Blacky 2024

APEX connect 2024 - Düsseldorf vom 22 bis 24. April 2024

Die Ausführung von PL/SQL Routinen messen , analysieren und optimieren

PL/SQL - SELECT FAST FROM PLSQL - PERFORMANCE ANALYSE UND TUNING

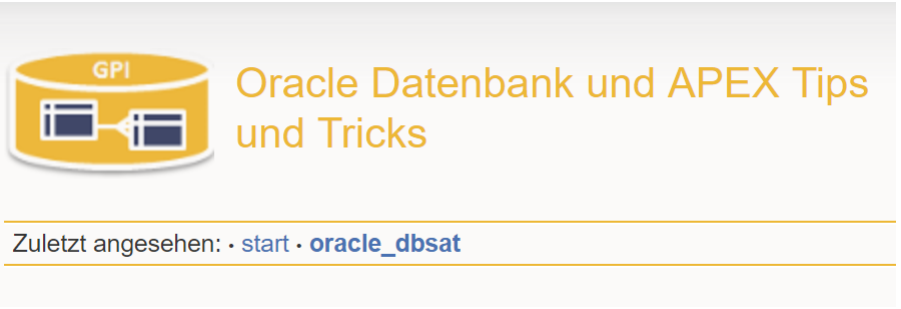
GPI Consult Gunther Pippèrr

ACE Associate

gunther@pipperr.de

Mein Blog

https://www.pipperr.de/dokuwiki/



Bergweg 14 - 37216 Witzenhausen/Roßbach
Im Kaufunger Wald

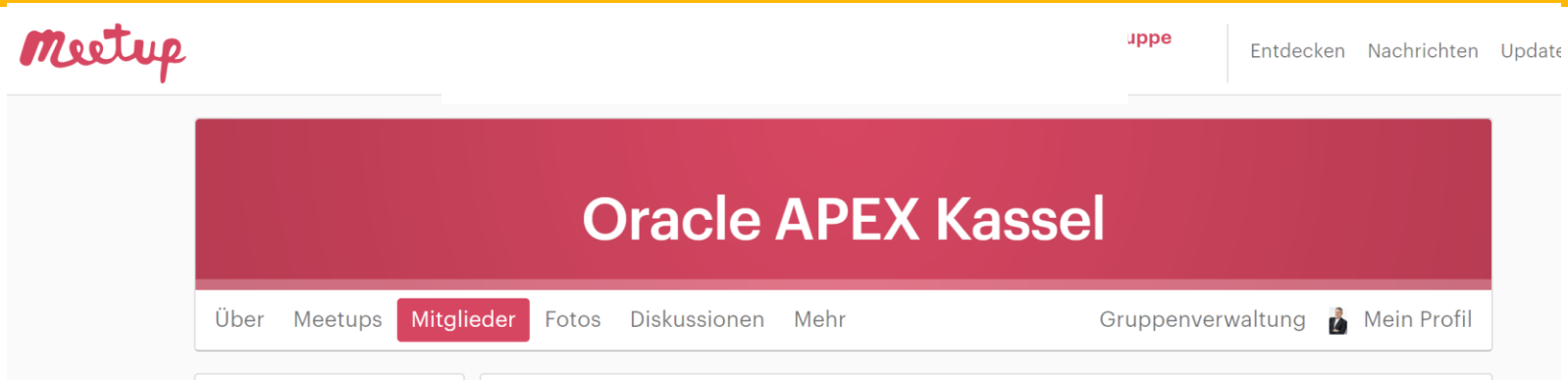
Freiberuflicher Oracle Datenbank Experte - Ich unterstütze Sie gerne in ihren Projekten.

APEX Meetup Gruppe Kassel-Göttingen

Raum für Veranstaltung in Kassel oder Göttingen gesucht, können Sie uns unterstützen ?

