDS -> DBMS_SQL
- Курсор закрывается только в том виде dynamic-SQL, куда произошло переключение
 - Оператор *CLOSE* в NDS
 - Процедура *CLOSE_CURSOR* в пакете DBMS_SQL
- Кэширование курсоров в DBMS_SQL продолжает выполняться (“*cost saving paradigm*”)

Новое в динамическом SQL

PL/SQL 11g: Другие новшества в динамическом SQL

- DBMS_SQL.PARSE может принимать на вход строки типа CLOB (> 32Kb)
 - Теперь нет необходимости в использовании VARCHAR2S
- EXECUTE IMMEDIATE поддерживает CLOB-строки
- Переменные привязки могут быть объектного типа, в том числе и коллекции
 - Раньше приходилось использовать для этого NDS
- Пакетное связывание (*bulk binding*) поддерживается для коллекций объектных типов

Новое в триггерах



Создание триггера в отключенном состоянии

В 11g: Триггер создан, но отключен

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

- Если триггер создан с ошибками компиляции, то любая DML-операция над таблицей будет приводить к ошибке: *“ORA-04098: trigger 'TRG' is invalid and failed re-validation”*
- Возможность создать триггер и включить его после – когда точно известно, что нет ошибок PL/SQL

Порядок срабатывания триггеров

В 11g: можно задавать порядок срабатывания

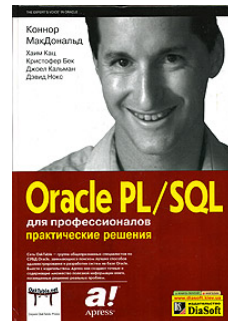
```
create or replace trigger Trg_2
  before insert on My_Table for each row
  follows Trg_1
begin
  ...
end;
/
```

- Раньше порядок срабатывания не гарантировался !

Триггер – не модуль PL/SQL

До 11g

- Триггер не являлся модулем PL/SQL
- С точки зрения PL/SQL VM - это анонимный блок
- Из-за этого не в теле триггеров не выполняется кэширование SQL-вызовов
- Коннор МакДональд, “PL/SQL для профессионалов”, стр. 238: “... избегайте выполнения SQL-операторов в триггере. Если это необходимо, поручите это действие процедуре.”



