sphinx IT CONSULTING

VPD als Mittel zum Performancetuning im DWH

18.11.2013 | Mag. Dr. Thomas Petrik

VPD - Virtual Private Database

- Row Level Security (RLS)
  - auf Basis von Session-Attributen
- Fine Grained Access Control (FGAC)
- seit Oracle 8.1.5
- seit 10g auch Column Security
- unverändert in 12c
- Teil der Enterprise Edition - kostenlos

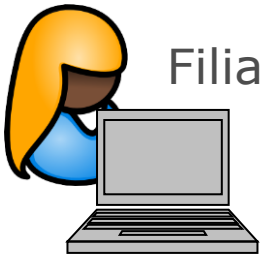
VPD - Funktionsweise

Revisionsmitarbeiter
sieht alle Kunden

```
select * from customers
```

CUSTOMERS			
CID	**CTYPE**	**CNAME**	**…**
100001	VIP	Lämpel	…
100002	CUS	Böck	…
100003	VIP	Bolte	…
100004	CUS	Müller	…
100005	CUS	…	…

VPD – Funktionsweise /2



Filialmitarbeiterin

sieht KEINE VIP-Kunden

```
select * from customers
```

CUSTOMERS			
CID	**CTYPE**	**CNAME**	**…**
100002	CUS	Böck	…
100004	CUS	Müller	…
100005	CUS	…	…

VPD – Tabelle + Policy-Function

```
CREATE TABLE customers
(
    cid       NUMBER (10)
   ,ctype     CHAR (3)
   ,cname     VARCHAR2(30)
   …
);
```

```
CREATE FUNCTION vpd_customers (
    owner    IN VARCHAR2
   ,tabname  IN VARCHAR2
) RETURN VARCHAR2
IS
BEGIN
    IF USER != 'REVISOR'
    THEN
        RETURN 'ctype = ''CUS''';
    ELSE
        RETURN NULL;
    END IF;
END;
```

implizite Where-Clause

VPD – DBMS_RLS.add_policy

```
BEGIN
    DBMS_RLS.add_policy (
         object_schema      => NULL
        ,object_name        => 'CUSTOMERS'
        ,policy_name        => 'PLC_CUSTOMERS'
        ,function_schema    => NULL
        ,policy_function    => 'VPD_CUSTOMERS'
        ,statement_types    => 'SELECT'
        ,policy_type        => DBMS_RLS.dynamic
        ,enable             => TRUE);
END;
```

nur für Selects

neu evaluiert bei jedem Aufruf

VPD – Execution Plan

```
SELECT * FROM customers;

-------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost
-------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |     2 |    54 |     3
|*  1 |  TABLE ACCESS FULL| CUSTOMERS |     2 |    54 |     3
-------------------------------------------------------------------

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

   1 - filter("CTYPE"='CUS')
```

VPD – implizite Where-Clause

- VPD steuert implizit eine zusätzliche Where-Clause ein

```
SELECT *
  FROM customers
 WHERE ctype = 'CUS'
```

- völlig transparent für den Benutzer
- zusätzliche Where-Clause sieht man
  - als Filter in explain plan
  - in v$vpd_policy

DWH – Performance-Falle

- Beispiel: nach Datum partitionierte Umsatztabelle
  - unterschiedliche Verteilung auf mehrere Filialen / Tag
  - täglich schwankende Verteilung

```
CREATE TABLE umsatz
(
   datum      DATE
  ,filiale    NUMBER (10)
  ,ktonr      NUMBER (12)
  ,betrag     NUMBER (15,2)
)
PARTITION BY RANGE (datum)(
     PARTITION p20130304 VALUES LESS THAN (TO_DATE ('20130305', 'yyyymmdd'))
    ,PARTITION p20130305 VALUES LESS THAN (TO_DATE ('20130306', 'yyyymmdd'))
    ,PARTITION phigh VALUES LESS THAN (maxvalue));
```

```
CREATE INDEX umsatz_ix00
   ON umsatz (filiale) LOCAL;
```

DWH – Performance-Falle /2

```
  SELECT datum, filiale, COUNT (1)
    FROM umsatz
GROUP BY datum, filiale
ORDER BY datum, filiale;
```

