Vergessene (?) SQL- und PL/SQL-Funktionen

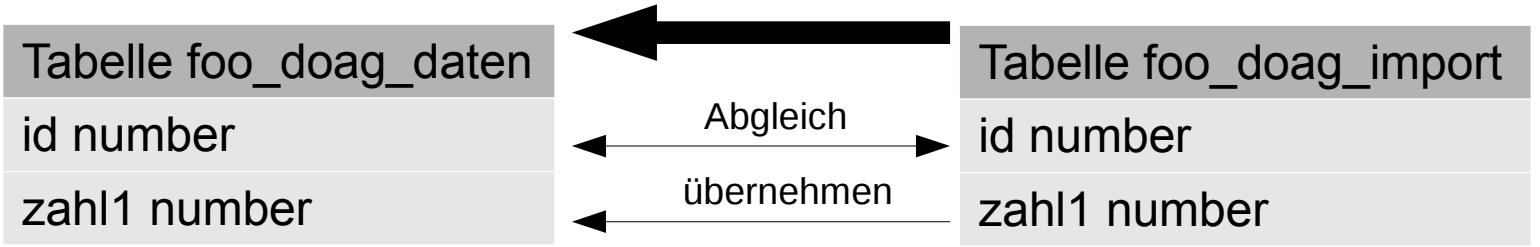
Folge 1

Vortrag DOAG-Regiogruppe Freiburg/Südbaden 29.01.13

bulk collect (1)

Aufgabenstellung:

ca. 50.000 eingelesene Werte an Hand einer ID auf eine vorhandene Tabelle übernehmen



bulk collect (2)

„Standard“-Lösung mit Schleife:

```
declare
begin

      for c in (select id, zahl1 from foo_doag_import)
      loop
            update foo_doag_daten
                  set zahl1 = c.zahl1
             where id = c.id
            ;
      end loop;

end;
/
```

bulk collect (3)

Lösung mit „bulk“-Anweisungen:

```
declare
      type zahlen_tt is table of number index by binary_integer;
      tab_id zahlen_tt;
      tab_wert zahlen_tt;

begin

      select id, zahl1
      bulk collect into tab_id, tab_wert
      from foo_doag_import;

      forall i in 1..tab_id.count
            update foo_doag_daten
                  set zahl1 = tab_wert(i)
             where id = tab_id(i)
      ;
end;
/
```

bulk collect (4)

Laufzeiten-Vergleich:

(ca. 50.000 Zeilen, DB 11R2, aktueller Standard-PC mit openSuSE Linux)

Laufzeiten in sec	mit Index auf ID	ohne Index auf ID
for-Schleife (statisches SQL)	7,8	69,8
for-Schleife (dynamisches SQL)	8,8	71,3
bulk collect (statisch+dynamisch)	3,8	66,3

Laufzeit-Gewinn v.a. durch

- Einsparung des „ständigen“ Context-Wechsels zwischen PL/SQL und SQL
- Einsparung der (identischen) parse-Phase bei dynamischem SQL

bulk collect (5)

- mehrere Möglichkeiten zur Fehlerbehandlung („save exceptions“)
- auch zusammen mit dynamischen SQL:

```
declare
        type zahlen_tt is table of number index by binary_integer;

        tab_id zahlen_tt;
        tab_wert zahlen_tt;

        l_id number := 1;
        l_spalte varchar2(30) := 'zahl1';
begin

        execute immediate  'select id, '|| l_spalte ||' from foo_doag_import where id >= :p_id'
        bulk collect into tab_id, tab_wert
        using l_id
        ;

        forall i in 1..tab_id.count
                execute immediate 'update foo_doag_regio set '|| l_spalte ||' = :p_wert  where id = :p_id '
                using tab_wert(i), tab_id(i)
        ;
end;
/
```

subselect (1)

Ausgangslage:

- Tabelle mit Wetterstationen mit eine Zeile pro Station, Jahr + 12 Monatswerte
- Jahre fehlen u.U. komplett (d.h. keine Zeile)

Station	Jahr	Jan	Feb	Mar	Apr	Mai	...
1030	2004	645	544	526	319	204	..
1030	2005	680	538	574	289	217	..
1030	2007	621	564	498	304	221	..
..	..	..	..	..	..	..	..