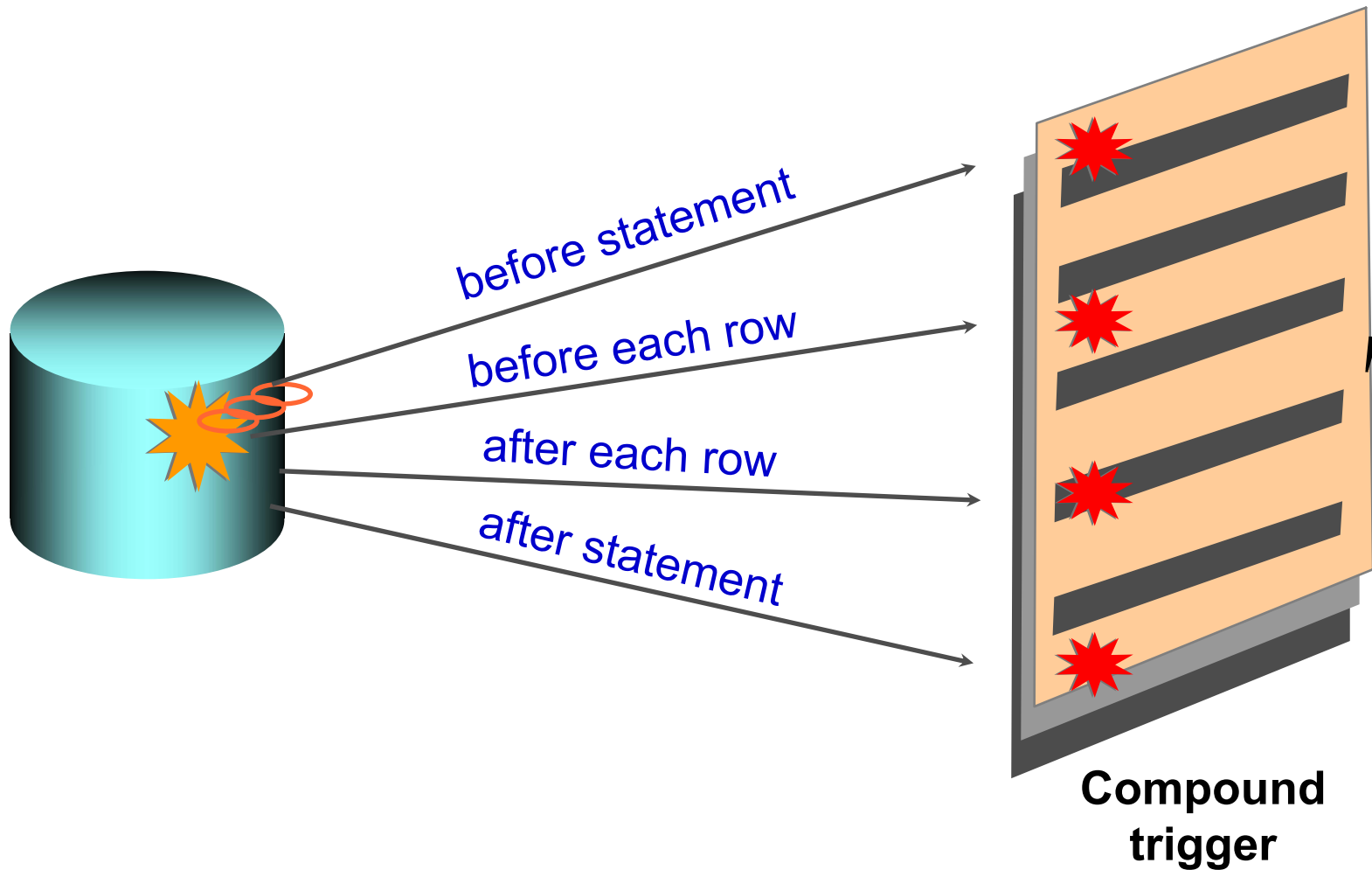
Триггер – модуль PL/SQL

В 11g

- Теперь триггер являлся модулем PL/SQL
- Исходный код триггера, как и любого PL/SQL-модуля виден в USER_SOURCE
- Теперь выполняется кэширование SQL-вызовов в теле триггера
- *“Fast DML triggers - DML triggers are up to 25% faster”*

Новый тип триггеров

В 11g: составной триггер (compound trigger)



The compound trigger

В 11g: новый тип триггеров

- Новый тип триггеров: Compound trigger – дает возможность обрабатывать все DML-операции над таблицей в одном триггере
- Может включать в себя глобальные переменные используемые в всех обработчиках событий:
 - Можно включать явный код инициализации триггера – выполняется один раз на оператор – перед выполнением обработчика “before statement”
 - Можно определять секцию завершения (finalization) Выполняется после обработки DML-оператора: за обработчиком “after statement” (даже если ранее была ошибка!)
 - Глобальные переменные триггера уничтожаются после завершения работы DML-оператора

The compound trigger

В 11g: Пример составного триггера

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

-- Эти переменные могут использоваться в всех обработчиках триггера
  Threshold constant pls_integer := 200;
  before statement is
  begin
    ...
  end before statement;

-- обработчик операции "after each row"
  before each row is
  begin
    null;
  end before each row;

-- деструктор триггера - обработчик "after statement"
  after statement is
  begin
    null;
  end after statement;
end;
```

The compound trigger

В 11g: Пример составного триггера

```
create trigger My_Compound_Trigger
  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_Trigger;
```

The compound trigger

В 11g: Пример - продолжение

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

  ...

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

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

  ...

end My_Compound_Trg;
```

The compound trigger

В 11g: Пример - завершение

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

...

