Database Memory Techniken für mehr Performance

Ulrike Schwinn
Oracle Deutschland B.V. & Co.KG
München

Schlüsselworte

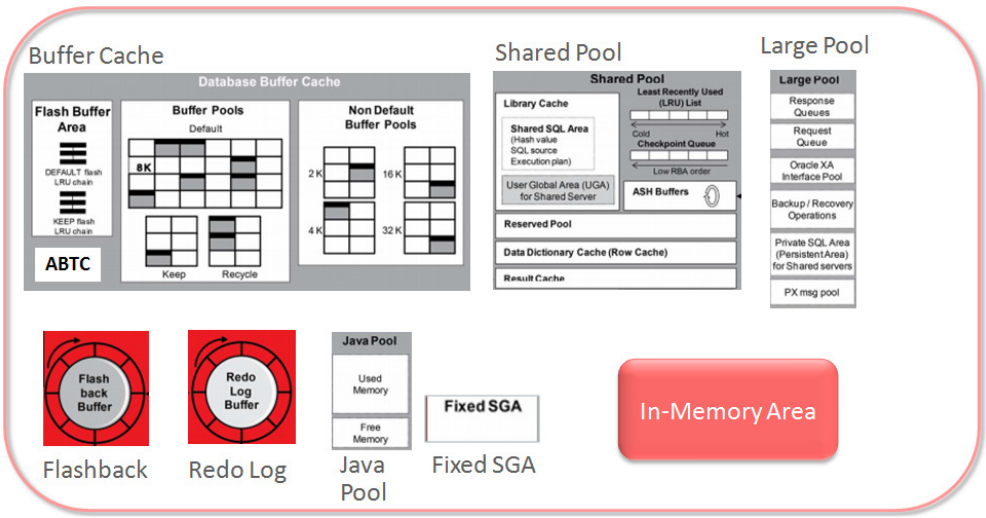
Buffer Cache, LRU, Temperatur, Keep und Recycle Pool, Preloading, Result Cache, Big Table Caching, Database Caching, In Memory DB, Column Store, Row Store

Einleitung

Oracle setzt auf den In-Memory Trend und bietet seit geraumer Zeit spezielle Techniken an, um Daten effizient im Memory zu speichern. Effektive Lagerung im Datenbank Cache ist dabei „groß geschrieben“, so daß beispielsweise „heiße Daten“ im schnellen Zugriff sind. Um bestimmte Bereiche und Applikationen noch besser bei der Ablage im Cache zu unterstützen, wurden zusätzliche Techniken in den letzten Jahren implementiert. So können Resultate von Statementausführungen oder PL/SQL Funktionen ebenso den Cache effizient nutzen, wie Tabellen, die mit „Full Table Scan“ Operationen abgefragt werden. Besonders auch mit der neuen Oracle Database In-Memory Technologie in Version 12*c* ist eine weitere interessante Neuigkeit hinzugefügt worden, die besonders bei analytischen Abfragen von großem Vorteil sein kann. Dabei zeichnet sie sich nicht nur durch eine neue Form des Caches aus, sondern profitiert zusätzlich von speziellen optimierten Statement-zugriffen.

Die Basics – Buffer Cache, LRU, Keep- und Recycle Pools

Starten wir mit ein paar grundsätzlichen Informationen, die dem Leser sicherlich aus seinen Anfangszeiten mit der Oracle Datenbank bekannt sein dürften. Das Oracle Datenbank Memory – die sogenannte **SGA** (System Global Area) – besteht aus mehreren Shared Memory Bereichen, die unterschiedliche Aufgaben erfüllen. Im Laufe der Jahre sind weitere Bereiche hinzugefügt worden, um das Funktionsfeld zu erweitern. Folgende Grafik liefert einen kleinen Einblick über die Bestandteile der SGA.