UMSATZ		
DATUM	**FILIALE**	**COUNT(1)**
04.03.2013	100	10
04.03.2013	200	10000
05.03.2013	100	10000
05.03.2013	200	10000

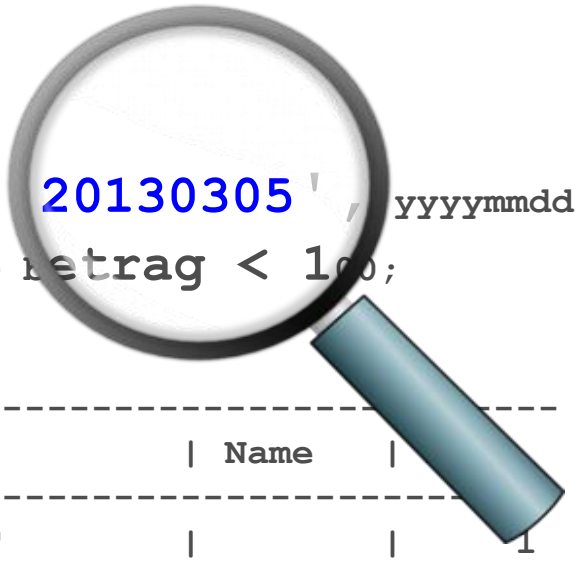
DWH – Performance-Falle /3

```
SELECT COUNT (1)
  FROM umsatz
 WHERE datum = TO_DATE ( '20130304', 'yyyymmdd')
   AND filiale = 100 AND betrag < 100;
```

```
-----------------------------------------------------------------------
| Id  | Operation                                | Name         | Rows  |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |              |     1 |
|   1 |  SORT AGGREGATE                          |              |     1 |
|   2 |   PARTITION RANGE SINGLE                 |              |    10 |
|*  3 |    TABLE ACCESS BY LOCAL INDEX ROWID| UMSATZ       |    10 |
|*  4 |     INDEX RANGE SCAN                     | UMSATZ_IX00  |    10 |
-----------------------------------------------------------------------
```

DWH – Performance-Falle /4

```
SELECT COUNT (1)
  FROM umsatz
 WHERE datum = TO_DATE ( '20130305', 'yyyymmdd')
   AND filiale = 100 AND betrag < 100;
```



```
-----------------------------------------------
| Id  | Operation                | Name    |
-----------------------------------------------
|   0 | SELECT STATEMENT         |         |     1
|   1 |  SORT AGGREGATE          |         |     1
|   2 |   PARTITION RANGE SINGLE|         | 10000
|*  3 |    TABLE ACCESS FULL     | UMSATZ  | 10000
-----------------------------------------------
```

DWH – Fluch der Bindvariablen

- Bindvariablen anstelle von Literalen - welcher Plan wird nun erstellt?
- Bind Peeking: seit Oracle 9i
  - die Bindvariablen der 1. Execution werden zu Hilfe genommen
    - v$sql_bind_capture
    - **`dbms_xplan.display_cursor(format => '+peeked_binds')`**
  - Plan bleibt unverändert
  - wechselnde Kardinalität bei folgenden Ausführungen bleibt unberücksichtigt
  - Plan ist für andere Partitionen ungeeignet
  - abschaltbar: _optim_peek_user_binds = false

Bind Peeking

```
EXEC :b1 := '20130304';
SELECT COUNT (1)
  FROM umsatz
 WHERE datum = TO_DATE (:b1,'yyyymmdd')
   AND filiale = 100 AND betrag < 100;

EXEC :b1 := '20130305';
SELECT ...

-------------------------------------------------------------------------
| Id  | Operation                              | Name         | Rows
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |              |
|   1 |  SORT AGGREGATE                        |              |      1
|   2 |   PARTITION RANGE SINGLE               |              |     10
|*  3 |    TABLE ACCESS BY LOCAL INDEX ROWID| UMSATZ       |     10
|*  4 |     INDEX RANGE SCAN                   | UMSATZ_IX00  |     10
-------------------------------------------------------------------------

Peeked Binds (identified by position):
--------------------------------------
   1 - :B1 (VARCHAR2(30), CSID=873): '20130304'
```

Adaptive Cursor Sharing

- Oracle 11g: Adaptive Cursor Sharing (ACS)
  - Basis Cursor wird als "**bind sensitive**" markiert
    - Predicate Columns müssen Histogramme haben
  - neuerliche Prüfung des Plans bei jeder weiteren Durchführung mit neuen Bindvariablen
  - Wenn sich ein besserer Plan ergibt, wird dieser als "**bind aware**" gekennzeichnet
    - Basisplan wird verworfen
    - neuer Child Cursor (gekoppelt an Bind Values)
  - Mindestens 1 mal erfolgt eine Execution mit ungünstigem Plan