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

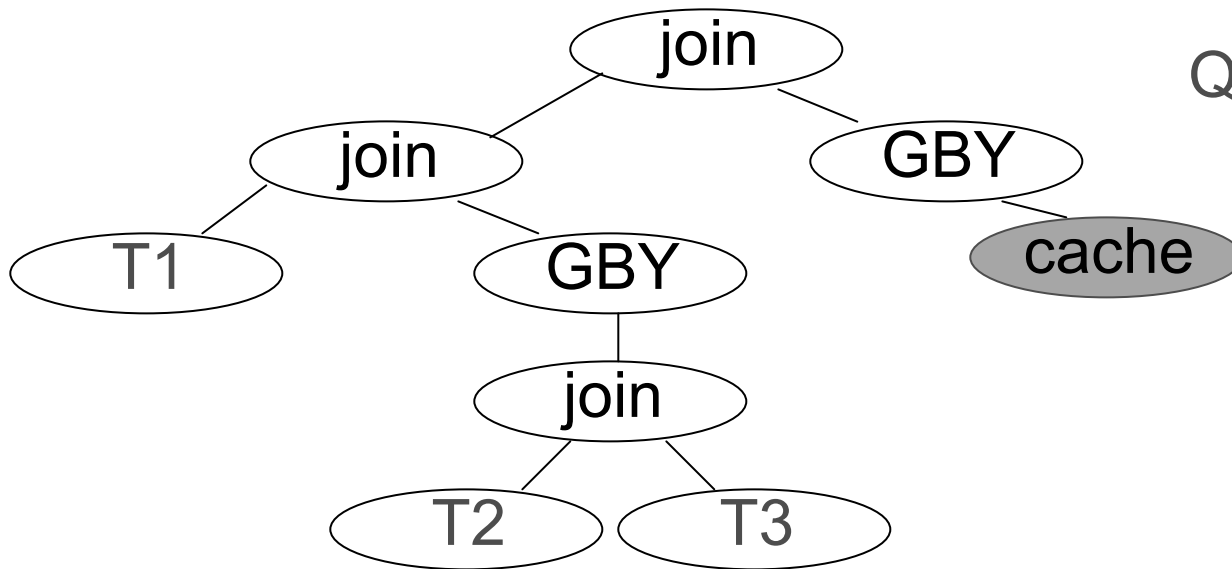
end My_Compound_Trigger;
```



Технология Result Cache

SQL Query Result Cache

- Возможно кэширование результатов запросов, подзапросов (query blocks)
 - Кэш совместно используется различными SQL операторами и сессиями пользователей
 - Значительное ускорение для операций чтения (read-only / read-mostly data)



Q2: Использует это прозрачно

SQL Query Result Cache

Кэширование запросов

- Несколько уровней контроля
 - Оператор: hint `result_cache, no_result_cache`
 - Сессия: параметр `RESULT_CACHE_MODE = force | manual | auto`
 - `force` - кэшировать все запросы
 - `manual` - кэшировать только запросы с hints
 - `auto` - решение о кэшировании принимает оптимизатор
- Полная согласованность результата
 - Кэш обновляется при изменении таблиц, из которых получен кэшируемый результат

SQL Query Result Cache

Пример