Betrachten wir zuerst den **Buffer Cache**. Für die meisten Operationen innerhalb der Oracle Datenbank wird der Buffer Cache zur Lagerung von Datenblöcken, die von Platte gelesen worden sind, verwendet. Ausgenommen sind dabei Operationen, die über einen Direct Path Read verarbeitet werden. Dies sind z.B. parallele Operationen oder Zugriffe auf große Tabellen. In welcher Form und wie lange werden die Daten im Buffer Cache gespeichert? Die Daten werden standardmäßig im Buffer Cache im **Row Format** (Zeilenformat) abgespeichert; es wird dabei weder komprimiert oder gar umsortiert. Da die Verwaltung der Buffer dabei effizient sein muss, wird ein Least Recently Used (kurz **LRU**) Algorithmus bei der Verwaltung der Blöcke im Buffer Cache verwendet. Blöcke, die ständig in Verwendung sind (auch „Hot" Blocks) werden an das oberen Ende der Liste plaziert, damit sie möglichst lange im Buffer Cache verweilen; selten verwendete Blöcke hingegen werden am unteren Ende der Liste gelagert. Neu mit Oracle Database 12*c* ist der **Temperatur basierende** Algorithmus auf Objektebene, der beim Automatic Big Table Caching Feature verwendet wird (mehr dazu im Kapitel Automatic Big Table Caching).

Möchte man das Automatische Memory Management nutzen, kann man mit dem Inititialisierungsparameter SGA_TARGET die Gesamtgröße der SGA festlegen. Automatisch werden dann die Memory Bereiche wie Buffer Cache, Shared Pool, Large Pool und Java Pool allokiert. Empfehlenswert ist dabei zusätzlich die Belegung der einzelnen poolspezifischen Parameter wie DB_CACHE_SIZE, SHARED_POOL_SIZE usw. um einen unteren Wert für diese Bereiche festzulegen. Die Advisor Views V$DB_CACHE_ADVISE bzw. V$MEMORY_CACHE_ADVISE, die auch graphisch im Enterprise Manager Cloud Control oder Database Express zur Verfügung stehen, geben dabei die Auslastung bei der aktuellen Buffergröße und bei einer hypothetischen Vergrößerung oder Verkleinerung des Pools an.

```
SQL> SELECT * FROM v$sga_target_advice ORDER BY sga_size;

 SGA_SIZE SGA_SIZE_FACTOR ESTD_DB_TIME ESTD_DB_TIME_FACTOR ESTD_PHYSICAL_READS
--------- --------------- ------------ ------------------- -------------------
      290              .5       448176              1.6578             1636103
      435             .75       339336              1.2552             1636103
      580               1       270344                   1             1201780
      725            1.25       239038               .8842              907584
      870             1.5       211517               .7824              513881
     1015            1.75       201866               .7467              513881
     1160               2       200703               .7424              513881
```

Für die meisten Anwendungen ist der Buffer Cache (auch Default Pool) sicher ausreichend. Wenn man allerdings mit einem Tablespace arbeiten möchte, der in Blockgröße von der Default Größe abweicht, muss dieser spezielle Pool explizit im Database Buffer Cache mit dem zugehörigen Parameter DB_<n>K_CACHE_SIZE konfiguriert werden - <n> steht für die Blockgröße. Erst danach lässt sich ein Tablespace mit **abweichender Blockgröße** zum Beispiel für komprimierte Tabellen erzeugen.

Kennt man sich mit dem Zugriffsmuster auf die Objekte aus, kann es auch sinnvoll sein den Buffer Cache in weitere Pools zu unterteilen. Es handelt sich dabei um den **KEEP** Pool für „Hot" Objekte und den **RECYCLE** Pool für „Cold" Objekte. Beide Pools werden separat mit den entsprechenden Parameter DB_KEEP_CACHE_SIZE bzw. DB_RECYCLE_CACHE_SIZE konfiguriert. Danach liegt es am Datenbankadministrator die Objekte dem entsprechenden Pool zuzuordnen und mit einem SQL Kommando in den Cache zu laden.

```
ALTER TABLE testkeep (BUFFER_POOL KEEP);
ALTER INDEX i_test (BUFFER_POOL KEEP);
SELECT /*+ FULL(T1) */ sum(numeric_column), min(txt_column) FROM testkeep T1;
```

Die Lagerung bzw. Auslagerung der Blöcke im KEEP bzw. RECYCLE Pool wird wie beim Default Pool über den LRU Algorithmus vorgenommen.

