Database In Memory 12.2 in der Praxis

Andreas Ballenthin
MYTOYS GROUP

Schlüsselworte
Oracle RDBMS 12.2 In Memory

Einleitung
Im Rahmen eines Redesigns des Data Warehouses setzen wir bei myToys offensiv neue Technologien ein. Wir entschieden uns für eine Lösung mit Exadata X6, Oracle RDBMS 12.2 und Tableau. Entscheidend war unter anderem die Database In Memory Option. Unsere Erfahrungen mit der Database In Memory Option 12.2 soll im Vortrag beleuchtet werden

Spaltenselektion, Segmentselektion, Komprimierung
Es ist möglich und nützlich, Spalten einer Tabelle gezielt In Memory zu laden oder dies gezielt nicht zu tun. Technische Attribute wie Ladezeitstempel, Lade-ID oder selten benutzte Attribute schließe ich von der In-Memory-Propagierung aus.

```
alter table f_sales_details_doag
   inmemory memcompress for query low (order_date,x_sales_qty_n)
   no inmemory (orderlines_id,x_sales_qty_a,x_sales_qty_bs,x_sales_qty_ns);
```

Im Dictionary finde ich dazu in gv$im_column_level:

```
TABLE_NAME            COLUMN_NAME    INMEMORY_COMPRESSION
--------------------  -------------  --------------------
F_SALES_DETAILS_DOAG  ORDERLINES_ID   NO INMEMORY
F_SALES_DETAILS_DOAG  ORDER_DATE      FOR QUERY LOW
F_SALES_DETAILS_DOAG  X_SALES_QTY_N   FOR QUERY LOW
F_SALES_DETAILS_DOAG  X_SALES_QTY_A   NO INMEMORY
F_SALES_DETAILS_DOAG  X_SALES_QTY_BS  NO INMEMORY
F_SALES_DETAILS_DOAG  X_SALES_QTY_NS  NO INMEMORY
```

Eine Abfrage, die ausschließlich In-Memory-Spalten benutzt, wird mit einem TABLE ACCESS INMEMORY FULL ausgeführt, ansonsten wird der Column-Store (erwartungsgemäß) nicht genutzt.

```
explain plan for
select /*+ parallel */ order_date, sum(x_sales_qty_n), sum(x_sales_qty_a)
  from dwh_im.f_sales_details_doag
 group by order_date;

-------------------------------------------------------------------------
| Id  | Operation                    | Name                | Rows  |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                     |   242 |
|   1 |  PX COORDINATOR              |                     |       |
|   2 |   PX SEND QC (RANDOM)        | :TQ10000            |   242 |
|   3 |    PX PARTITION RANGE ALL    |                     |   242 |
|   4 |     HASH GROUP BY            |                     |   242 |
|   5 |      TABLE ACCESS STORAGE FULL| F_SALES_DETAILS_DOAG |    18M|
-------------------------------------------------------------------------

Note
-----
   - automatic DOP: Computed Degree of Parallelism is 2
```

Es ist zudem möglich, Column Stores mit unterschiedlichen Kompressionsverfahren (nocompress, for query low/high, for capacity low/high) zu komprimieren. Einen Anwendungsfall habe ich dafür nicht gefunden. Wesentlich praxisrelevanter ist die Funktionalität, gezielt (Sub-)Partitionen des Row Stores In-Memory zu laden und dies mit verschiedenen Komprimierungsverfahren zu tun. So lassen sich häufig genutzte Partitionen In-Memory halten und weniger häufig genutzte Partitionen davon ausschließen. Allerdings zeigt ein Ausführungsplan missverständliche Informationen.
Mit einem Beispiel möchte ich meine Beobachtungen verdeutlichen:

```
create table dwh_im.f_sales_details_doag2
 (orderlines_id char (40 char) not null enable,
  order_date date not null enable,
  x_sales_qty_n number (38,0),
        constraint f_sls_d_doag2_pk primary key (orderlines_id)
…
  partition by range (order_date) interval (numtoyminterval(1, 'month'));
```

Partition	Konfiguration
to_date ('012017','MMYYYY')	no inmemory
to_date ('022017','MMYYYY')	no inmemory
to_date ('032017','MMYYYY')	no inmemory
to_date ('042017','MMYYYY')	inmemory memcompress for capacity low
to_date ('052017','MMYYYY')	inmemory memcompress for capacity low
to_date ('062017','MMYYYY')	inmemory memcompress for capacity low
to_date ('072017','MMYYYY')	inmemory memcompress for query low
to_date ('082017','MMYYYY')	inmemory memcompress for query low