  - Weitere Restriktionen
    - nicht mit Parallel Query
    - nur für einfache Operatoren (>,<,=,like etc.)
    - max. 14 Bindvariable

ACS - Internals

- v$sql: Parent Cursor (child_number = 0)
  - Column is_shareable = 'Y'
    - Bind Peeking wurde verwendet
  - Column is_bind_sensitive = 'Y'
    - ACS wird verwendet
- v$sql: neue Child Cursor sind "bind aware"
  - Column is_bind_aware = 'Y'
  - Parent Cursor wird später verworfen (is_shareable = 'N')
- Cursor Sharing Views
  - v$sql_cs_histogram
  - v$sql_cs_statistics
  - v$sql_cs_selectivity
- Oracle Note: 740052.1

DWH – Fluch der Bindvariablen /2

- Ungünstiger Plan im DWH bedeutet bei großen Datenmengen und komplexen Joins meist, dass die Abfrage nie fertig wird.
- Daher kommt ACS nie zum Zug
  - außerdem zu viele Restriktionen
- Lösung:
  - Umschreiben des Codes auf Literale
    - hoher Aufwand
    - oft gar nicht möglich
  - transparentes Einfügen einer zusätzlichen Where-Clause
    - Abfragen laufen meist in einem einheitlichen Datumskontext
    - ideal für Batchaktivitäten

VPD & DWH - Lösungsansatz

- Ein PL/SQL-Aufruf soll in einem bestimmten Datumskontext laufen
  - alle betroffenen Tabellen haben die gleiche VPD
- Vergabe einer eindeutigen Laufnummer, ID, etc.
- Steuertabelle: CTL_ACTION
  - Relation ACTION_ID / DATUM
- ACTION_ID wird in der Session hinterlegt
  - PL/SQL-Variable, oder
  - im Context, z.B. dbms_application_info.set_action
- VPD-Function holt DATUM aus CTL_ACTION

VPD & DWH – Lösungsansatz /2

```
begin
   -- ACTION in SYS_CONTEXT befüllen
   dbms_application_info.set_action('BATCH1');

   -- ACTION mit Datum verknüpfen
   insert into ctl_action (action_id, datum)
      values('BATCH1','20130304');
   commit;

   -- eigentliche Berechnung
   calculate(…);

   delete from ctl_action where action_id = 'BATCH1';
   commit;
end;
```

