Hatékony PL/SQL programok írása

Hatékony PL/SQL programok írása

- a fordító optimalizációs beállításai
- a feltételes fordítás
- fordító figyelmeztetései
- a natív nyelvű fordítás
- néhány teljesítményhangolási tanács

A fordító beállításai

- A fordító viselkedését inicializációs paraméterek szabályozzák. Három szinten lehet beállítani:
  - Oracle példány: ALTER SYSTEM SET…
  - Munkamenet szinten (a példány szintűt felüldefiniálja): ALTER SESSION SET…
  - Az egyes programegységek fordításakor is megadhatók (legerősebb mód)(kiv. néhány): ALTER… COMPILE

A fordító beállításai

- Inicializációs paraméterek
  - PLSQL_OPTIMIZE_LEVEL
  - PLSQL_CODE_TYPE
  - PLSQL_DEBUG
  - PLSQL_WARNINGS
  - PLSQL_CCFLAGS
  - NLS_LENGTH_SEMANTICS
- Információ az érvényben lévő beállításokról a már lefordított programegységek esetén:
  - DBA_PLSQL_OBJECT_SETTINGS
  - ALL_PLSQL_OBJECT_SETTINGS
  - USER_PLSQL_OBJECT_SETTINGS

A fordító beállításai

- ALTER … COMPILE … REUSE SETTINGS:
  - Egy újrafordítás során a korábbi beállítások megtartja az aktuális munkamenet- és példányszintű beállításoktól függetlenül.

```
ALTER PROCEDURE proc
    COMPILE
    PLSQL_OPTIMIZE_LEVEL=0
    PLSQL_DEBUG=FALSE
    REUSE SETTINGS;
```

- Az ALTER … COMPILE utasítás DEBUG utasításrészének hatása megegyezik a PLSQL_DEBUG=TRUE beállításával.

Az optimalizáló fordító

- A PL/SQL kódot az optimalizáló
  - gyorsabbá tudja tenni, optimalizálni tudja,
  - a kódot részben átstrukturálja,
  - a számításokhoz minél hatékonyabb módot választ
  - anélkül, hogy a kód értelmét megváltoztatja
  - garantálja, hogy nem hagy ki explicit alprogramhívást
  - garantálja, hogy nem ír elő új alprogram bevezetését
  - garantálja, hogy nem vált ki olyan kivételt, ami eredetileg nem váltódna ki
  - előfordulhat, hogy az optimalizált kód nem vált ki kivételt ott, ahol az optimalizálatlan kód kiváltana.

Az optimalizáló fordító

- PLSQL_OPTIMIZE_LEVEL inicializációs paraméter:
  - az optimalizálás mértékét szabályozza
  - értékei:
    - 2: teljes körű optimalizálás, alapértelmezett érték
    - 1: lokális optimalizálás (számítások, kifejezések optimalizálása)
    - 0: az optimalizáló kikapcsolása

Az optimalizáló fordító

- Optimalizáló műveletek:
  1. *Matematikai számítások optimalizálása*
     a) Fordítási időben kiértékelhető kifejezések kiértékelése
     b) Egy kevésbé hatékony művelet helyett egy hatékonyabb használata
     c) Kereszteredmények számítása
     d) A precedencia által nem kötött műveletek kiértékelési sorrendjének tetszőleges megválasztása

Az optimalizáló fordító

- Optimalizáló műveletek:
  2. *Ciklus magjából a ciklusváltozótól független számítás kiemelése*

```
CREATE OR REPLACE PROCEDURE proc_szamitas(p_Iter PLS_INTEGER) IS
 a PLS_INTEGER; b PLS_INTEGER; c PLS_INTEGER; d PLS_INTEGER;
BEGIN
  FOR i IN 1..p_Iter LOOP
    a := i+1; b := i-2;
    c := b-a+1; d := b-a-1;   -- ismétlődő kifejezés előfordulása
  END LOOP;
END proc_szamitas;
/
SHOW ERRORS;

-- Fordítás optimalizálással, majd futtatás
ALTER PROCEDURE proc_szamitas COMPILE PLSQL_OPTIMIZE_LEVEL=2
  PLSQL_DEBUG=FALSE;
SET TIMING ON EXEC proc_szamitas(10000000); SET TIMING OFF;
-- Eltelt: 00:00:03.06

-- Fordítás optimalizálás nélkül, majd futtatás
ALTER PROCEDURE proc_szamitas COMPILE PLSQL_OPTIMIZE_LEVEL=0
  PLSQL_DEBUG=FALSE;
SET TIMING ON EXEC proc_szamitas(10000000); SET TIMING OFF;
-- Eltelt: 00:00:05.54
```

```
 …
  FOR i IN 1..p_Iter
  LOOP
    a := 3+1;
    b := 3-2;
    c := b-a+1;
    d := b-a-1;   -- ismétlődő kifejezés előfordulása
  END LOOP;
…

-- Fordítás optimalizálással, majd futtatás
-- Eltelt: 00:00:00.46

-- Fordítás optimalizálás nélkül, majd futtatás
-- Eltelt: 00:00:05.53
```

```
…
BEGIN
  a := 3+1;
  b := 3-2;
  c := b-a+1;
  d := b-a-1;   -- ismétlődő kifejezés előfordulása
  FOR i IN 1..p_Iter
  LOOP
    NULL; -- ez maradt a ciklusmagból
  END LOOP;
END proc_szamitas;
…

-- Fordítás optimalizálással, majd futtatás
-- Eltelt: 00:00:00.45

-- Fordítás optimalizálás nélkül, majd futtatás
-- Eltelt: 00:00:00.45
```