```
select /*+ RESULT_CACHE */
       p.prod_category,
       sum(s.amount_sold) revenue
from
       products p,
       sales      s
where
       s.prod_id = p.prod_id and
       s.time_id
       between to_date('01-JAN-2006','dd-MON-yyyy')
              to_date('31-DEC-2006','dd-MON-yyyy') and
group by
       rollup (p.prod_category)
```

SQL Query Result Cache

Пример – план запроса

Id	Operation	Name
0	SELECT STATEMENT	
1	RESULT CACHE	fz6cm4jbpcwh48wcyk60m7qypu
2	SORT GROUP BY ROLLUP	
* 3	HASH JOIN	
4	PARTITION RANGE ITERATOR	
* 5	TABLE ACCESS FULL	SALES
6	VIEW	index\$_join\$_001
* 7	HASH JOIN	
8	INDEX FAST FULL SCAN	PRODUCTS_PK
9	INDEX FAST FULL SCAN	PRODUCTS_PROD_CAT_IX

SQL Query Result Cache

Ограничения

- Кэширование отключено для запросов содержащих
 - Временные или dictionary-таблицы
 - Недетерминированные PL/SQL-функции
 - Обращение к последовательности (CURRVAL и NEXTVAL)
 - Недетерминированные SQL-функции: `current_date`, `sysdate`, `sys_guid` т.д.
- Result cache для распределенных запросов:
 - `result_cache_remote_expiration > 0`
 - 0 – не кэшировать (по умолчанию)
 - DML/DDDL на удаленной БД не приводит к обновлению кэша

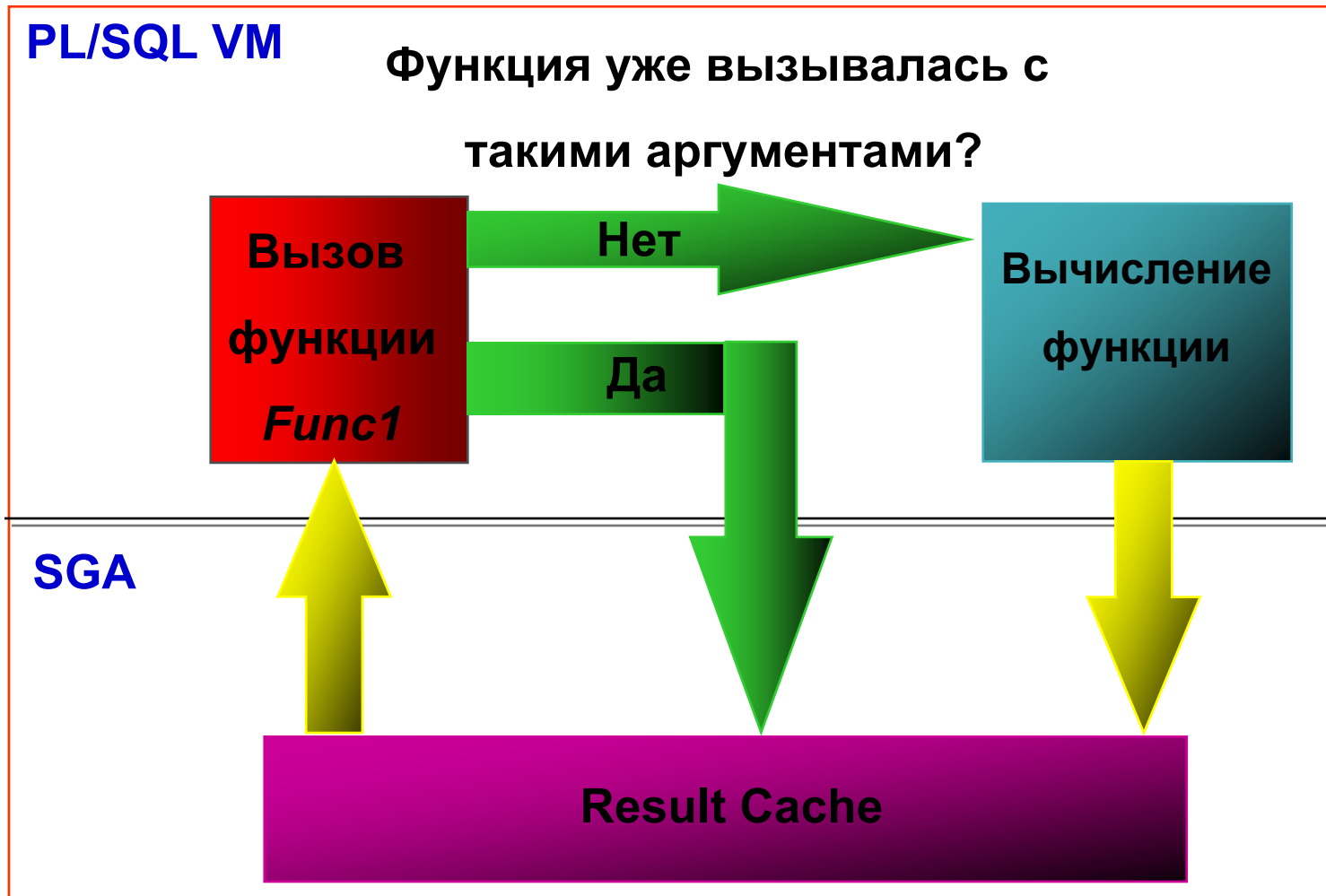
PL/SQL Result Cache

В 11g: Кэширование результатов вычисления PL/SQL-функций

- Подстановка значения из кэша вместо вычисления функции
- Если функция уже выполнялась с теми же самыми параметрами, то вместо выполнения функции PVM может брать значение из этого кэша !
- Разделяется между сессиями
- Может применяться ТОЛЬКО для детерминированных функций, и функций не имеющих побочных эффектов (определяет разработчик) !

PL/SQL Result Cache

В 11g: Принципы работы



PL/SQL Result Cache

В 11g: Создание кэшируемой функции

- При создании функции указывается:
 - Ключевое слово RESULT_CACHE
 - Фраза RELIES_ON – перечень таблиц от которых зависит результат функции
- При изменении таблиц указанных в RELIES_ON кэш для этой функции автоматически очищается