Speziell beim Tunen von Abfragen stellt man sich häufig die Frage, ob die Blöcke der abgefragten Objekte auch im Cache liegen. Verwendet man die AUTOTRACE Funktion in SQL*Plus kann man über die Statistiken wie „Physical Reads“ Hinweise darauf erhalten, ob und wie viele Plattenzugriffe für die Ausführung erforderlich waren. Bezogen auf die Blöcke eines Objekts gibt V$BH Auskunft über die Anzahl der Blöcke, die tatsächlich im Cache liegen.

```
SQL> SELECT o.object_name, o.object_type, o.owner, COUNT(*) NUMBER_OF_BLOCKS
     FROM dba_objects o, v$bh bh
     WHERE o.data_object_id = bh.objd
     AND (o.owner in ('SH'))
     GROUP BY o.object_name, o.owner, o.object_type
     ORDER BY COUNT(*);

OBJECT_NAME                OBJECT_TYPE     OWNER NUMBER_OF_BLOCKS
-------------------------- --------------- ----- ----------------
SALES_COPY                 TABLE           SH                    1
```

In unserem Beispiel liegt offensichtlich nur ein Block des Tabellensegments SALES_COPY im Buffer Cache.

Bei „Full Table Scan“ Operationen nutzt die Oracle Datenbank einen internen Algorithmus für den Buffer Cache. Ist die Tabelle klein (SMALL) wird beim Zugriff der Buffer Cache verwendet. Ist die Tabelle hingegen groß (LARGE), erfolgen „Direct Path Reads“. Dies bedeutet, es wird direkt von den Datendateien in die PGA gelesen. Der Parameter "_small_table_threshold" legt dabei die untere Grenze für die Größe von kleinen (SMALL) Tabellen fest. In Oracle Database 12*c* gibt es dazu folgende Ergänzung (laut Database Administration Guide Kapitel 14):

“Starting in Oracle Database 12c Release 1 (12.1.0.2), the buffer cache of a database instance automatically performs an internal calculation to determine whether memory is sufficient for the database to be fully cached in the instance SGA, and if caching tables on access would be beneficial for performance. If the whole database can fully fit in memory, and if various other internal criteria are met, then Oracle Database treats all tables in the database as small tables, and considers them eligible for caching. However, the database does not cache LOBs marked with the ***NOCACHE*** *attribute.Result Cache*”

Automatic Big Table Caching in 12c
Eigens für parallele und serielle „Full Table Scans“ ist in Oracle Database 12*c* ein neuer Cache Bereich im Buffer Cache eingeführt worden - der sogenannte **Automatic Big Table Caching** (auch kurz ABTC).

Die Idee dahinter ist einfach: Ein gewisser Teil des Buffer Caches wird für die Speicherung großer Objekte oder Teile davon reserviert, so dass die Abfragen von der Speicherung im

Cache profitieren können. Statt „Direct Path Reads“ wird dann der Big Table Cache verwendet. Um zu entscheiden, welche Objekte den Big Table Cache verwenden können, nutzt ABTC mehrere Kriterien. Entscheidend für die Nutzung ist dabei nicht nur die Größe des Objekts und die Größe des Big Table Caches. Im Unterschied zum Standard Buffer Cache Verhalten, das auf dem Block Level orientierten LRU Algorithmus basiert, spielt hier die **"Temperatur" der Objekte** (nicht Blöcke) - bei der Nutzung des Big Table Caches eine wichtige Rolle. Wichtig zu wissen ist, dass ABTC in Oracle Real Application Clusters (Oracle RAC) Umgebungen nur mit Parallel Query unterstützt wird; in Single Instance Umgebungen kann ABTC auch mit seriellen Abfragen verwendet werden.