SELECT APEX FAN from KASSLER GEGEND

https://www.meetup.com/de-DE/Oracle-APEX-Kassel/



Agenda

1 **Die PL/SQL Architektur der Datenbank**

2 PL/SQL Performance in der Datenbank analysieren und auswerten

3 Der PL/SQL Optimizer

4 Erste, einfache Maßnahmen, um performantes PL/SQL zu entwickeln

5 Aufruf von "SQL ⇔ PL/SQL" optimieren - SQL Transpiler in Oracle 23c

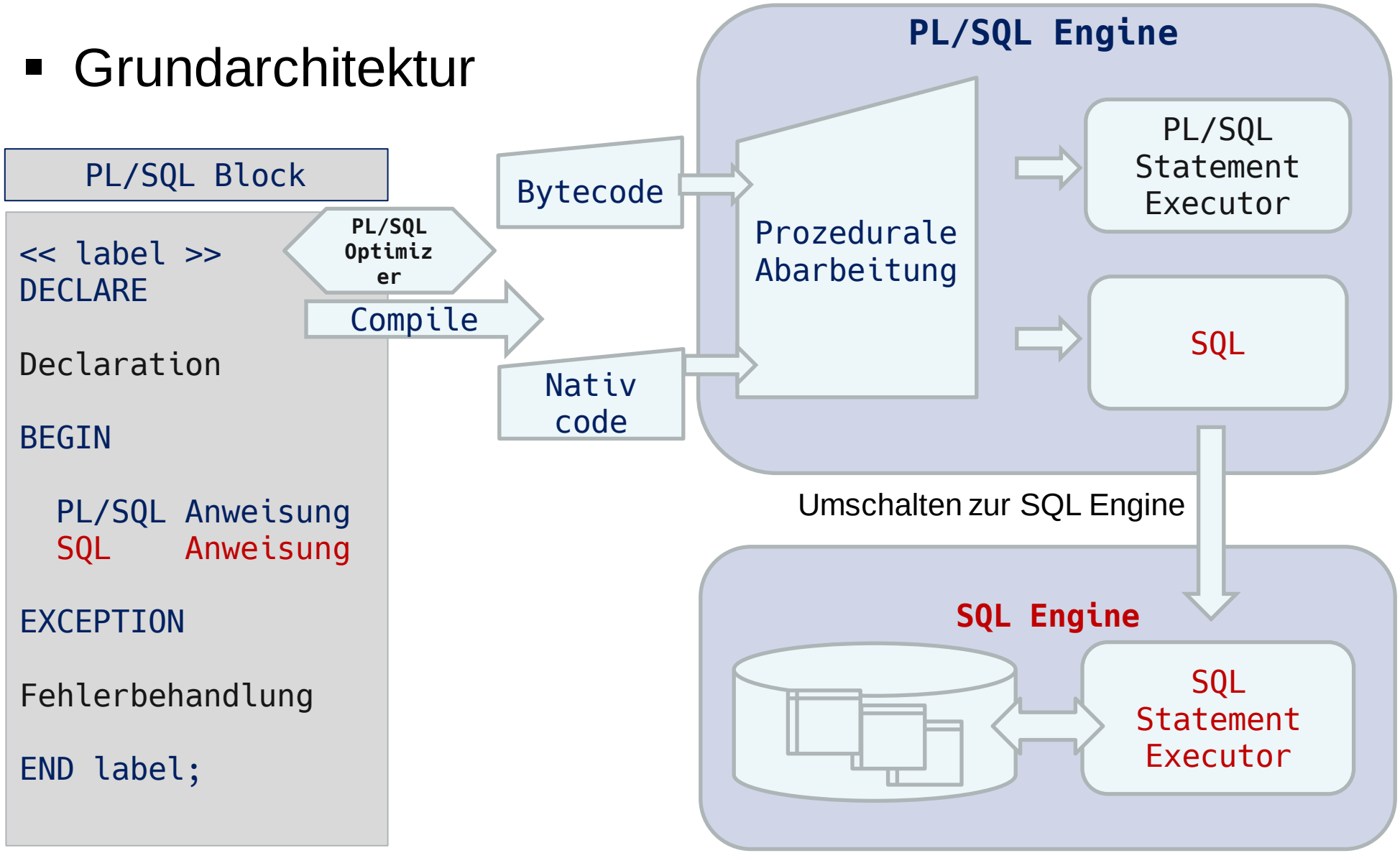
PL/SQL – Die Bewerbungsgespräch Fakten

- PL/SQL – Procedural Language extensions to Structured Query Language“
- Basiert auf der Programmiersprache „Ada“
  - Eine Variante der “Descriptive Intermediate Attributed Notation” for Ada (DIANA), einer baumstrukturierten Zwischensprache
  - Diese wird mit Metanotation namens Interface Definition Language (IDL) definiert
  - Der DIANA code wird intern von Compilern etc. verwendet
- Implementiert den SQL/PSM Standard
- Inzwischen in Teilen Standardisiert
- Seit Oracle 6 , seit 7 „stored“

Mehr https://oracle-internals.com/blog/2020/04/29/a-not-so-brief-but-very-accurate-history-of-pl-sql/

Die Oracle PL/SQL Engine

- Grundarchitektur



Die Performance Ansätze aus der Architektur

- PL/SQL-Optimierung
  - Compiler Settings
  - Speicher Verwendung
  - Variablen Eigenschaften
  - Schleifen Logik
  - Aufruf von SQL

- SQL-Optimierung
  - Daten Aggregation
    - Ausführungsplan

Regel 1a – Möglichst kleine Anzahl von SQL-Aufrufe in PL/SQL

Regel 1b – Möglichst kleine Anzahl von PL/SQL Aufrufen in SQL

Wo lässt sich das meiste für PL/SQL herausholen

Native Compilation

NOCOPY Hint

Datentypen optimieren

PL Compiler Automatic Optimization (Level 3)

Klassische Code Optimierung

Pragma UDF

FORALL / BULCK COLLET

RESULT_CACHE

DML mit DBMS_PARALLEL

Commit Handling

Nativ SQL

!!! SQL Always First !!!

Wie erkenne ich PL/SQL Probleme?

- Der DBA

  - Der Entwickler quengelt, dass alles so langsam ist
  - Der Kaufmann jammert, warum wir so viel OCPU benötigen
  - Auf DB & CPU-Time ⇔ PL/SQL-Time bei der Auswertung der gesamten DB-Umgebung achten
    - Auswertung über V$SYS_TIME_MODEL
    - Stichwort " PL/SQL execution elapsed time"

https://www.oracle.com/technical-resources/articles/schumacher-analysis.html

- Der Entwickler

  - Der Anwender quengelt, dass alles so langsam ist
  - Auswertung mit dem Profiler Tools

PL/SQL – Wo wird der Code gespeichert?