Az optimalizáló fordító

- Optimalizáló műveletek:
  3. *A CONSTANT kulcsszó figyelembevétele*

```
CREATE OR REPLACE PROCEDURE proc_konstans(p_Iter  PLS_INTEGER) IS
  c_Konstans   CONSTANT NUMBER := 98765;
  v_Konstans            NUMBER := 98765;
  v1                    NUMBER := 1;
  v2                    NUMBER;
  -- Tesztelést segítő változók
  v_Ures_ciklus_ideje   NUMBER; t                     NUMBER;
  t1                    NUMBER; t2                    NUMBER;
  -- Eljárás esetleges mellékhatással
  PROCEDURE proc_lehet_mellekhatasa IS BEGIN NULL; END;
  -- Tesztelést segítő eljárások
  PROCEDURE cimke(p_Cimke VARCHAR2) IS
  BEGIN DBMS_OUTPUT.PUT(RPAD(p_Cimke, 20)); END cimke;

  PROCEDURE eltelt(p_Ures_ciklus  BOOLEAN DEFAULT FALSE) IS
  BEGIN
    t := t2-t1;
    IF p_Ures_ciklus THEN v_Ures_ciklus_ideje := t; END IF;
    DBMS_OUTPUT.PUT_LINE('-  eltelt: ' || LPAD(t, 5)
        || ',  ciklusidő nélkül:' || LPAD((t-v_Ures_ciklus_ideje), 5));
  END eltelt;
```

```
-- Az eljárás törzse
BEGIN
  cimke('Üres ciklus'); t1 := DBMS_UTILITY.GET_TIME;
  FOR i IN 1..p_Iter LOOP proc_lehet_mellekhatasa; END LOOP;
  t2 := DBMS_UTILITY.GET_TIME;
  eltelt(p_Ures_ciklus => TRUE);

  cimke('Változó használata'); t1 := DBMS_UTILITY.GET_TIME;
  FOR i IN 1..p_Iter LOOP
    proc_lehet_mellekhatasa;
    v2 := v1 + v_Konstans * 12345;
  END LOOP;
  t2 := DBMS_UTILITY.GET_TIME; eltelt;

  cimke('Konstans használata'); t1 := DBMS_UTILITY.GET_TIME;
  FOR i IN 1..p_Iter LOOP
    proc_lehet_mellekhatasa;
    v2 := v1 + c_Konstans * 12345;
  END LOOP;
  t2 := DBMS_UTILITY.GET_TIME; eltelt;
END proc_konstans;
/
SHOW ERRORS;
```

```
PROMPT 1. PLSQL_OPTIMIZE_LEVEL=2 -- Fordítás optimalizálással, majd futtatás
ALTER PROCEDURE proc_konstans COMPILE PLSQL_OPTIMIZE_LEVEL=2
   PLSQL_DEBUG=FALSE;
EXEC proc_konstans(2000000);

PROMPT 2. PLSQL_OPTIMIZE_LEVEL=0 -- Fordítás optimalizálás nélkül, és futtatás
ALTER PROCEDURE proc_konstans COMPILE PLSQL_OPTIMIZE_LEVEL=0
   PLSQL_DEBUG=FALSE;
EXEC proc_konstans(2000000);

/* Egy tipikusnak mondható kimenet:
1. PLSQL_OPTIMIZE_LEVEL=2
Az eljárás módosítva.
Üres ciklus          -  eltelt:    78,  ciklusidő nélkül:    0
Változó használata   -  eltelt:   186,  ciklusidő nélkül:  108
Konstans használata  -  eltelt:   117,  ciklusidő nélkül:   39
A PL/SQL eljárás sikeresen befejeződött.

2. PLSQL_OPTIMIZE_LEVEL=0
Az eljárás módosítva.
Üres ciklus          -  eltelt:    79,  ciklusidő nélkül:    0
Változó használata   -  eltelt:   246,  ciklusidő nélkül:  167
Konstans használata  -  eltelt:   245,  ciklusidő nélkül:  166
A PL/SQL eljárás sikeresen befejeződött.*/
```