VPD & DWH – Lösungsansatz /3

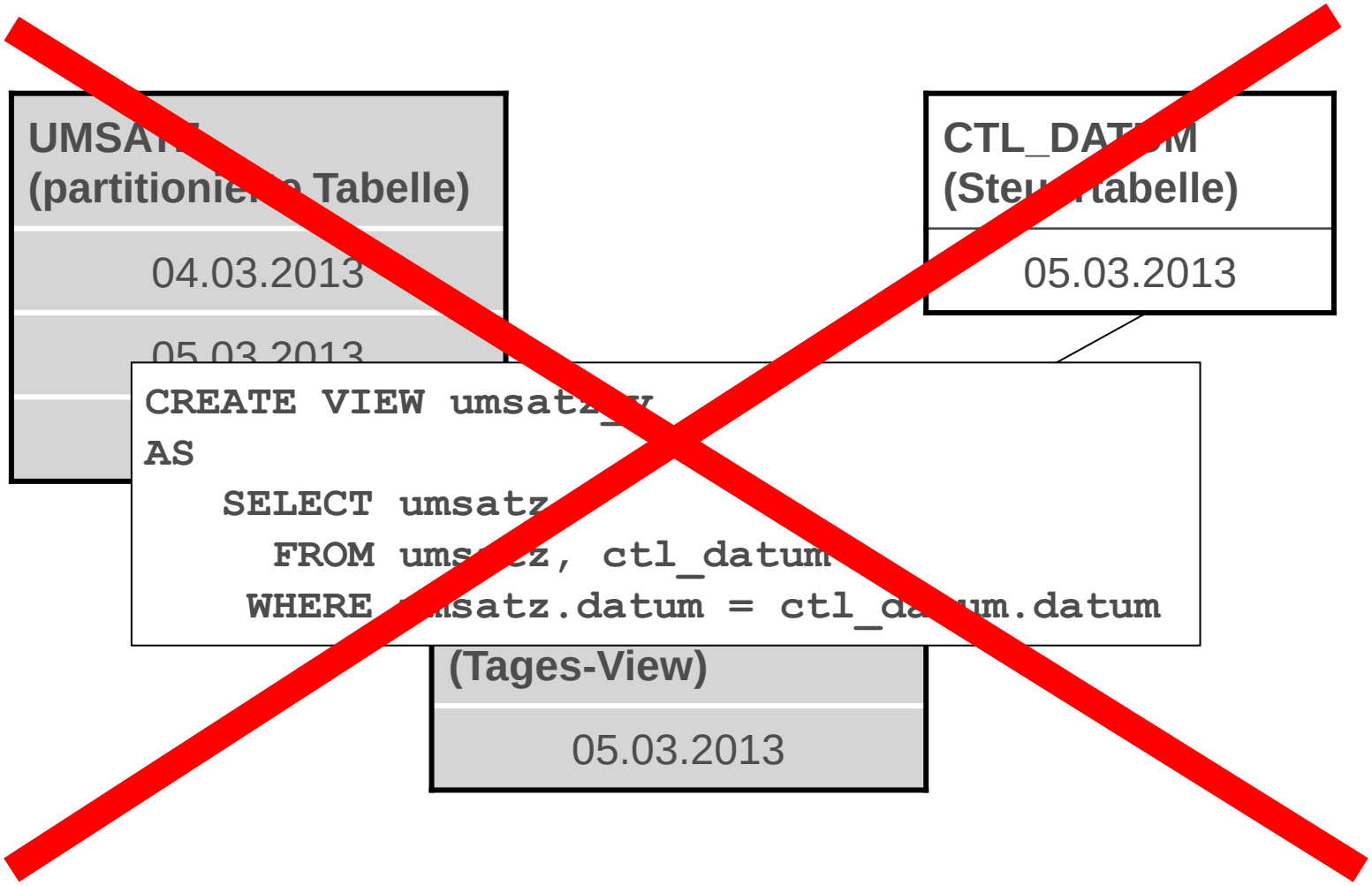
```
CREATE FUNCTION vpd_batch (owner IN VARCHAR2, tabname IN
   VARCHAR2) RETURN VARCHAR2
IS
   v_datum   VARCHAR2 (8);
BEGIN
   BEGIN
      SELECT datum
        INTO v_datum
        FROM ctl_action
       WHERE action_id = SYS_CONTEXT ('USERENV', 'ACTION');
   EXCEPTION
      WHEN NO_DATA_FOUND
      THEN
         RETURN NULL;
   END;

   RETURN 'datum = to_date(''' || v_datum || ''',''yyyymmdd'')';
END;
```