- Wo ist mein Code? In der Tabelle SOURCE$

```
select NAME, TYPE, LINE, TEXT
  from USER_SOURCE
 where type like 'P%’
 order by  1,2,3;
```

NAME	TYPE	LINE	TEXT
BULCK_LOAD_TEST	PROCEDURE	1	procedure bulck_load_test (p_testcase in pls_integer
BULCK_LOAD_TEST	PROCEDURE	2	, p_key in varchar2)
BULCK_LOAD_TEST	PROCEDURE	3	as
BULCK_LOAD_TEST	PROCEDURE	4	type cur_typ is ref cursor;
BULCK_LOAD_TEST	PROCEDURE	5	
BULCK_LOAD_TEST	PROCEDURE	6	type driver_id is table of varchar2 (9) index by binary_integer;
BULCK_LOAD_TEST	PROCEDURE	7	
BULCK_LOAD_TEST	PROCEDURE	8	v_cur cur_typ;
BULCK_LOAD_TEST	PROCEDURE	9	v_driverids driver_id;
BULCK_LOAD_TEST	PROCEDURE	10	
BULCK_LOAD_TEST	PROCEDURE	11	-- mit bind variablen
BULCK_LOAD_TEST	PROCEDURE	12	p_sql varchar2 (200) := 'update EMPLOYEES2 set SALARY = SALARY*0.19 where DEPARTMENT_ID = :key';

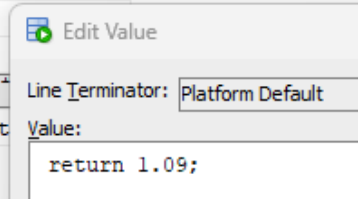
PL/SQL Code anpassen, dass es keiner merkt

- Funktion anlegen

```
create function getStatic
return number
is
begin
 return 1.19;
end getStatic ;
```

- SOURCE$ als SYS ändern

OBJ#	LINE	SOURCE
100452	1	function getStatic
100452	2	return number
100452	3	is
100452	4	begin
100452	5	[illegible]
100452	6	end getSt



Solange nichts übersetzt wird, passiert auch nichts.
Uhrzeiten blieben unverändert!

```
select CREATED, LAST_DDL_TIME, TIMESTAMP
from user_objects where OBJECT_ID=100452;
```

```
select getStatic from dual;
1.19
```

```
alter function getStatic compile;
```

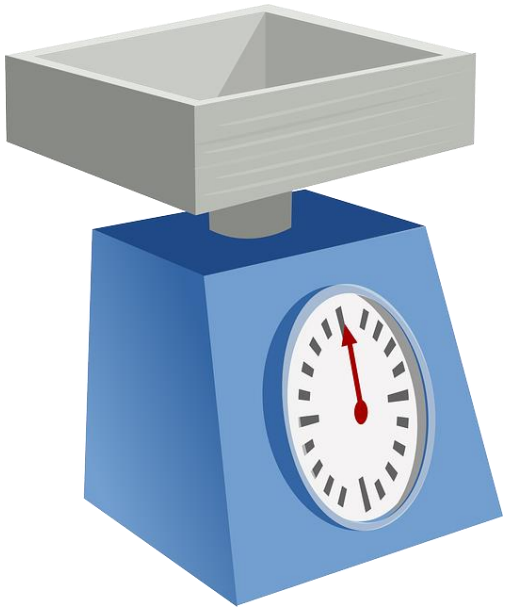
```
select getStatic from dual;
1.09
```

Audit ? – Guter Platz für ein Easter Egg für Privilegien Eskalation?

Wie sehen wir wo die Zeit bleibt?

- Messen - Aber wie und was?

- Stoppuhr
  - DBMS_UTILITY.GET_TIME;
  - DBMS_UTILITY.GET_CPU_TIME
  - Hundertstel Sekunde genau



- Profiling
  - PL/SQL Profiler
    - DBMS_PROFILER
    - Vorteil: kein Zugriff auf DB-Server notwendig!
  - PL/SQL Hierarchical profiler (11g Release 1)
    - DBMS_HPROF

https://www.oracle-developer.net/display.php?id=307

PL/SQL Hierarchical Profiler (1)

- DBMS_HPROF - ab Oracle 11g Release 1
  - Nachteil: Zugriff auf DB Directory Objekt notwendig!
- Vorbereitung
  - EXECUTE Rechte auf das Package DBMS_HPROF
  - READ und WRITE Rechte auf ein logisches DB Directory Objekt

```
Als SYS:

create directory PROFILER_DIR as '/opt/oracle/profiler';

grant all on directory PROFILER_DIR to GPI;
grant execute on dbms_hprof to GPI;
grant  SELECT_CATALOG_ROLE  to GPI;
```

  - Repository-Hilfstabellen im Anwender Schema

```
Als PL/SQL User:

execute DBMS_HPROF.CREATE_TABLES();
```

PL/SQL Hierarchical Profiler (2)

- Trace der PL/SQL Routine starten

```
Als PL/SQL User:

execute dbms_hprof.start_profiling(
      LOCATION => ‘PROFILER_DIR‘
   ,  FILENAME => ‘Profile_run_15_03_2024.trc');



execute plsql_test_routine;



execute dbms_hprof.stop_profiling;
```

PL/SQL Hierarchical Profiler (3)

- Trace auswerten im OS mit plshprof

```
Als Oracle User im Betriebsystem:

plshprof /opt/oracle/profiler/Profile_run_15_03_2024.trc
```

Function Elapsed Time (microsecs) Data sorted by Total Subtree Elapsed Time (microsecs)