Az optimalizáló fordító

- Optimalizáló műveletek:
  4. *Csomaginicializálás késleltetése*

```
-- Csomag inicializálással és az inicializálástól nem függő
   tagokkal
CREATE OR REPLACE PACKAGE csomag_inittel IS
  -- Néhány csomagbeli elem
  c_Konstans CONSTANT  NUMBER := 10;
  v_Valtozo            NUMBER := 10;
  TYPE t_rec IS RECORD (id NUMBER, nev VARCHAR2(10000));
END csomag_inittel;
/

CREATE OR REPLACE PACKAGE BODY csomag_inittel IS
-- Csomaginicializáló blokk
BEGIN
  DBMS_OUTPUT.PUT_LINE('Csomaginicializáló blokk.');
  -- Egy kis várakozás, sokat dolgozik ez a kód...
  DBMS_LOCK.SLEEP(10);
  -- A DBMS_LOCK csomag a SYS sémában van,
  -- használatához EXECUTE jog szükséges.
END csomag_inittel;
/
```

```
-- Csomagot hivatkozó eljárás
CREATE OR REPLACE PROCEDURE proc_csomag_inittel IS
  -- Csomagbeli típus hivatkozása
  v_Rec  csomag_inittel.t_rec;
  v2     csomag_inittel.v_Valtozo%type;
BEGIN
  -- Csomagbeli konstans hivatkozása
DBMS_OUTPUT.PUT_LINE('Csomag konstansa: ' || csomag_inittel.c_Konstans);
END proc_csomag_inittel;
/
-- Tesztek
-- Új munkamenet nyitása, hogy a csomag ne legyen inicializálva
CONNECT plsql/plsql
SET SERVEROUTPUT ON FORMAT WRAPPED;
-- Fordítás optimalizálással, majd futtatás
ALTER PROCEDURE proc_csomag_inittel COMPILE PLSQL_OPTIMIZE_LEVEL=2
  PLSQL_DEBUG=FALSE;
SET TIMING ON
EXEC proc_csomag_inittel; SET TIMING OFF;
 -- Eltelt: 00:00:00.01
```

```
-- Új munkamenet nyitása, hogy a csomag ne legyen inicializálva
CONNECT plsql/plsql
SET SERVEROUTPUT ON FORMAT WRAPPED;
-- Fordítás optimalizálás nélkül, majd futtatás
ALTER PROCEDURE proc_csomag_inittel COMPILE PLSQL_OPTIMIZE_LEVEL=0
  PLSQL_DEBUG=FALSE;

SET TIMING ON
EXEC proc_csomag_inittel;
-- Eltelt: 00:00:10.01
SET TIMING OFF;
```

Az optimalizáló fordító

- Optimalizáló műveletek:
  5. *Indexet tartalmazó kifejezések indexelésének gyorsítása*

```
CREATE OR REPLACE PROCEDURE
 proc_indexes_kifejezes(p_Iter  PLS_INTEGER) IS
  TYPE t_tab IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
  v_Tab         t_tab;
  v_Index       NUMBER;
  v_Dummy       NUMBER;
   -- Tesztelést segítő változók
  t                     NUMBER;
  t1                    NUMBER;
  t2                    NUMBER;
  v_Ures_ciklus_ideje   NUMBER;
  -- Tesztelést segítő eljárások
  PROCEDURE cimke(p_Cimke VARCHAR2) IS
  BEGIN DBMS_OUTPUT.PUT(RPAD(p_Cimke, 25)); END cimke;

  PROCEDURE eltelt(p_Ures_ciklus  BOOLEAN DEFAULT FALSE) IS
  BEGIN
    t := t2-t1;
    IF p_Ures_ciklus THEN v_Ures_ciklus_ideje := t;
    END IF;
    DBMS_OUTPUT.PUT_LINE('-  eltelt: ' || LPAD(t, 5)
        || ',  ciklusidő nélkül:' || LPAD((t-v_Ures_ciklus_ideje), 5));
  END eltelt;
```

```
-- Az eljárás törzse
BEGIN
  cimke('Üres ciklus értékadással'); t1 := DBMS_UTILITY.GET_TIME;
  FOR i IN 1..p_Iter LOOP
    v_Index := i-i; -- Egy értékadás
  END LOOP;
  t2 := DBMS_UTILITY.GET_TIME;  eltelt(p_Ures_ciklus => true);

  cimke('Változó index'); t1 := DBMS_UTILITY.GET_TIME;
  FOR i IN 1..p_Iter LOOP
    v_Index := i-i; -- Egy értékadás
    v_Tab(v_Index) := 10;
  END LOOP;
  t2 := DBMS_UTILITY.GET_TIME; eltelt;

  cimke('Változatlan index');  t1 := DBMS_UTILITY.GET_TIME;
  v_Index := 0;
  FOR i IN 1..p_Iter LOOP
    v_Dummy := i-i; -- Egy értékadás, hogy ugyanannyi kód legyen.
    v_Tab(v_Index) := 10;
  END LOOP;
  t2 := DBMS_UTILITY.GET_TIME; eltelt;
 END proc_indexes_kifejezes;
/
```

```
SET SERVEROUTPUT ON FORMAT WRAPPED;

PROMPT 1. PLSQL_OPTIMIZE_LEVEL=2
-- Fordítás optimalizálással, majd futtatás
ALTER PROCEDURE proc_indexes_kifejezes COMPILE PLSQL_OPTIMIZE_LEVEL=2
  PLSQL_DEBUG=FALSE;
EXEC proc_indexes_kifejezes(2000000);


PROMPT 2. PLSQL_OPTIMIZE_LEVEL=0
-- Fordítás optimalizálás nélkül, majd futtatás
ALTER PROCEDURE proc_indexes_kifejezes COMPILE PLSQL_OPTIMIZE_LEVEL=0
  PLSQL_DEBUG=FALSE;
EXEC proc_indexes_kifejezes(2000000);
```

```
/*Egy tipikusnak mondható kimenet:

...
1. PLSQL_OPTIMIZE_LEVEL=2
Az eljárás módosítva.

Üres ciklus értékadással -  eltelt:    40,  ciklusidő nélkül:    0
Változó index            -  eltelt:   110,  ciklusidő nélkül:   70
Változatlan index        -  eltelt:    83,  ciklusidő nélkül:   43

A PL/SQL eljárás sikeresen befejeződött.

2. PLSQL_OPTIMIZE_LEVEL=0

Az eljárás módosítva.

Üres ciklus értékadással -  eltelt:    50,  ciklusidő nélkül:    0
Változó index            -  eltelt:   146,  ciklusidő nélkül:   96
Változatlan index        -  eltelt:   148,  ciklusidő nélkül:   98

A PL/SQL eljárás sikeresen befejeződött.
*/
```