Der Ausführungsplan einer Abfrage auf Januardaten zeigt TABLE ACCESS STORAGE FULL, der einer Abfrage auf Aprildaten zeigt TABLE ACCESS INMEMORY FULL, beides ist richtig. Missverständlich ist jedoch der Ausführungsplan für März- und Aprildaten zusammen, dort ist die falsche Information TABLE ACCESS INMEMORY FULL zu sehen, obwohl hier sowohl Storage- als auch InMemory-Zugriff notwendig sind.

```
-------------------------------------------------------------------------
| Id  | Operation                         | Name                | Rows  |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT                  |                     |     1 |
|   1 |  SORT AGGREGATE                   |                     |     1 |
|   2 |   PX COORDINATOR                  |                     |       |
|   3 |    PX SEND QC (RANDOM)            | :TQ10000            |     1 |
|   4 |     SORT AGGREGATE                |                     |     1 |
|   5 |      PX BLOCK ITERATOR            |                     |   341K|
|*  6 |       TABLE ACCESS INMEMORY FULL| F_SALES_DETAILS_DOAG2 |   341K|
-------------------------------------------------------------------------

   6 - inmemory("ORDER_DATE"<=TO_DATE(' 2017-04-02 00:00:00', 'syyyy-mm-dd hh24:mi:ss')
              AND "ORDER_DATE">=TO_DATE(' 2017-03-31 00:00:00','syyyy-mm-dd hh24:mi:ss'))
       filter("ORDER_DATE"<=TO_DATE(' 2017-04-02 00:00:00', 'syyyy-mm-dd hh24:mi:ss')
              AND "ORDER_DATE">=TO_DATE(' 2017-03-31 00:00:00','syyyy-mm-dd hh24:mi:ss'))
```

Virtual Columns (In-Memory-Expressions)
Ich empfinde die Funktionalität, eine virtuelle Spalte im Row-Store zu definieren, um sie im Column Store "materialisieren" zu können, als wichtig und sinnvoll.

```
alter table f_sales_details_doag add diff_sales_qty_n_a as (x_sales_qty_n - x_sales_qty_a);
```

In einer Konfiguration

```
alter table f_sales_details_doag
   inmemory memcompress for query low (order_date,x_sales_qty_n,diff_sales_qty_n_a)
   no inmemory (orderlines_id,x_sales_qty_a,x_sales_qty_bs,x_sales_qty_ns);
```

führt eine Abfrage auf diff_sales_qty_n_a jedoch für mich nicht erwartet zu einem TABLE ACCESS STORAGE FULL. Erst wenn beide Basisspalten der virtuellen Spalte InMemory geladen sind `(inmemory memcompress for query low (order_date,x_sales_qty_n,x_sales_qty_a, diff_sales_qty_n_a)`, wird wie intuitiv erwartet ein TABLE ACCESS INMEMORY FULL ausgeführt.
Abgeleitete Kennzahlen, wie Margenberechnungen, Währungsdarstellungen oder Umsatzwerte, kommen hinreichend häufig vor. Im RDBMS Oracle 12.1 mussten diese Kennzahlen auch im Row Store materialisiert werden.
Das intensiv beworbene Feature Expressions Statistics Store benutzen wir noch nicht.

Join Groups
Wird eine Join Group angelegt, so benutzen die beiden Joinattribute ein gemeinsames Compression Directory. So können Joins direkt im Compression Directory und eben nicht auf den Datenwerten ausgeführt werden. Leider ist diese Funktionalität nicht zu Ende entwickelt worden. Hinweise auf die Verwendung von Join Groups in Ausführungsplänen fehlen, ein Monitoring, ob eine Join Group korrekt populiert wurde, fehlt ebenfalls. Die Verwendung einer Join Group kann im Einzelfall mit notdürftig aus einem mit DBMS_SQLTUNE.REPORT_SQL_MONITOR_XML erstellten Report nachgewiesen werden. In den von mir geprüften realen Szenarien konnte ich keine signifikanten Performanceunterschiede zwischen InMemory mit Join Groups und InMemory ohne Join Groups feststellen.