Das Setup dazu ist einfach und im laufenden Betrieb möglich. Eingeschaltet wird das ABTC im **Single Instance** Umfeld über den dynamischen Initialisierungsparameter DB_BIG_TABLE_CACHE_PERCENT_TARGET. Dieser Parameter reserviert einen dedizierten Anteil am Buffer Cache (in Prozent). Um das Feature zu demonstrieren, nutzen wir nun eine einfache Abfrage auf die Tabelle SALES_COPY, die auch nach mehrmaliger Durchführung (hier dreifach) nicht im Buffer Cache gespeichert wird. Es wird ein Full Table Scan durchgeführt unter Verwendung von 35341 „Physical Reads“.

```
SQL> set autotrace traceonly
SQL> select sum(prod_id) from sales_copy;

Execution Plan
----------------------------------------------------------
Plan hash value: 2728018880
--------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows | Bytes | Cost (%CPU)| Time      |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |     1 |     4 |  9629   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE    |            |     1 |     4 |            |          |
|   2 |   TABLE ACCESS FULL| SALES_COPY |  7350K|    28M|  9629   (1)| 00:00:01 |
--------------------------------------------------------------------------------
Statistics
----------------------------------------------------------
         0  recursive calls
         0  db block gets
     35346  consistent gets
     35341  physical reads
         0  redo size
       550  bytes sent via SQL*Net to client
       551  bytes received via SQL*Net from client
         2  SQL*Net roundtrips to/from client
         0  sorts (memory)
         0  sorts (disk)
         1  rows processed
```

Im nächsten Schritt setzen wir nun den Parameter DB_BIG_TABLE_CACHE_PERCENT_TARGET auf 40. Das bedeutet 40 Prozent des Buffer Caches wird für Scans und damit für den Big Table Cache verwendet; die verbleibenden 60 Prozent stehen für andere Workloads zur Verfügung.

```
SQL> ALTER SYSTEM SET db_big_table_cache_percent_target=40;
```

Nun wiederholen wir die Abfrage auf die Tabelle SALES_COPY und führen erneut Abfragen auf die V$ Views V$BT_SCAN_CACHE und V$BT_SCAN_OBJ_TEMPS aus. Parallel dazu werden weitere Workloads und Scans durchgeführt.

```
SQL> SELECT bt_cache_alloc, bt_cache_target, object_count, memory_buf_alloc,
     min_cached_temp FROM v$bt_scan_cache;

BT_CACHE_ALLOC BT_CACHE_TARGET OBJECT_COUNT MEMORY_BUF_ALLOC
-------------- --------------- ------------ ----------------
    .400005755              40            3            50685
1  rows processed

SQL> SELECT o.object_name, cached_in_mem, size_in_blks, policy, temperature
     FROM v$bt_scan_obj_temps bt, dba_objects o
     WHERE bt.dataobj#=o.object_id;

OBJECT_NAME          CACHED_IN_MEM SIZE_IN_BLKS     POLICY TEMPERATURE
-------------------- ------------- ------------ ---------- -----------
FACT_PP_OUT_ITM_XXX          41700        44878    MEM_PART      185000
AB_ELEMENT_RELA               2644         2644        DISK        1000
SALES_COPY                       0        35421        DISK        5000
```

Die Tabelle FACT_PP_OUT_ITM_XXX ist fast vollständig in Memory (siehe Policy MEM_PART); die Tabellen SALES_COPY und AB_ELEMENT_RELA hingegen werden weiterhin von Disk gelesen (siehe Policy DISK). Die Entscheidung über das Cachen der Tabellen wird über den Spaltenwert TEMPERATURE gesteuert. Die minimale Temperatur liegt bei 1000. Bei jedem Zugriff auf ein Objekt erhöht die Datenbank die Temperatur des entsprechenden Objekts. Ein Objekt im Big Table Cache kann nur von einem Objekt mit höherer Temperatur verdrängt werden. Ist der Big Table Cache nicht ausreichend groß - wie in unserem Fall können nur die Objekte mit der höchsten Temperatur gespeichert werden. Dabei kommt auch ein teilweises Cachen des Objekts in Frage. Soll auch die Tabelle SALES_COPY im ABTC liegen, kann beispielsweise die Größe des Big Table Caches dynamisch erhöht werden.