Az optimalizáló fordító

- Optimalizáló műveletek:
  6. *Statikus kurzor FOR ciklusok együttes hozzárendeléssel történő helyettesítése*

```
CREATE OR REPLACE PROCEDURE proc_ciklus_bulk IS
  TYPE t_num_tab  IS TABLE OF NUMBER;
  TYPE t_char_tab IS TABLE OF VARCHAR2(100);
  v_Num_tab       t_num_tab;
  v_Char_tab      t_char_tab;
  cur             SYS_REFCURSOR;
   -- Tesztelést segítő változók
  t1                  NUMBER; t2                      NUMBER;
  lio1                NUMBER; lio2                    NUMBER;
  v_Session_tag       VARCHAR2(100);

  -- Tesztelést segítő eljárások
  PROCEDURE cimke(p_Cimke VARCHAR2) IS
  BEGIN DBMS_OUTPUT.PUT(RPAD(p_Cimke, 20)); END cimke;

  PROCEDURE tag_session IS
  BEGIN
    v_Session_tag := 'teszt-' || SYSTIMESTAMP;
    DBMS_APPLICATION_INFO.SET_CLIENT_INFO(v_Session_tag);
  END tag_session;
```

```
FUNCTION get_logical_io RETURN NUMBER IS
  rv NUMBER;
BEGIN
   SELECT st.value
    INTO rv
    FROM v$sesstat st, v$session se, v$statname n
   WHERE n.name = 'consistent gets'
     AND se.client_info = v_Session_tag
     AND st.sid = se.sid
     AND n.statistic# = st.statistic#;
  RETURN rv;
END get_logical_io;

PROCEDURE eltelt IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('- eltelt: ' || LPAD(t2-t1, 5)
      || ',  LIO (logical I/O) :' || LPAD((lio2-lio1), 7));
END eltelt;
```

```
BEGIN -- Az eljárás törzse
  tag_session;
cimke('Statikus kurzor FOR'); t1:= DBMS_UTILITY.GET_TIME;
  lio1 := get_logical_io;
  FOR i IN ( SELECT k1.id, k1.cim
             FROM konyv k1, konyv, konyv, konyv, konyv, konyv
             WHERE ROWNUM <= 100000) LOOP NULL; END LOOP;
  t2   := DBMS_UTILITY.GET_TIME; lio2 := get_logical_io; eltelt;
cimke('Kurzor BULK FETCH'); t1:= DBMS_UTILITY.GET_TIME;
  lio1 := get_logical_io;
  OPEN cur FOR SELECT k1.id, k1.cim
               FROM konyv k1, konyv, konyv, konyv, konyv, konyv
               WHERE ROWNUM <= 100000;
  LOOP
    FETCH cur BULK COLLECT INTO v_Num_tab, v_Char_tab LIMIT 100;
    EXIT WHEN v_Num_tab.COUNT = 0;
    FOR i IN 1..v_Num_tab.COUNT LOOP NULL; END LOOP;
  END LOOP;
  CLOSE cur;
  t2   := DBMS_UTILITY.GET_TIME; lio2 := get_logical_io; eltelt;
EXCEPTION WHEN OTHERS
 THEN IF cur%ISOPEN THEN CLOSE cur; END IF; RAISE;
END proc_ciklus_bulk;
/
```

```
SET SERVEROUTPUT ON FORMAT WRAPPED;
-- Fordítás optimalizálással, majd futtatás

PROMPT 1. PLSQL_OPTIMIZE_LEVEL=2
ALTER PROCEDURE proc_ciklus_bulk COMPILE PLSQL_OPTIMIZE_LEVEL=2
  PLSQL_DEBUG=FALSE;
EXEC proc_ciklus_bulk;

PROMPT 2. PLSQL_OPTIMIZE_LEVEL=0
-- Fordítás optimalizálás nélkül, majd futtatás
ALTER PROCEDURE proc_ciklus_bulk COMPILE PLSQL_OPTIMIZE_LEVEL=0
  PLSQL_DEBUG=FALSE;
EXEC proc_ciklus_bulk;
```

```
/*
  Egy tipikusnak mondható kimenet:

...
1. PLSQL_OPTIMIZE_LEVEL=2
Az eljárás módosítva.

Statikus kurzor FOR - eltelt:   100,  LIO (logical I/O) :  34336
Kurzor BULK FETCH   - eltelt:    81,  LIO (logical I/O) :  34336

A PL/SQL eljárás sikeresen befejeződött.


2. PLSQL_OPTIMIZE_LEVEL=0
Az eljárás módosítva.

Statikus kurzor FOR - eltelt:   636,  LIO (logical I/O) : 133336
Kurzor BULK FETCH   - eltelt:    80,  LIO (logical I/O) :  34336

A PL/SQL eljárás sikeresen befejeződött.

*/
```