```
create function getAmount(pAccountId in number,  
                          pCurrency   in varchar2) return number  
  result_cache relies_on (accounts, accounts_amounts); is  
begin  
  ... ..  
  return xRes;  
end;
```

PL/SQL Result Cache

В 11g: управление кэшем результатов

- Фраза RELIES_ON может быть опущена
- При этом задача поддержка актуальности кэша возлагается на программиста
 - Разработчик определяет события по которым кэш очищается (например: триггер, API - пакет таблицы)
- Для этого служит пакет DBMS_RESULT_CACHE и его метод INVALIDATE

```
begin
  ... ..
  if xHasChanged then
    DBMS_RESULT_CACHE.INVALIDATE (USER, 'getAmount') ;
  end if;
  ... ..
end;
```

PL/SQL Result Cache

В 11g: API для управления кэшем результатов

- Встроенный пакет DBMS_RESULT_CACHE
 - INVALIDATE – очистка кэша для функции
 - FLUSH – очистка всего кэша
 - BYPASS – включение/выключение кэша на уровне экземпляра
 - STATUS – проверка статуса кэша
 - MEMORY_REPORT – вывод отчета об использовании кэша

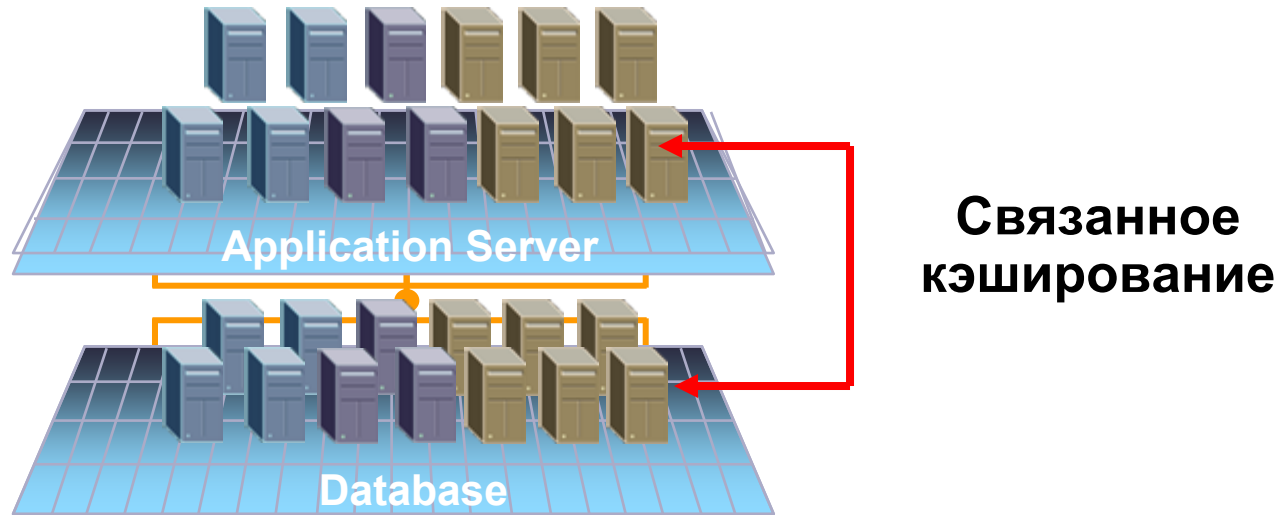
SQL & PL/SQL Result Cache

Настройка

- Result Cache располагается в разделяемом пуле
 - Разделяется всеми сессиями
- Параметры инициализации
 - RESULT_CACHE_MAX_SIZE – задает макс. размер кэша в SGA
 - RESULT_CACHE_MAX_RESULT – макс. % от кэша для одного объекта
 - RESULT_CACHE_REMOTE_EXPIRATION – кэширование объектов по db-link (через сколько секунд проверять актуальность)
- Мониторинг – динамические представления
 - [G]V\$RESULT_CACHE_STATISTICS
 - [G]V\$RESULT_CACHE_OBJECTS
 - [G]V\$RESULT_CACHE_DEPENDENCY
 - [G]V\$RESULT_CACHE_MEMORY

OCI Client Result Cache

Кэширование на клиенте