Im Gegensatz zur Single Instance Implementierung wird das Automatic Big Table Caching in Oracle Real Application Clusters (Oracle RAC) Umgebungen nur von parallelen Abfragen unterstützt. Voraussetzung zur Verwendung ist in diesem Fall das Setzen von zwei Parametern:

- PARALLEL_DEGREE_POLICY muss den Wert AUTO oder ADAPTIVE (neu in 12*c*) haben.
- DB_BIG_TABLE_CACHE_PERCENT_TARGET gibt den Anteil am Database Buffer Cache (in Prozent) zur Verwendung an.

In RAC Umgebungen mit mehreren Nodes wird dabei das Objekt bzw. werden die Fragmente eines Objekts auf die Nodes verteilt. Das Monitoren und die Verwendung gleicht ansonsten der seriellen Verarbeitung. Auch hier erfolgt das Monitoring über die entsprechenden GV$Views.

Der Ergebnis- Cache

Seit Oracle Database 11*g* gibt es einen neuen Cache-Bereich im Shared Pool, der speziell für Ergebnismengen reserviert ist - den sogenannten **Result Cache**. Ergebnisse von Statements, die den Result Cache nutzen, werden bei der Ausführung im Result Cache abgelegt und bei den Folgeausführungen wiederverwendet. Die Ausführungszeit reduziert sich dabei drastisch. Um konsistente Abfragen zu garantieren, wird bei Änderungen an den Tabellenwerten der Cache automatisch invalidiert. Statements bzw. PL/SQL Funktionen, die

sich häufig wiederholen und deren Ergebnisse nur mit aufwändigen Mitteln d.h. mit hohem Ressourcenaufwand und Ausführungszeiten zu realisieren sind, können nun einfach vom Result Cache profitieren. Der Result Cache kann übrigens client- oder serverseitig Verwendung finden. Eine clientseitige Verwendung würde die Nutzungvon OCI Calls voraussetzen.

Der Einsatz ist besonders in folgenden Fällen von Vorteil:

- Langlaufenden und rechenintensive SQL-Abfragen
- Rechenintensive PL/SQL-Funktionen
- Vorhersehbare SQL-Abfragen
- Gleichbleibende deterministische Ergebnismengen
- Kleine Ergebnismengen bzw. ausreichendes Memory
- Geringe DML-Aktivität auf den zugrundeliegenden Tabellen

Um das Konzept zur erklären, konzentrieren wir uns hier auf den serverseitigen Result Cache.

Der Result Cache wird von den folgenden vier Parametern beeinflusst:

- RESULT_CACHE_MAX_RESULT (Default bei 5%)
- RESULT_CACHE_MAX_SIZE (Default O/S abhängig)
- RESULT_CACHE_MODE (Werte: MANUAL/FORCE)
- RESULT_CACHE_REMOTE_EXPIRATION (Default 0)

Mit dem Parameter RESULT_CACHE_MAX_SIZE wird die Gesamtgrösse des reservierten Bereichs für den Result Cache im Shared Pool festgelegt. Dabei wird der Speicher für Ergebnisse von SQL-Abfragen UND auch für Ergebnisse von PL/SQL-Funktionen reserviert. Wird dieser Parameter auf den Wert 0 gesetzt, ist der Result Cache ausgeschaltet. Der Defaultwert ist in der Regel allerdings ungleich 0. RESULT_CACHE_MAX_RESULT legt den prozentualen Anteil am gesamten Result Cache für die einzelnen Ergebnisse fest. Beide Parameter benötigen das ALTER SYSTEM- Privileg. Wird auf Remote Objekte zugegriffen, kann mit dem Parameter RESULT_CACHE_REMOTE_EXPIRATION festgelegt werden, wie lange das Resultat in Minuten im Cache verbleibt. Dieser Parameter ist mit ALTER SESSION oder ALTER SYSTEM einstellbar.