Automatic Subprogram Inlining (11g)

- Minden alprogramhívásnak költsége van, pl. ha cikluson belül hívjuk meg.
- Csökkenti az alprogramhívás költségét, azzal, hogy az alprogram kódját bemásolja a ciklusba
- Kezelése: PLSQL_OPTIMIZE_LEVEL=3: esetén automatikusan bemásolódhat a kód
- INLINE pragmával explicit módon ki és be lehet kapcsolni

```
ALTER SESSION SET PLSQL_OPTIMIZE_LEVEL=2;
SET SERVEROUTPUT ON
DECLARE
l_loops NUMBER := 10000000;
l_start NUMBER;
l_return NUMBER;
FUNCTION add_numbers (p_1 IN NUMBER, p_2 IN NUMBER) RETURN
   NUMBER AS
BEGIN
RETURN p_1 + p_2;
END add_numbers;
BEGIN
 l_start := DBMS_UTILITY.get_time;
 FOR i IN 1 .. l_loops
   LOOP
   --PRAGMA INLINE (add_numbers, 'YES');
   l_return := add_numbers(1, i);
   END LOOP;
DBMS_OUTPUT.put_line('Elapsed Time: ' ||
   (DBMS_UTILITY.get_time - l_start) || ' hsecs');
END;
/
Elapsed Time: 509 hsecs PL/SQL procedure successfully
   completed.
```

Feltételes fordítás

- Feltételes fordítás során a tényleges fordítást előfordítás előzi meg.
- Az előfordító a fordításkor fennálló körülményektől függően a megadott kód szövegéből egy előfordítói direktívákat már nem tartalmazó kódot állít elő, ez kerül azután lefordításra. A direktívák kis- és nagybetűre nem érzékenyek.
- Használatának tipikus esetei:
  - Egy általános rutin megírásakor fel tudunk készülni arra, hogy a kód különböző adatbázis-kezelő verziókban is ki tudja használni a nyelv új eszközeit az adott környezetben rendelkezésre álló verziótól függően.
  - Nyomkövetési kódot helyezhetünk el a programban, amit a feltételes fordítás segítségével csak fejlesztői környezetben használunk.

Feltételes fordítás

Az előfordítói szerkezetekben használható kifejezések speciálisan csak fordítási időben kiértékelhető BOOLEAN, PLS_INTEGER és VARCHAR2 típusú kifejezések lehetnek. Jelölésük: rendre a *bkf*, *pkf*, *vkf.*

```
bkf:    {TRUE | FALSE | NULL |
    bkf hasonlító_operátor bkf | pkf hasonlító_operátor pkf |
    NOT bkf | bkf AND bkf | bkf OR bkf | {bkf|pkf|vkf} IS [NOT] NULL |
    csomagbeli_boolean_statikus_nevesített_konstans |
      boolean_értékdirektíva}

pkf:    {pls_integer_literál | NULL |
    csomagbeli_pls_integer_statikus_nevesített_konstans |
      pls_integer_értékdirektíva}

vkf:    {char_literál | NULL |
    TO_CHAR(pkf) | TO_CHAR(pkf, vkf, vkf) |
    {pkf|vkf} || {pkf|vkf} |  varchar2_értékdirektíva}
```

Feltételes fordítás

- Nevesített konstans ($ jel nélküli hivatkozás)
- Értékdirektívák: $$azonosító; PLSQL_CCFLAGS paraméterből kapnak értéket
- Elágaztató direktíva:

```
$IF bkf $THEN kód
    [ $ELSIF bkf $THEN kód ]…
    [ $ELSE kód ]
$END
```