keine Where-Clause,
keine Einschränkung

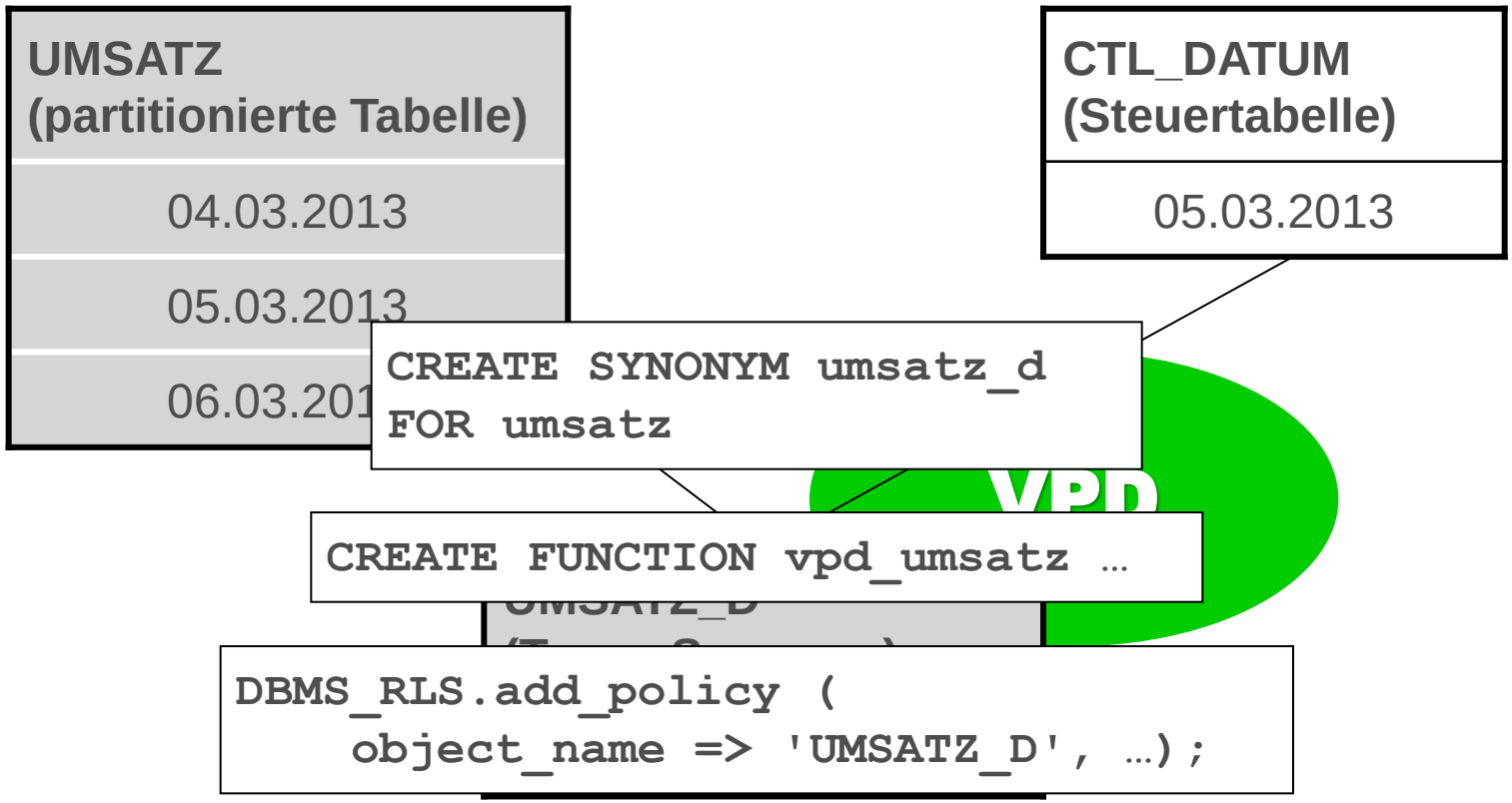
VPD & DWH – Lösungsansatz /4

- Steuertabelle CTL_ACTION kann durch den Tabellennamen erweitert werden
  - Datumseinschränkung tabellenspezifisch
  - von / bis Datum möglich
  - beliebige weitere Einschränkungen (Filialen, Institute, etc.)
- VPD-Function und Policy können für alle Tabellen ident sein
  - Tabelle wird der Funktion als Inputvariable übergeben
  - Wenn das Datumsfeld nicht in jeder Tabelle "DATUM" heißt, müsste auch diese Information hinterlegt werden.

DWH - tagesaktuelles Reporting



DWH - tagesaktuelles Reporting /2



Datenschutz mittels VPD

- Datenänderung soll auf ein bestimmtes Datum beschränkt sein
  - Tagesverarbeitung
  - keine Änderung historischer Daten
- Schutz vor unbeabsichtigter Änderung
  - z.B. durch Software Bugs
- Einführung eines Quasi-Read-Only-Modus
  - kein RO-Modus für einzelne Partitionen
  - Partitionen könnten allenfalls in einen RO-Tablespace verschoben werden
  - kein RO-Modus auf Row-Ebene in nicht-partitionierten Tabellen

Datenschutz mittels VPD /2

- VPD-Function mit Einschränkung auf das aktuelle Verarbeitungsdatum
  - analog Performance-VPD
- DBMS_RLS.add_policy:
  - `statement_types => 'SELECT,INSERT,UPDATE,DELETE'`
  - `update_check => TRUE`
- INSERT auf ein anderes Datum gibt ORA-28115
- UPDATE und DELETE führen die Changes nicht durch
  - Nachteil: keine Fehlermeldung

Zusammenfassung

- Vermeidung von Bindvariablen im DWH
  - Ungleiche Datenverteilung über die Partitionen führt zu suboptimalen Plänen
- Zusätzliche Where-Klauseln mit Literalen
  - durch VPD eingesteuert
  - Verhalten der VPD über Steuertabellen oder Context kontrollierbar
  - Code mit Bindvariablen kann bleiben
  - Optimierungsmaßnahme durch DBA umsetzbar
- VPD als Schutz vor ungewollten Datenänderungen
  - VPD für insert, update, delete
- Keine Neuerungen in Oracle 12c

Q & A

Mag. Dr. Thomas Petrik | 0043 1/599 31 – 0
thomas.petrik@sphinx.at