Materialized Views
Mit In-Memory-Materialized-Views habe ich positive Erfahrungen gesammelt.

```
create materialized view mv_fsd_doag_day_art_shop parallel
 build immediate refresh complete on demand
enable query rewrite as
select f.order_date
       ,f.d_multishop_salespoint_sk
       ,f.d_article_sk
       ,sum(f.x_sales_qty_n)
       ,sum(f.x_sales_qty_a)
       ,sum(f.x_sales_qty_bs)
       ,sum(f.x_sales_qty_ns)
  from f_sales_details f
 group by f.order_date
         ,f.d_multishop_salespoint_sk
         ,f.d_article_sk;

alter table mv_fsd_doag_day_art_shop add constraint mv_fsd_doag_day_art_shop_fk_1
  foreign key (d_article_sk) references d_articles (article_sk) rely disable;
alter table mv_fsd_doag_day_art_shop add constraint mv_fsd_doag_day_art_shop_fk_2
  foreign key (d_multishop_salespoint_sk) references d_multishop_salespoints
(multishop_salespoint_sk) rely disable;
alter table mv_fsd_doag_day_art_shop add constraint mv_fsd_doag_day_art_shop_fk_3
  foreign key (order_date) references d_date (date_dt) rely disable;

alter table mv_fsd_doag_day_art_shop inmemory priority critical duplicate all;
```

Hilfreich ist die volle Integration von Query Rewrite und In-Memory-Back-Joins. Eine Parallelisierung der SQL-Statements ist nach wie vor sinnvoll und unerlässlich. Parallelausführung ist regelmäßig schneller als die Verwendung von Vektortransformationen.

```
alter session force parallel query;
explain plan for
select  dag.master_brand
       ,d_multishop_salespoint_sk
       ,sum(f.x_sales_qty_ns)
  from f_sales_details f
```

```
  inner join d_articles da
    on f.d_article_sk=da.article_sk
  inner join d_article_groupings dag
     on da.d_art_grp_id=dag.article_grp_id
 group by d_multishop_salespoint_sk
         ,dag.master_brand;
select * from table (dbms_xplan.display());
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                      | Rows  |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                           |   272K|
|   1 |  PX COORDINATOR                          |                           |       |
|   2 |   PX SEND QC (RANDOM)                    | :TQ10003                  |   272K|
|   3 |    HASH GROUP BY                         |                           |   272K|
|   4 |     PX RECEIVE                           |                           |   272K|
|   5 |      PX SEND HASH                        | :TQ10002                  |   272K|
|   6 |       HASH GROUP BY                      |                           |   272K|
|*  7 |        HASH JOIN                         |                           |   112M|
|   8 |         PX RECEIVE                       |                           |  4636K|
|   9 |          PX SEND BROADCAST               | :TQ10001                  |  4636K|
|  10 |           VIEW                           | VW_GBF_10                 |  4636K|
|  11 |            HASH GROUP BY                 |                           |  4636K|
|  12 |             PX RECEIVE                   |                           |  4636K|
|  13 |              PX SEND HASH                | :TQ10000                  |  4636K|
|  14 |               HASH GROUP BY              |                           |  4636K|
|* 15 |                HASH JOIN                 |                           |  4636K|
|  16 |                 TABLE ACCESS INMEMORY FULL | D_ARTICLE_GROUPINGS     | 83768 |
|  17 |                 PX BLOCK ITERATOR        |                           |  4636K|
|  18 |                  TABLE ACCESS INMEMORY FULL | D_ARTICLES              |  4636K|
|  19 |         PX BLOCK ITERATOR                |                           |   112M|
|  20 |          MAT_VIEW REWRITE ACCESS INMEMORY FULL| MV_FSD_DOAG_DAY_ART_SHOP |   112M|
-------------------------------------------------------------------------------------------
```

Meine Untersuchungen zur optimalen Performance im Rahmen des Redesigns des myToys Data Warehouses sind bei weitem noch nicht abgeschlossen. Neue Inkremente stellen uns vor neue Herausforderungen und Hürden und bieten so Anreiz und Notwendigkeit neuer Erkenntnisse.

Kontaktadresse:
Andreas Ballenthin
MYTOYS GROUP
Potsdamer Straße 192
D-10783 Berlin

Telefon: +49 (0) 30-72 62 01
E-Mail andreas.ballenthin@mytoys.de
Internet: www.mytoys.de