- Hibadirektíva: `$ERROR vkf $END`

```
ALTER SESSION SET PLSQL_CCFLAGS='debug:2';
CREATE OR REPLACE PROCEDURE proc_preproc IS
BEGIN
  $IF $$debug = 1 $THEN
    DBMS_OUTPUT.PUT_LINE('Van DEBUG');
  $ELSIF $$debug = 0 $THEN
    DBMS_OUTPUT.PUT_LINE('Nincs DEBUG');
  $ELSIF $$debug IS NOT NULL $THEN
    $ERROR 'Érvénytelen DEBUG beállítás: ' ||
        'PLSQL_CCFLGAS=' || $$PLSQL_CCFLAGS $END
  $END
  DBMS_OUTPUT.PUT_LINE('Kód');
END proc_preproc;
/
SHOW ERRORS;
```

```
ALTER PROCEDURE proc_preproc COMPILE
  PLSQL_CCFLAGS='debug:1';
SET SERVEROUTPUT ON FORMAT WRAPPED;
BEGIN
  DBMS_PREPROCESSOR.PRINT_POST_PROCESSED_SOURCE(
      object_type => 'PROCEDURE'
    , schema_name => 'PLSQL'
    , object_name => 'PROC_PREPROC');
END;
/
/*
Az eljárás módosítva.
PROCEDURE proc_preproc IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Van DEBUG');
  DBMS_OUTPUT.PUT_LINE('Kód');
END proc_preproc;
A PL/SQL eljárás sikeresen befejeződött.
*/
```

```
ALTER SESSION SET PLSQL_CCFLAGS='konstansom_erteke:TRUE';
CREATE OR REPLACE package pack_konstansok IS
  c_Konstans CONSTANT BOOLEAN := $$konstansom_erteke;
END;
/
CREATE OR REPLACE PROCEDURE proc_preproc_pack IS
BEGIN
  $IF pack_konstansok.c_Konstans $THEN
  DBMS_OUTPUT.PUT_LINE('pack_konstansok.c_konstans TRUE');
  $ELSE
 DBMS_OUTPUT.PUT_LINE('pack_konstansok.c_konstans FALSE');
  $END
END proc_preproc_pack;
/

SET SERVEROUTPUT ON FORMAT WRAPPED;
EXEC proc_preproc_pack;
-- pack_konstansok.c_konstans TRUE
-- A csomag explicit újrafordítása az eljárást érvényteleníti
ALTER PACKAGE pack_konstansok COMPILE
   PLSQL_CCFLAGS='konstansom_erteke:FALSE';
-- Most fog az eljárás újrafordulni automatikusan
EXEC proc_preproc_pack; -- pack_konstansok.c_konstans FALSE
```

Feltételes fordítás

Az előfordítói direktívák használatára vonatkozóan a következő szabályok érvényesek:

- Nem befolyásolhatják tárolt adatbázistípusok szerkezetét, ezért nem használhatók objektumtípus specifikációjában és sémában tárolt kollekciótípusok megadásakor.
- Használhatók a paraméter nélküli programegységekben az IS/AS kulcsszó után, kivéve a triggereket, a névtelen blokkokat és az előző pontbeli objektumtípus specifikációt.
- Használhatók a paraméterrel rendelkező tárolt függvényben és eljárásban a formális paraméterlistát nyitó zárójel után.
- Használhatók triggerekben és névtelen blokkokban közvetlenül a kezdő DECLARE vagy BEGIN után.
- Környezeti gazdaváltozó helykijelölője nem szerepelhet elágaztató direktíván belül. Ez névtelen blokkban fordulhatna elő.

A fordító figyelmeztetései

- Figyelmeztetéseket a fordítás során kaphatunk.
- A kódban található nem biztonságos, nem konzisztens kódrészlet előfordulását jelzik.
- A figyelmeztetések tartalmaznak egy hibakódot PLW-XXXXX alakban
- 3 csoport: SEVERE, PERFORMANCE, INFORMATIONAL
- PLSQL_WARNINGS paraméter: DISABLE, ENABLE, ERROR
- Információ:
  - {DBA|ALL|USER}_ERRORS
  - SQL*Plus SHOW ERRORES
- DBMS_WARNING CSOMAG

```
ALTER SESSION SET PLSQL_WARNINGS='DISABLE:ALL'
  PLSQL_CCFLAGS='';
-- Csúnya, de hibátlan alprogram
CREATE OR REPLACE PROCEDURE proc_warn IS
  v_Id    VARCHAR2(100); v_Name  VARCHAR2(100);
  to_char BOOLEAN;  -- PLW-05004, megengedett, mert
                    -- a TO_CHAR nem fenntartott szó
BEGIN
  $IF $$kojak $THEN $END
    -- PLW-06003, kojak nincs a PLSQL_CCFLAGS-ben
  SELECT cim INTO v_Name
    FROM konyv WHERE id = v_Id;
-- PLW-07204, id NUMBER, v_id VARCHAR2
  IF FALSE THEN NULL;
-- PLW-06002, az IF feltétele mindig hamis
  END IF;
END proc_warn;
/
SHOW ERRORS;
```

```
ALTER PROCEDURE proc_warn COMPILE
  PLSQL_WARNINGS='DISABLE:ALL', 'ENABLE:SEVERE';
SHOW ERRORS;

/*
SP2-0805: Az eljárás fordítási figyelmeztetésekkel lett
   módosítva.

Hibák PROCEDURE PROC_WARN:

LINE/COL ERROR
-------- -----------------------------------------------
   ----------------
5/3      PLW-05004: a(z) TO_CHAR azonosító a STANDARD
   csomagban is
         definiálva van vagy beépített SQL-elem.
*/
```

```
ALTER PROCEDURE proc_warn COMPILE
  PLSQL_WARNINGS='DISABLE:ALL', 'ENABLE:PERFORMANCE';
SHOW ERRORS;
/*
SP2-0805: Az eljárás fordítási figyelmeztetésekkel lett
   módosítva.

Hibák PROCEDURE PROC_WARN:

LINE/COL ERROR
-------- -----------------------------------------------
   ----------------
11/15    PLW-07204: az oszlop típusától eltérő konverzió
   optimum alatti
         lekérdezési szerkezetet eredményezhet
*/
```

```
ALTER PROCEDURE proc_warn COMPILE
  PLSQL_WARNINGS='DISABLE:ALL', 'ENABLE:INFORMATIONAL';
SHOW ERRORS;
/*
SP2-0805: Az eljárás fordítási figyelmeztetésekkel lett
   módosítva.

Hibák PROCEDURE PROC_WARN:

LINE/COL ERROR
-------- -----------------------------------------------
   ----------------
7/7      PLW-06003: ismeretlen lekérdezési direktíva:
   '$$KOJAK'
13/6     PLW-06002: Nem elérhető kód
*/
```

```
ALTER PROCEDURE proc_warn COMPILE
  PLSQL_WARNINGS='ERROR:ALL';
SHOW ERRORS;
/*
Figyelmeztetés: Az eljárás módosítása fordítási hibákkal
   fejeződött be.

Hibák PROCEDURE PROC_WARN:

LINE/COL ERROR
-------- -----------------------------------------------
   ----------------
7/7      PLS-06003: ismeretlen lekérdezési direktíva:
   '$$KOJAK'
*/
```