- Кэширует результаты запроса на клиенте
- Улучшает производительность работы с таблицами, используемыми в основном для чтения (read-mostly)
 - Более быстрое время отклика – исключается передача по сети
 - Уменьшает нагрузку на процессоры сервера
- Согласован с сервером
 - Кэш проактивно обновляется, когда изменяется выборка
 - Сихронизация кеша (как в RAC)

OCI Client Result Cache

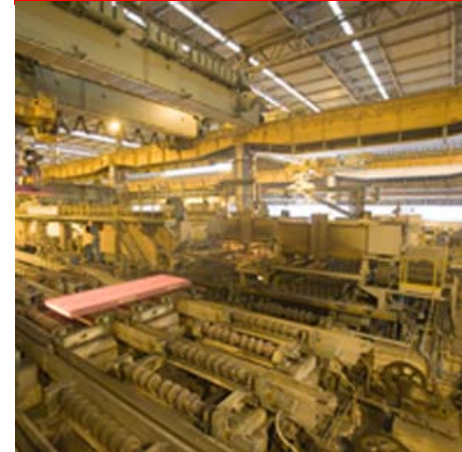
- Унифицированный интерфейс с Server result cache
- Кэш процесса разделяется несколькими сессиями
- Параметр CLIENT_RESULT_CACHE_LAG
 - Через сколько миллисекунд после последнего обращения к серверу делать проверку кэша на актуальность
- Параметр CLIENT_RESULT_CACHE_SIZE
 - Максимальный объем памяти выделяемый на клиенте под кэш
- Работает со всеми 11g OCI-based клиентами
 - Включая ODP.Net, JDBC OCI Driver, PHP, ODBC

OCI Client Result Cache

Настройки на клиенте

- Параметры в файле sqlnet.ora
 - OCI_RESULT_CACHE_MAX_SIZE (имеет приоритет перед CLIENT_RESULT_CACHE_SIZE)
 - OCI_RESULT_CACHE_MAX_RSET_SIZE
 - OCI_RESULT_CACHE_MAX_RSET_ROWS

Новые инструменты для разработчиков



Анализ и оптимизация производительности в PL/SQL

В 11g: новый профайлер

- Встроенный пакет DBMS_HPROF
 - На выходе формируется trc-файл (текстовый)
 - Вывод дополнительной информации: dynamic-sql, static-sql, init-секция и т.д.
 - Показ иерархии вызовов
 - plshprof – утилита для преобразования trc-файла в html-отчет
 - Не требуется перекомпиляция (DEBUG) и изменение кода

Анализ и оптимизация производительности в PL/SQL

В 11g: пример работы профайлера

- Включить профилирование для процедуры pkg.myproc