5293704 microsecs (elapsed time) & 10000026 function calls

Subtree	Ind%	Function	Ind%	Descendants	Ind%	Calls	Ind%	Function Name	SQL ID	SQL TEXT
5293704	100%	12	0.0%	5293692	100%	4	0.0%	__plsql_vm		
5293692	100%	976	0.0%	5292716	100%	4	0.0%	__anonymous_block		
5292626	100%	2710411	51.2%	2582215	48.8%	1	0.0%	GPI.PRAGMARUN.PRAGMARUN (Line 1)		
2582111	48.8%	2582111	48.8%	0	0.0%	10000000	100%	GPI.PRAGMARUN.PRAGMARUN.ADD_NUMBERS (Line 8)		
83	0.0%	72	0.0%	11	0.0%	1	0.0%	SYS.DBMS_OUTPUT.GET_LINES (Line 267)		
56	0.0%	4	0.0%	52	0.0%	2	0.0%	SYS.DBMS_OUTPUT.PUT_LINE (Line 164)		
50	0.0%	50	0.0%	0	0.0%	2	0.0%	SYS.DBMS_OUTPUT.PUT (Line 125)		
32	0.0%	32	0.0%	0	0.0%	2	0.0%	SYS.DBMS_UTILITY.GET_CPU_TIME (Line 1479)		
12	0.0%	12	0.0%	0	0.0%	2	0.0%	SYS.DBMS_UTILITY.GET_TIME (Line 463)		
11	0.0%	11	0.0%	0	0.0%	3	0.0%	SYS.DBMS_OUTPUT.GET_LINE (Line 192)		
7	0.0%	7	0.0%	0	0.0%	1	0.0%	SYS.DBMS_OUTPUT.ENABLE (Line 71)		
4	0.0%	4	0.0%	0	0.0%	1	0.0%	SYS.DBMS_UTILITY.__pkg_init		
2	0.0%	2	0.0%	0	0.0%	2	0.0%	SYS.DBMS_OUTPUT.NEW_LINE (Line 172)		
0	0.0%	0	0.0%	0	0.0%	1	0.0%	SYS.DBMS_HPROF.STOP_PROFILING (Line 747)		

PL/SQL Hierarchical Profiler (4)

- Trace in DB schreiben

```
Als PL/SQL User:

Declare
   v_runid number;
Begin
   v_runid:=DBMS_HPROF.analyze('PROFILER_DIR'
                               ,'Profile_run_15_03_2024.trc');
end;
/

select * from DBMSHP_RUNS;
```

In der DB auswerten

```
Als PL/SQL User:

with run_data as (
SELECT fi.symbolid,
               pci.parentsymid,
               RTRIM(fi.owner || '.' || fi.module || '.' || NULLIF(fi.function,fi.module), '.') AS
name,
               NVL(pci.subtree_elapsed_time, fi.subtree_elapsed_time) AS subtree_elapsed_time,
               NVL(pci.function_elapsed_time, fi.function_elapsed_time) AS function_elapsed_time,
               NVL(pci.calls, fi.calls) AS calls
        FROM   dbmshp_function_info fi
               LEFT JOIN dbmshp_parent_child_info pci ON fi.runid = pci.runid AND fi.symbolid =
pci.childsymid
        WHERE  fi.runid = 1
        AND    fi.module != 'DBMS_HPROF'
)
SELECT RPAD(' ', (level-1)*2, ' ') || a.name AS name,
       a.subtree_elapsed_time,
       a.function_elapsed_time,
       a.calls
FROM   run_data  a
CONNECT BY a.parentsymid = PRIOR a.symbolid
START WITH a.parentsymid =1;
```

Query Result

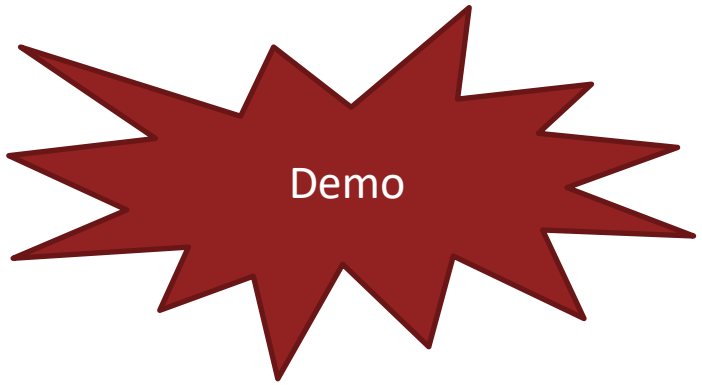
SQL | All Rows Fetched: 11 in 0,014 seconds

	NAME	SUBTREE_ELAPSED_TIME	FUNCTION_ELAPSED_TIME	CALLS
1	GPI.PRAGMARUN	5292626	2710411	1
2	GPI.PRAGMARUN.PRAGMARUN.ADD_NUMBERS	2582111	2582111	10000000
3	SYS.DBMS_OUTPUT.PUT_LINE	56	4	2
4	SYS.DBMS_OUTPUT.NEW_LINE	2	2	2
5	SYS.DBMS_OUTPUT.PUT	50	50	2
6	SYS.DBMS_UTILITY.GET_CPU_TIME	32	32	2
7	SYS.DBMS_UTILITY.GET_TIME	12	12	2
8	SYS.DBMS_UTILITY.__pkg_init	4	4	1
9	SYS.DBMS_OUTPUT.ENABLE	7	7	1
10	SYS.DBMS_OUTPUT.GET_LINES	83	72	1
		11	11	3

https://oracle-base.com/articles/11g/plsql-hierarchical-profiler-11gr1

Der Erweiterte Profiler - DBMS_HPROF

- DBMS_HPROF



01-dbms_hprof_profiler.sql

Standard Profiler - DBMS_PROFILER (1)