- auszuwertende Jahre und Monate sind in Tabellen hinterlegt

subselect (2)

Abfrage „Jahr, Monat, Gradtagszahl“ einschließlich „fehlender“ Jahre, dazu Jahre+Monate als subselect

```
select
       j.jahr
      ,j.monat
      ,decode(j.monat
            ,1 ,w.monat_01
            ...
            ,12,w.monat_12
            ) as gtzahl
  from
        em3_t_wetterstation w
      -- subselect liefert vollständiges Kreuzprodukt (Jahr, Monat)
      ,(select jahr, monat
          from em3_t_auswertung_monate m
               ,em3_t_auswertung_jahre j
       ) j
  where w.wetterstationen_nr(+) = 9029
     and w.jahr(+) = j.jahr
;
```

subselect (3)

Wie zuvor, aber zusätzlich das Maximum der Januar-Werte im +/- 5-Jahre-Zeitraum

```
select
       j.jahr
      ,j.monat
      , (select max(monat_12) from em3_t_wetterstation
         where jahr between j.jahr-5 and j.jahr+5) as januar_max
      ,decode(j.monat
            ,1 ,w.monat_01
            ...
            ,12,w.monat_12
            ) as gtzahl
  from
       em3_t_wetterstation w
      ,(select jahr, monat
          from em3_t_auswertung_monate m
               ,em3_t_auswertung_jahre j
       ) j
  where w.wetterstationen_nr(+) = 9029
    and w.jahr(+) = j.jahr
;
```

subselect (4)

Abfrage: Jahre+Monate ohne Gradtagszahlen, mittels „not exists“ + subselect

```
select
        j.jahr
      , m.monat
  from em3_t_auswertung_monate m
      ,em3_t_auswertung_jahre j
  where jahr < extract (YEAR from sysdate)
    and not exists
        (select wetterstationen_nr
           from em3_t_wetterstation w
          where jahr = j.jahr
            and decode(m.monat
                      ,1 ,w.monat_01
                      .....
                      ,12,w.monat_12
                      ) is not null
        )
  order by jahr, monat
;
```

„with“ (1)

- Unterabfragen werden ausgelagert in einen „temporär definierten View“
- „with t_view as (select ….) select …. from t_view“
- komplexe Abfragen können damit besser strukturiert werden
- die mit „with“ definierte Abfrage kann im select mehrfach verwendet werden
- alternative Lösung zu subselect oder views
- Laufzeiten müssen im Einzelfall verglichen werden

„with“ (2)

Abfrage „Jahr, Monat, Gradtagszahl“ einschließlich „fehlender“ Jahre, dazu Jahre+Monate mit „with“

```
with t_alle_monate as
      -- vollständiges Kreuzprodukt (Jahr, Monat)
       (select jahr, monat
          from em3_t_auswertung_monate m
                ,em3_t_auswertung_jahre j
       )
select
       j.jahr
      ,j.monat
      ,decode(j.monat
            ,1 ,w.monat_01
            ...
            ,12,w.monat_12
            ) as gtzahl
 from
        em3_t_wetterstation w
      , t_alle_monate j
 where w.wetterstationen_nr(+) = 9029
    and w.jahr(+) = j.jahr
;
```

materialized views (1)

- Ergebnisse komplexer Abfragen als „de-normalisierte“ Tabelle hinterlegen
- typische Anwendungsgebiete: DWH, Schnittstellen
- MV sind „read only“
- MV können mit eigenen Indices versehen werden
- MV belegen Speicherplatz
- performanter als herkömmliche Views, dafür i.d.R. keine aktuelle Daten
- Problemzone ist der „refresh“ der Daten
  - manuell vs. zeitgesteuert vs. automatisch
  - full vs. incremental („fast“)
- nicht geeignet für zeitkritische Daten mit hoher Änderungsrate

Fragen?

Vorschläge für Fortsetzungen?

Beiträge von ???