Einschalten läßt sich der Result Cache über die Session Einstellung FORCE mit dem Parameter RESULT_CACHE_MODE, über das Tabellenattribut RESULT_CACHE (MODE FORCE) oder feingranular über den Hint RESULT_CACHE.

```
ALTER SESSION SET result_cache_mode=force;

SELECT a.department_id         "Department",
       a.num_emp/b.total_count "%_Employees",
       a.sal_sum/b.total_sal   "%_Salary"
FROM (
  SELECT department_id,
  COUNT(*)       num_emp,
  SUM(salary)    sal_sum
```

```
 FROM employees
 GROUP BY department_id) a,
(
SELECT
  COUNT(*)    total_count,
  SUM(salary) total_sal
 FROM employees
 ) b
ORDER BY a.department_id;
```

Die Operation "RESULT CACHE" im Ausführungsplan zeigt an, dass der Result Cache erzeugt bzw. bei wiederholter Ausführung genutzt wird. Handelte es sich dabei um eine wiederholte Ausführung, hat sich die Ausführungszeit stark reduziert, da beispielsweise keine „Logical Reads“ (hier „consistent gets“) mehr erforderlich sind.

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation             | Name                         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                              |    11 |   715 |     7  (15)| 00:00:01 |
|   1 |  RESULT CACHE         | 4804bju78byhk5gkz1r3u2w301 |       |       |            |          |
|   2 |   SORT ORDER BY       |                              |    11 |   715 |     7  (15)| 00:00:01 |
|   3 |    NESTED LOOPS       |                              |    11 |   715 |     7  (15)| 00:00:01 |
|   4 |     VIEW              |                              |     1 |    26 |     3   (0)| 00:00:01 |
|   5 |      SORT AGGREGATE   |                              |     1 |     4 |            |          |
|   6 |       TABLE ACCESS FULL| EMPLOYEES                    |   107 |   428 |     3   (0)| 00:00:01 |
|   7 |     VIEW              |                              |    11 |   429 |     4  (25)| 00:00:01 |
|   8 |      SORT GROUP BY    |                              |    11 |    77 |     4  (25)| 00:00:01 |
|   9 |       TABLE ACCESS FULL| EMPLOYEES                    |   107 |   749 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Result Cache Information (identified by operation id):
------------------------------------------------------

   1 - column-count=3; dependencies=(HR.EMPLOYEES); name="SELECT a.department_id         "Department",
       a.num_emp/b.total_count "%_Employees",
       a.sal_sum/b.total_sal   "%_Sal"


Statistics
----------------------------------------------------------
         0  recursive calls
         0  db block gets
         0  consistent gets
         0  physical reads
         0  redo size
```

Folgendes Beispiel soll die Verwendung in einer PL/SQL Funktion demonstrieren. Die Result Cache Funktion lässt sich nur im PL/SQL aktivieren und ist somit dem Entwickler vorbehalten. Wichtig zu wissen ist, dass ein Zusammenhang zwischen der anzahl der verschiedenen Eingabeparametern und der Anzahl der Result Caches besteht.