- Die Variante für keine Rechte auf DB Directories
- ab 8i
  - Hilfstabellen über „$ORACLE_HOME/rdbms/admin/proftab.sql” anlegen
    - Alternativ Script “proftab.sql“ über google suchen
  - Ablauf:

```
execute dbms_profiler.start_profiler;

→ PL/SQL ausführen

execute dbms_profiler.stop_profiler;

→ Auswerten

select runid, run_date, run_comment from
plsql_profiler_runs
```

SQL für HTML-Bericht über My Oracle Support Note Doc ID 243755.1

PL/SQL Compiler Optimizing

- PL/SQL Optimizer optimiert den Byte Code möglichst „intelligent“ vor dem eigentlichen Übersetzen
- Der Level kann eingestellt werden
  - 0 – keine Optimierung
  - 1 – Nur kleine Änderungen
  - 2 – Maximal Mögliche Code Anpassung (Default)
  - 3 (ab Oracle 11g) – In-Lining von lokalen Unterprogrammen + Optimierung vom Level 2

```
SHOW PARAMETER PLSQL_OPTIMIZE_LEVEL


Alter session set PLSQL_OPTIMIZE_LEVEL=3;
```

Wie kann das Ergebnis kondoliert werden?

- Kontrolle nur indirekt möglich, dazu auf die Compiler Warnungen achten

```
ALTER SESSION SET PLSQL_WARNINGS='ENABLE:ALL';
```

- Erzeugt Meldungen wie:

```
LINE/COL  ERROR
--------- -------------------------------------------------------------
..
8/3       PLW-06027: procedure "ADD_NUMBERS" is removed after inlining
21/5      PLW-06005: inlining of call of procedure 'ADD_NUMBERS' was done
21/5      PLW-06004: inlining of call of procedure 'ADD_NUMBERS' requested
```

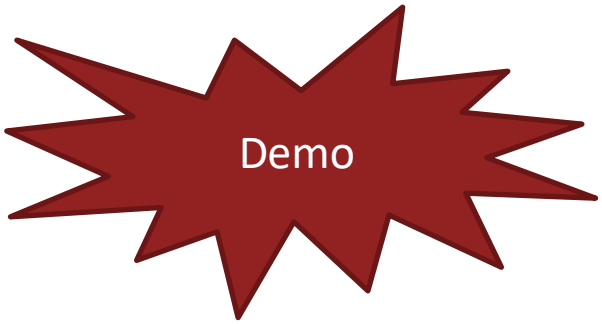
Optimizer Level im Code setzen

- Pragma verwenden, vor Aufruf der Funktion für die Selektive Auswahl des Features

```
PRAGMA INLINE (<name des subprogrammes> , 'YES');
```

YES oder NO

- Interner Code
  kann größer werden!

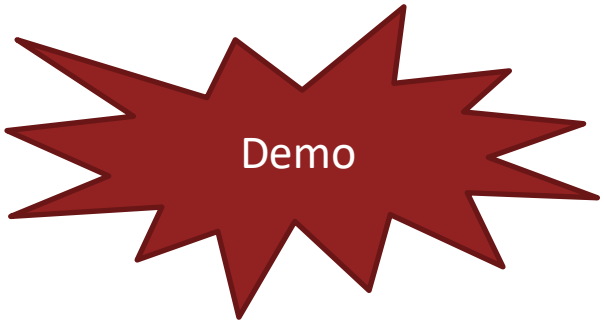


03-Pragma_Settings_inline.sql

Demo von https://oracle-base.com/articles/11g/automatic-subprogram-inlining-11gr1

Risiko bzgl. Compiler Setting auf 3

- Code muss sorgfältig geprüft werden!
- Je nach Code Logik können unerwartete Ergebnisse eintreten!



04.optimizer_code_level_3_demo.sql

Idee Jan Gorkow – Red Stack Magazin 02.2024

PL/SQL - Code Größe

- Wie groß wird das Ganze in der Datenbank?

```
select NAME, TYPE, SOURCE_SIZE, PARSED_SIZE, CODE_SIZE
  from user_object_size
where type like 'P%';
```

NAME	TYPE	SOURCE_SIZE	PARSED_SIZE	CODE_SIZE
BULCK_LOAD_TEST	PROCEDURE	4573	1263	3847
EBA_DEMO_APPR	PACKAGE	4645	4543	2222
EBA_DEMO_APPR	PACKAGE BODY	14474	2489	14482
EBA_DEMO_APPR_DATA	PACKAGE	137	276	213
EBA_DEMO_APPR_DATA	PACKAGE BODY	6968	1267	11414

- PARSED_SIZE - Size (in bytes) of the "flattened" DIANA parsed Code
- SOURCE_SIZE - Size of the source in bytes.
- CODE_SIZE - Size of the PL/SQL Code

Compiler Settings ?