```
VARIABLE runid NUMBER;  
BEGIN  
    :runid := DBMS_HRPROF.analyze(  
        'PLSHPROF_DIR',  
        'test.trc',  
        trace => '"HR"."PKG"."MYPROC"');  
END;
```

- Получить отчет
 - `% plshprof -output report test.trc`

Анализ и оптимизация производительности в PL/SQL

В 11g: пример отчета профайлера

2831 microseconds (elapsed time) & 12 function calls

Subtree	Ind%	Function	Descendant	Ind%	Calls	Ind%	Function Name
2831	100%	93	2738	96.7%	2	16.7%	__plsq_vm
2738	96.7%	310	2428	85.8%	2	16.7%	__anonymous_block
2428	85.8%	15	2413	85.2%	1	8.3%	HR.TEST.TEST (Line 1)
2413	85.2%	435	1978	69.9%	3	25.0%	HR.TEST.TEST.FOO (Line 3)
1978	69.9%	1978	0	0.0%	3	25.0%	HR.TEST.__static_sql_exec_line5 (Line 5)
0	0.0%	0	0	0.0%	1	8.3%	SYS.DBMS_HPROF.STOP_PROFILING (Line 53)

Анализ исходного кода PL/SQL

В 11g: новый инструмент - PL/Scope

- Сбор информации об всех идентификаторах исходного кода
 - Аналог Cscope для языка C (<http://cscope.sourceforge.net>)
 - Параметр PLSCOPE_SETTINGS – для определения типа собираемых идентификаторов
 - Представление USER_IDENTIFIERS – для просмотра
 - Предварительно нужна перекомпиляция
 - Для wrapped-кода сбор информации невозможен !
 - Интеграция с SQL Developer 2.0

Анализ исходного кода PL/SQL

В 11g: PL/Scope - пример

```
SQL> alter session set plscope_settings='identifiers:all';
```

```
SQL> alter type TObject compile;
```

```
SQL> select name, type, usage from USER_IDENTIFIERS;
```

NAME	TYPE	USAGE
V_PSTATE	FORMAL IN	DECLARATION
SELF	FORMAL IN OUT	DECLARATION
SETSTATE	PROCEDURE	DECLARATION
SETSTATE	PROCEDURE	DEFINITION
GETSTATE	FUNCTION	DECLARATION
GETSTATE	FUNCTION	DEFINITION
GETNEWID	FUNCTION	DECLARATION
GETNEWID	FUNCTION	DEFINITION
GETID	FUNCTION	DECLARATIO

Заключение



Заключение

Резюме по новым возможностям в PL/SQL

- Улучшение производительности
 - Отслеживание зависимостей на уровне элементов объектов
 - Real PL/SQL native compilation
 - Inline-подстановка
 - SQL & PL/SQL Result Caches
 - Составные триггеры
- Для того чтобы и использовать эти возможности требуются лишь небольшие усилия!

Заключение

Резюме по новым возможностям в PL/SQL

- Новая функциональность:
 - Расширения в динамическом SQL
 - *DBMS_SQL* – улучшение в безопасности
 - Контроль доступа через пакеты *UTL_TCP*, и д.р.
 - *Regexr_Count()*, в SQL и PL/SQL
 - Поддержка “super” – вызов перегруженного метода в ООП PL/SQL
 - Создание отключенного триггера; порядок срабатывания триггеров
 - “when others then null” – новые предупреждения компилятора

Заключение

Резюме по новым возможностям в PL/SQL

- Удобство использования
 - Последовательности в PL/SQL-выражениях
 - Оператор *continue*
 - Именованная и смешанная нотация вызовов PL/SQL в SQL

- Инструменты
 - PL/Scope
 - PL/SQL Hierarchical Profiler

Информация

- Общая информация: www.oracle.com/database
- Технологическая информация:
Oracle Technology Network
<http://otn.oracle.com/products/database/oracle11g>
- Информация для контактов:
Игорь Мельников
e-mail: Igor.Melnikov@oracle.com
тел. +7 (495) 641-14-42



ORACLE®

THE INFORMATION COMPANY