```
CREATE OR REPLACE FUNCTION get_datum (
p_id NUMBER, p_format VARCHAR2
)
RETURN VARCHAR2 RESULT_CACHE IS
  v_datum DATE;
BEGIN
   select hiredate into v_datum from emp where empno = p_id;
   RETURN TO_CHAR(v_datum, p_format);
END;
```

Monitoren lässt sich der Status und die Verwendung der Result Caches sehr gut über die Views V$RESULT_CACHE_OBJECTS und V$RESULT_CACHE_STATISTICS. Mit dem

Package DBMS_RESULT_CACHE lässt sich ein Memory Report erzeugen, die Result Caches invalidieren, flushen oder sogar die Result Caches umgehen.

Der neue Cache in 12c – In-Memory Column Store
Wie zu Beginn schon erläutert, werden die Daten traditional im Buffer Cache im sogenannten Row Format gespeichert. Zeilenformate (Row Formate) sind ideal für Online Transaktionen, da sie schnelle Zugriffe auf alle Spalten einer Zeile gewähren. Beim Spaltenformat werden die Spalten hingegen in einer speziellen Spaltenstruktur gespeichert. Ideal ist diese Speicherform für analytische Abfragen, in denen wenige Spalten aber eine große Datenmenge abgefragt werden. Was passiert aber bei DML Abfragen? Beim Zeilenformat kann eine Zeile schnelle und effizient mit einer Operation verarbeitet werden, Spaltenformate hingegen eignen sich nur wenig für zeilenbasierte Operationen. Die neue **Oracle Database In-Memory Option** bietet hier das beste von beidem: **Spaltenformat** Ablage, der Column Store, in einem effektiven komprimierten Format für bestimmte ausgewählte Objekte um analytischen Abfragen zu beschleunigen und simultane Speicherung im Buffer Cache sobald DML Operationen oder andere Operationen dies erfordern.

Das Konfigurieren und die Nutzung ist recht einfach. Wie bei den anderen Pools, muss ein Bereich dafür festgelegt werden. Dies geschieht mir dem Parameter INMEMORY_SIZE. Wichtig zu wissen ist, dass der Column Store ein statischer Pool ist; das bedeutet, die Datenbank muss bei Änderung des Pools neu gestartet werden. Der Column Store besteht dabei aus zwei verschiedenen Poolarten: der 1 MB Pool für die Daten und der 64K Pool für die Metadaten.

```
SQL> select pool, alloc_bytes, used_bytes, populate_status
     from v$inmemory_area;

POOL                      ALLOC_BYTES USED_BYTES POPULATE_STATUS
------------------------- ----------- ---------- -------------------------
1MB POOL                    854589440  674234368 DONE
64KB POOL                   201326592    5832704 DONE
```

Wie kommen die Daten in den Pool? Die ausgewählten Objekte (Tabellen, Materialisierte Views oder Partitionen) werden über ein CREATE oder ALTER Kommando mit speziellen Attributen belegt, die die Speicherung im Column Store (INMEMORY), die Komprimierungsart (MEMPCOMPRESS) und die Priorität (PRIORITY) der Speicherung festlegen.

```
ALTER TABLE customers INMEMORY MEMCOMPRESS FOR QUERY LOW PRIORITY critical;
```

Bitte beachten Sie, dass die verschiedenen In-Memory Komprimierungsarten zusätzlich für verschiedene Charakteristiken von Workloads optimiert sind. Dies bedeutet, dass nicht allein die Einsparung von Speicherplatz sondern auch die Art der Anwendung eine Rolle bei der Wahl des richtigen In-Memory Komprimierungstyps spielt. "MEMCOMPRESS FOR QUERY LOW" liefert beispielsweise höhere Platzeinsparungsraten als "MEMCOMPRESS FOR DML". Allerdings sollte der Komprimierungstyp "MEMCOMPRESS FOR DML" bei hohen Datenmanipulationsraten immer dem Komprimierungstyp "MEMCOMPRESS FOR QUERY LOW" vorgezogen werden.

Danach werden die Daten je nach Priorität entweder von einem Hintergrund Prozess oder nach einem ersten Zugriff im Pool gespeichert. Einmal gespeichert wird das Objekt nicht mehr verdrängt. Eine explizite Auslagerung mit dem entsprechenden ALTER Kommando ist natürlich immer möglich. Ausgezeichnet Überwachen lässt sich der Inhalt des Column Stores beispielsweise auch über die View V$IM_SEGMENTS.

```
SQL> SELECT v.segment_name     name,
                    v.partition_name partition,
                    v.bytes/1024/1024       orig_M,
                    v.inmemory_size/1024/1024  in_memem
                    v.inmemory_compression comp,
                    v.bytes_not_populated not_pop,
                    v.inmemory_priority prio
    FROM   v$im_segments v