- Auf dem Object abfragen

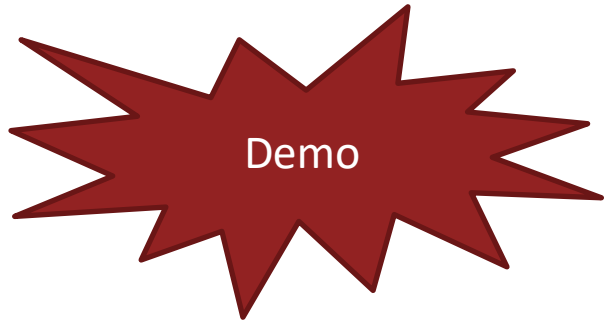
```
select name
        , type
        , plsql_optimize_level
        , plsql_code_type
        , plsql_debug
        , plsql_warnings
        , nls_length_semantics
        , plsql_ccflags
        , plscope_settings
 from user_plsql_object_settings
where name =  'PRAGMARUN';
```

NAME	TYPE	PLSQL_OPTIMIZE_LEVEL	PLSQL_CODE_TYPE	PLSQL_DEBUG	PLSQL_WARNINGS	NLS_LENGTH_SEMANTICS	PLSQL_CCFLAGS	PLSCOPE_SETTINGS
PRAGMARUN	PROCEDURE	2	NATIVE	FALSE	DISABLE:ALL	BYTE	(null)	IDENTIFIERS:ALL

Nativ übersetzen

- PLSQL Code Type auf Nativ stellen

```
ALTER SESSION SET  PLSQL_CODE_TYPE='NATIVE';
```

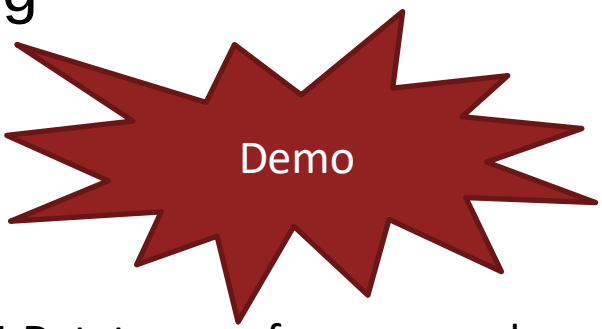


- Bei hohem Anteil numerischer Operationen sinnvoll

03-Pragma_Settings_inline.sql

Spielen die nativen Datentypen noch eine Rolle?

- PLS_INTEGER versus NUMBER
  - PLS_INTEGER wird direkt auf die Hardware abgebildet
  - NUMBER verwendet die Oracle Libraries für Plattform Unabhängigkeit (früher > 32Bit (34?)) Datentyp
  - Genauer als die reine CPU Zahl Darstellung



05-Datatyp_performance.sql

https://oracle-base.com/articles/misc/performance-of-numeric-data-types-in-plsql
https://connor-mcdonald.com/2022/07/08/pl-sql-choosing-the-best-data-type/
https://asktom.oracle.com/ords/f?p=100:11:0::::P11_QUESTION_ID:9538060400346837218

PLS_INTEGER zu NUMBER => Konvertierung => Zeitverlust beachten!

Daten Konvertierung vermeiden

- Datentypen beachten!
- Konvertierung vermeiden

```
Decleare
  v_id number(11,0):=1
  v_cust_id varchar2(11)=1;
Begin
  v_id := ‘100’;
  v_cust_id := 100;
End;
```

Pass By Value - Pass By Reference

```
Procedure myBigCopy(p_wert IN OUT NOCOPY varchar2);
```

- Pass By Value
  IN OUT
  - Default in PLSQL
  - Buffer wird angelegt

- Pass By Reference
  IN OUT NOCOPY
  - NOCOPY Hint
  - Pointer Logik
  - Vorteil
    - Erzeugen eines Buffers (Zeit und Platz!) wird eingespart
  - Nachteil
    - Bei einer Exception kann ein unerwarteter Wert in der referenzierten Variable stehen

Pass By Value - Pass By Reference

```
declare
  v_in_out varchar(32000);
  procedure setValues(
     p_value IN OUT varchar2
  )
  is
  begin
    p_value:='Schritt1';
    p_value:=p_value || 'B' + 1;
  end;
begin
   v_in_out:='WertA';
   begin
     setValues(v_in_out);
   exception
      when others then null;
    end;

DBMS_OUTPUT.put_line('Wert:' || v_in_out);

end;
```

Wert : WertA

```
declare
  v_in_out varchar(32000);
  procedure setValues(
     p_value IN OUT NOCOPY varchar2
  )
  is
  begin
    p_value:='Schritt1';
    p_value:=p_value || 'B' + 1;
  end;
begin
   v_in_out:='WertA';
   begin
     setValues(v_in_out);
   exception
      when others then null;
    end;

DBMS_OUTPUT.put_line('Wert:' || v_in_out);

end;
```

Wert : Schritt1

Pass By Reference - NOCOPY

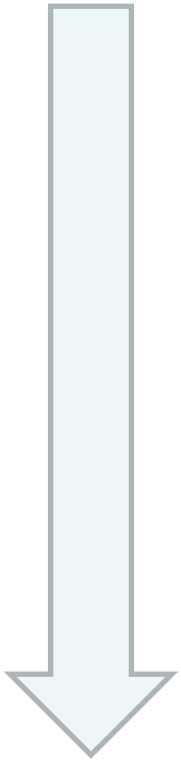
- Bei großen Objekten / Collections interessant

- Exception Handling beachten!

06-NoCopy_ExceptionHandling.sql

Parallelisierung von PL/SQL

- PL/SQL arbeitet im Prinzip immer prozedural ohne Nebenläufigkeit



Was können wir tun? Parallel Starten!