Natív fordítás

- Natív fordítás során a p-kódból a fordító egy olyan C nyelvű forrásszöveget hoz létre, amely közvetlenül tartalmazza az utasításoknak megfelelő API hívásokat.
- Ezt a C forrást aztán a rendszerben rendelkezésre álló C fordítóval fordítja be egy osztott könyvtárba
- A programegység végrehajtásakor elég az osztott könyvtárat használni.
- A kód értelmezésének ideje csökken
- Nagy számításigényű kódoknál teljesítménynövekedés
- PLSQL_CODE_TYPE inicializációs paraméterrel szabályozzuk: értéke INTERPRETED vagy NATIVE lehet

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

Főbb hangolási alapelvek:

- Mérjünk. Mérjünk hangolás előtt, hogy legyen referenciánk.
- Tűzzük ki a célt, amivel elégedettek leszünk.
- Keressük meg a legszűkebb keresztmetszetet. A *Pareto-elv*, másik nevén a *80/20 szabály*
- Egyszerre egy módosítást végezzünk.
- A módosítás hatását jósoljuk meg, állítsunk fel hipotézist.
- Mérjünk.
- A fejlesztés során minél később gondolunk a teljesítményre, annál költségesebb a hangolás.
- Ha a fejlesztés során túl korán kezdünk el a teljesítménnyel foglalkozni az a tervezést félreviheti.
- A méréshez és a szűk keresztmetszet kereséséhez használhatjuk:
  - a legegyszerűbb eszközöket: SET TIMING; DBMS_UTILITY.GET_TIME
  - komolyabb eszközöket: DBMS_PROFILER, DBMS_TRACE,
  - a dinamikus teljesítménynézetek ($SESSTAT, V$SYSSTAT stb.)
  - STATSPACK programcsomag vagy az SQL trace eszközrendszer.

Tanácsok hangolási tanácsok megfogadásához:

- Egészséges kételkedés
- Bizonyosság

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

1. Hangoljuk a kódba ágyazott SQL-utasításokat.
2. Használjuk a RETURNING utasításrészt DML utasításoknál a sor adatainak visszaolvasására.
3. Használjunk együttes hozzárendelést. (FORALL, BULK COLLECT)

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

4. Ha lehet, használjunk SQL-t PL/SQL helyett:
   - Szűrjünk SQL-ben
   - Kurzor ciklusába ágyazott lekérdezéseknél a kurzor SELECT-je és a magban szereplő lekérdezés esetlegesen egyetlen SELECT-be is összevonható allekérdezések és összekapcsolások használatával.

```
FOR v_Konyv IN (SELECT id, cim FROM konyv)
  LOOP
  FOR v_Kocsonzo IN (SELECT kolcsonzo FROM
   kolcsonzes
                      WHERE konyv = v_Konyv.id)
    LOOP
     :
    END LOOP;
  END LOOP;
-- helyette
FOR v_Konyv_kolcsonzo IN (
  SELECT kv.id, kv.cim, ko.kolcsonzo
    FROM konyv kv, kolcsonzes ko
   WHERE ko.konyv = kv.id)
 LOOP
  :
 END LOOP;
```

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

4. Ha lehet, használjunk SQL-t PL/SQL helyett:
   - Kurzor ciklusába ágyazott DML utasításoknál a ciklus és a DML utasítás összevonható egyetlen MERGE utasításba.
   - Néha a több ciklussal összekombinált lekérdezések, esetleg a ciklusmagba ágyazott elágaztató utasítások lényegében halmazműveleteket valósítanak meg. (SQL halmazműveletek használata)