Zum Beispiel das
Oracle Schedule Job Chain Feature verwenden

- Programmatisch Koordination der Prozesse sicherstellen
  - Advanced Queuing (AQ) / Transactional Event Queues (TxEventQ)
  - Oracle Lock Feature mit Wait verwenden

- Package DBMS_PARALLEL_EXECUTE Für DML nützen

Packages im Speicher "pinnen"

- Häufig verwendete Packages "fest" im Speicher halten
- Mit Package "DBMS_SHARED_POOL" pinnen

```
GRANT execute ON dbms_shared_pool TO <APP_SCHEMA>;
```

```
EXECUTE dbms_shared_pool.keep ('STANDARD', 'P');
```

- Kandidaten sind z.B.:
  - STANDARD
  - DBMS_STANDARD
  - DBMS_UTILITY
  - DBMS_DESCRIBE
  - DBMS_OUTPUT

PL/SQL – der Ultimative Performance Tipp

- So wenig wie möglich SQL in PL/SQL verwenden!

- Und wenn SQL => möglichst blockweise verarbeiten

Aufrufe von PL/SQL in SQL vermeiden

- Beispiel
  - Funktion erzeugt nach Regel Kunden Nummer
    - Vorteil: Sauberer Code, eine zentrale Stelle, alles richtig gemacht
    - Nachteil: Wird gelegentlich 1.000.000 aufgerufen und oft wird der Aufruf des SQL‘ s über PL/SQL Methoden durchgeführt

- Problem:
  - Kontext Switch zwischen SQL und PL/SQL Engine

- Lösungen:
  - Hart in SQL codieren (Schwer sauber zu warten)
    - WITH FUNCTION ab 12c
  - Pragma UDF (Kann, muss aber nicht helfen)
  - Result Cache
  - Ab Oracle 21 - SQL Macro / 23c Transpiler Feature

UDF Pragma

- Pragma "UDF" (User Defined Function)
- Pragma gibt dem Compiler die Information, dass die Funktion viel/hauptsächlich in SQL verwendet wird und optimiert Zugriff auf die Funktion

- Idee
  - Reduzierung der Kosten für den PL/SQL => SQL Context Switch Overhead

Function Caching – Result Cache (1)

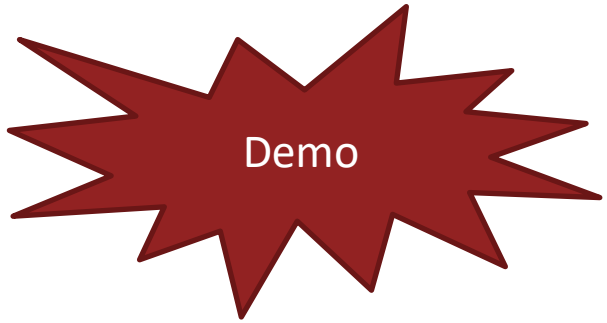
Enterprise Edition!

- Anweisung "RESULT_CACHE"
  - Berechnung wird nicht erneut durchgeführt
  - Ist der Parameter gleich und das abhängige Objekt unverändert wird einfach der Wert aus dem Cache zurückgeben
  - Aber: Overhead beachten, zwischen Aufbau Cache und eigentlicher Programm Logik!
  - Lohnt sich nicht für "einfache Funktionen" mit vielen verschiedenen Parameter Werten im Aufruf!

https://docs.oracle.com/en/database/oracle/oracle-database/19/tgdba/tuning-result-cache.html

Function Caching – Result Cache (2)

- Einrichten "RESULT_CACHE"
  - RESULT_CACHE_MAX_SIZE
  - RESULT_CACHE_MAX_RESULT
  - RESULT_CACHE_REMOTE_EXPIRATION
- Wird es überhaupt verwendet;



07-result_cache_demo.sql

```
select DBMS_RESULT_CACHE.status from dual;

------
ENABLED



SET SERVEROUTPUT ON
EXECUTE DBMS_RESULT_CACHE.MEMORY_REPORT
```

```
R e s u l t   C a c h e   M e m o r y   R e p o r t
[Parameters]
Block Size          = 4K bytes
Maximum Cache Size  = 7936K bytes (1984 blocks)
Maximum Result Size = 196K bytes (49 blocks)
[Memory]
Total Memory = 8396624 bytes [0.878% of the Shared Pool]

... Fixed Memory = 4208 bytes [0.000% of the Shared Pool]
... Dynamic Memory = 8392416 bytes [0.878% of the Shared Pool]
....... Overhead = 265952 bytes
....... Cache Memory = 7936K bytes (1984 blocks)
........... Unused Memory = 813628718 blocks
........... Used Memory = -813626734 blocks
............... Dependencies = 73 blocks (73 count)
............... Results = -813626807 blocks
............... Temp Metadata = 0 blocks
............... Temporary Tablespace Memory = 0 blocks
................... PLSQL    = 1 blocks (1 count)
................... CDB      = 1 blocks (1 count)
................... Invalid = -813626809 blocks (203 count)
```

With Function

- Ab 12c

```
with
  function  getKundenNummer_with(p_kde_id number)
   return varchar2
  as
  begin
    return  case when p_kde_id < 1000
          then lpad(to_char(p_kde_id),4,'0')
           else to_char(p_kde_id)
            end;
  end;
select * from EMPLOYEES where  getKundenNummer_with(EMPLOYEE_ID) = '0106';
/
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(CASE  WHEN "EMPLOYEE_ID"<1000 THEN
              LPAD(TO_CHAR("EMPLOYEE_ID"),4,'0') ELSE TO_CHAR("EMPLOYEE_ID") END
              ='0106')
```

21/23c – das nächste Große Ding

- 21c – SQL Makro
  - SQL-Texte in Pl/SQL Routinen definieren

- 23c -Transpiler
  - PL/SQL automatisch in SQL-Ausdrücke transformieren

https://blogs.oracle.com/coretec/post/sql-transpiler-in-23c

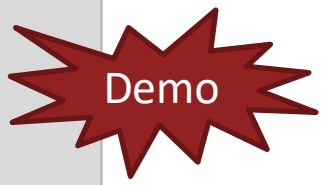
21c – SQL Macro

- SQL Text in PL/SQL definieren und verwenden
  - Template Technik
  - Beim ersten Parsen wird aus einer PL/SQL Routine ein "inline" SQL Code
  - Vorteil:
    SQL-Abfragen, wie die Berechnung einer Kundennummer, können an einer Stelle zentral gepflegt werden
  - Voraussetzung / Nachteil
    - Ausdruck muss vollständig in reinen SQL formulierbar sein!

21c – SQL Macro

- Beispiel:

```
create or replace function getKundenNummer_SM(p_kde_id
number)
return varchar2
SQL_MACRO(SCALAR)
as
begin
  return('case when p_kde_id < 1000
          then lpad(to_char(p_kde_id),4,''0'')
          else to_char(p_kde_id)
           end'
  );
end;
/
```



08-sql_macro.sql

```
select * from EMPLOYEES
where  getKundenNummer_SM(EMPLOYEE_ID) = '0106';
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(CASE  WHEN "EMPLOYEE_ID"<1000 THEN
              LPAD(TO_CHAR("EMPLOYEE_ID"),4,'0') ELSE TO_CHAR("EMPLOYEE_ID") END
              ='0106')
```

23c - Transpiler 23c

- Automatisch PL/SQL in SQL umsetzen
- PL/SQL muss "umsetzbar" sein wie:
  - Basic SQL scalar types: CHARACTER, DATE-TIME, and NUMBER
  - String types (CHAR, VARCHAR, VARCHAR2, NCHAR, etc.)
  - Numeric types (NUMBER, BINARY DOUBLE, etc.)
  - Date types (DATE, TIME, INTERVAL, and TIMESTAMP)
  - Local variables (with optional initialization at declaration) and constants
  - Parameters with optional (simple) default values
  - Variable assignment statements
  - Expressions which can be translated into equivalent SQL expressions
  - IF-THEN-ELSE statements
  - RETURN statements
  - Expressions and local variables of BOOLEAN type
- Aktuell nur "function"?

23c - Transpiler 23c

- Einstellung prüfen und aktivieren

```
Show parameter sql_transpiler

alter session set sql_transpiler=on;
```

- "normale" PL/SQL Routine anlegen

```
create or replace function add_number(p1 in number, p2 in number)
return number as
begin
  return p1 + p2;
end;
/
```

```
select * from EMPLOYEES where  add_number(EMPLOYEE_ID,1) = 106;
```

- Wird automatisch in SQL Code umgeschrieben!

09-23c_transpiler.sql

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("EMPLOYEE_ID"+1=106)
```

SQL Macro ⇔ Transpiler 23c

- **SQL Maco**
  - PL/SQL Funktion vom Typ SQL_MACRO
  - Muss für die Query, die es verwendet optimiert und angepasst werden

- **PL/SQL to SQL Transpiler**
  - Umwandeln von PL/SQL Funktionen in SQL-Ausdrücke
  - Sehr effizient für "einfache" PL/SQL Funktionen, ohne wieder eigene SQL-Logik

https://blogs.oracle.com/coretec/post/sql-transpiler-in-23c

Aufrufe von SQL in PL/SQL zusammenfassen

- SQL-Statements „Bulk“ am Stück ohne Context Switch verarbeiten
- Stichwort BULK COLLECT und FORALL

Oracle Sample Schemas https://github.com/oracle-samples/db-sample-schemas/

Demo mit https://www.pipperr.de/dokuwiki/doku.php?id=prog:plsql_bulk_collect

Fazit

- Wo immer möglich Native SQL einsetzen
  - BULK COLLECT Feature verwenden
- PL/SQL Funktionsaufrufe in SQL vermeiden
  - SQL Macro ab 21c verwenden
  - With Function in SQL verwenden

Quellen

- Suchen Sie nach Steven Feuerstein
  - wie https://www.youtube.com/watch?v=EoyRxPxU26U
- https://oracle-internals.com/blog/2020/04/29/a-not-so-brief-but-very-accurate-history-of-pl-sql/
- https://www.oracle.com/technical-resources/articles/schumacher-analysis.html
- https://oracle-base.com/articles/11g/plsql-hierarchical-profiler-11gr1
- RedStack Magazin 02.2024
- https://oracle-base.com/articles/23c/automatic-plsql-to-sql-transpiler-23c
- https://danischnider.wordpress.com/2022/05/15/performance-tips-pl-sql-functions-in-sql-queries/

Mehr

- Blog Gunther Pippèrr
  - https://www.pipperr.de/dokuwiki/

- Bildmaterial : https://pixabay.com

FAQ

Berta 2024

F&A

Fragen

PL/SQL Performance