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

4. Ha lehet, használjunk SQL-t PL/SQL helyett:
   - Használjuk SQL-ben az analitikus függvényeket, aggregálásnál a KEEP utasításrészt. Nagyon gyakran lehet sok-sok lekérdezést megspórolni ezekkel.
   - Használjuk ki a SELECT lehetőségeit, a csoportképző eszközöket, a CONNECT BY, MODEL utasításrészeket stb.
   - Használjuk a feltételes INSERT-et olyan kurzort használó ciklus helyett, ahol a magban bizonyos feltételektől függően végzünk beszúrást egy-vagy több táblába.

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

4. Ha lehet, használjunk SQL-t PL/SQL helyett:
   - Néha ciklusok helyett használhatunk sorszámokat és egyéb adatokat generáló lekérdezéseket, amiket aztán más utasításokban alkérdésként használhatunk.

```
FOR i IN 1..100
LOOP
  INSERT INTO t(sorszam, paros_paratlan)
    VALUES(i, DECODE(MOD(ROWNUM,2),0, 'páros',
   'páratlan'));
END LOOP;
-- helyette
INSERT INTO t(sorszam, paros_paratlan)
  SELECT ROWNUM sorszam, DECODE(MOD(ROWNUM,2),0, 'páros',
   'páratlan') paros_paratlan
    FROM dual CONNECT BY LEVEL <= 100;
```

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

4. Ha lehet, használjunk SQL-t PL/SQL helyett:
   - Használjuk DML utasításoknál a hibanaplózás funkciót, ahelyett hogy minden utasítás előtt explicit ellenőriznénk a DML utasítás előfeltételeit
5. SQL-ben használt függvényeknél, ha lehet, adjuk meg a DETERMINISTIC kulcsszót.
6. SQL-ben szereplő függvényhívások vagy bonyolult kifejezések esetén, törekedjünk a hívások/kiértékelések számának minimalizálására.

```
SELECT DISTINCT SQRT(x) FROM ...
-- helyette
SELECT SQRT(distinct_x) FROM
    (SELECT DISTINCT x AS distinct_x FROM ... )
```

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

7. Nagyméretű paraméterek átadásakor fontoljuk meg a NOCOPY használatát.
8. Logikai operátorokkal képzett bonyolult kifejezésekben használjuk ki a rövidzár kiértékelést.
9. Kerüljük az adatkonverziót, főleg az implicit adatkonverziót. Használjunk megfelelő típusú literálokat és változókat.
10. A számítások eredményeit gyorsítás céljából őrizzük meg.
11. Számításoknál használjunk megfelelő típust

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

12. Lokális VARCHAR2 típusú változók hosszát válasszuk elég nagyra a futási idejű hibák megelőzése céljából.
13. A hatékonyabb memóriahasználat és a függőségek egyszerűsítése érdekében a logikailag összetartozó alprogramokat tegyük egy csomagba.
14. A munkamenetek által visszatérően és elég gyakran használt nagyméretű csomagokat rögzítsük a memóriába a DBMS_SHARED_POOL csomaggal. Az ilyen csomag kódja mindig a memóriában, az osztott tartományban marad, függetlenül az osztott tartomány telítődésétől és a csomag használatának sűrűségétől.
15. Ne használjuk a COMMIT parancsot túl gyakran és fölöslegesen. Ha tehetjük, tegyük a COMMIT-ot cikluson kívülre, vagy cikluson belül valamilyen gyakorisággal hajtsuk csak végre.

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

16. Memóriában rendezhetünk asszociatív tömbbel. Ha egy asszociatív tömbbe elemeket szúrunk be és az elemeken a FIRST, NEXT vagy a LAST, PRIOR párokkal iterálunk, a kulcsokon rendezetten haladunk végig.
17. Explicit OPEN utasítással megnyitott kurzor esetén a használó blokk kivételkezelőjének valamennyi ágában biztosítsuk, hogy a kurzor véletlenül se maradhasson nyitva. A nyitott kurzorok száma adatbázis példányonként limitált, telítettsége kényszerleálláshoz is vezethet.
18. Készítsük fel a programjainkat arra, hogy támogassák az előre nem látott hibák könnyű lokalizálását és javítását.
    - DBMS_UTILITY csomag
        - FORMAT_CALL_STACK,
        - FORMAT_ERROR_BACKTRACE
        - FORMAT_ERROR_STACK

Tanácsok a hangoláshoz és a hatékonyság növeléséhez

19. A kód átláthatóságát növeli és a „copy-paste” hibák előfordulását csökkenti az, ha az alprogramokat, triggereket és csomagokat záró END utasításban használjuk a programegység nevét.
20. Értelmezzük a fordító figyelmeztetéseit, a kód javításával törekedjünk azok megszüntetésére.
21. Ismerjük meg a szerver alapcsomagjait.
22. Ismerjük meg és használjuk a beépített SQL függvényeket.