NAME        PARTITION             ORIG_M  IN_MEM_M COMP          NOT_POP PRIO
----------- -------------------- -------- --------- ------------- ------- ----
...
SALES       SALES_Q2_1999              8      1.125 FOR QUERY LOW       0 NONE
SALES       SALES_Q1_1999              8      1.125 FOR QUERY LOW       0 NONE
SALES       SALES_Q3_2001              8      1.125 FOR QUERY LOW       0 NONE
SALES       SALES_Q1_2001              8      1.125 FOR QUERY LOW       0 NONE
CUSTOMERS                             13      4.125 FOR QUERY LOW       0 CRITICAL
CUSTOMER                              64      5.125 FOR QUERY LOW       0 CRITICAL
SALES       SALES_Q4_2001              8     5.1875 FOR QUERY LOW       0 NONE
```

Wie bei den anderen Memory Techniken ist auch hier keine Änderung an der Applikation erforderlich. Wenn der Zugriff im Column Store sinnvoll ist, wird der Optimizer automatisch – sogar mit speziell für den Column Store entwickelten Operationen – die Abfrage durchführen. Spezielle Scan-Techniken und die Verwendung von **SIMD Operationen** (Single Instruction for Multiple Data values), die durch die aktuelle Mikroprozessortechnik möglich wird, führen dabei zu extrem schnellen Resultaten und beschleunigen die Abfragen um Faktoren. Für Testzwecke eignet sich dabei der Session Parameter INMEMORY_QUERY, mit dem man den Zugriff auf den Column Store erlauben oder verhindern kann.

Es macht natürlich nur Sinn, ein Objekt im Column Store zu platzieren, wenn ausreichend Platz zur Verfügung steht. Eine Warnung in der Alert Datei wird ausgegeben, falls dies nicht der Fall ist. Der Compression Advisor (DBMS_COMPRESSION) kann bei der Vorausberechnung der Größenordnung im Column Store helfen. Zusätzlich dazu existiert seit Kurzem auch ein spezieller In-Memory Advisor (siehe My Oracle Support Note), der die Entscheidung welche Objekte überhaupt zur Speicherung im Column Store geeignet sind, erleichtern kann.

Fazit

Dieser Artikel hat sich auf die bekannten und die häufig eingesetzten Aspekte der Memory Techniken konzentriert und stellt nicht den Anspruch auf Vollständigkeit. Da die meisten Techniken nur kurz beschrieben wurden, empfiehlt sich die nachfolgende Lektüre der Handbücher oder der exzellenten White Paper. Darüberhinaus ist zu erwarten, dass sich die Technologie weiterentwickeln, so dass dieser Artikel sicher nach einer Fortsetzung verlangt.

Zusammenfassend sollte folgendes deutlich werden. Alle Techniken können und sollen ergänzend Verwendung finden, damit optimale Performance für die Datenbankanwendungen erreicht werden können. Beispielsweise spricht nichts dagegen die Oracle Database In-

Memory Technik mit der Result Cache Technik zu kombinieren, falls unterschiedliche Abfragen davon profitieren können. Die Abfragen müssen in der Regel auch nicht angepasst werden. Ausnahme sind der PL/SQL Result Cache bzw. das explizite Ausschalten oder Einschalten beim Testen einer bestimmten Abfrage. Tabellenattribute, Sessioneigenschaften oder einfach nur das Bereitstellen einer Technik reicht im Normalfall zum Einsatz aus. Alle weiteren Datenbankoptionen und -mechanismen wie z.B. im Bereich Security, Hochverfügbarkeit usw. werden wie gewohnt unterstützt. Auch das optimierte Zusammenspiel mit den Technologien aus dem Data Warehouse und Analytics Umfeld wie Partitionierung und Parallelisierung ist garantiert. Die Verwaltung und Verwendung einer Oracle Datenbank unterscheidet sich also nicht von den bislang genutzten Methoden. Die Arbeit der Administratoren bzw. der Anwendungsentwickler ändert sich nicht, kein Mehraufwand ist zu erwarten – die komplette Oracle Infrastruktur bleibt ohne Modifikation erhalten. Also einfach am Besten ausprobieren!

Weitere Informationen

My Oracle Support
- In Memory Advisor Download (MOS Note 1965343.1)

Blogs
- Oracle Data Warehouse Blog: https://blogs.oracle.com/datawarehousing/
- In-Memory DB Blog: https://blogs.oracle.com/In-Memory/
- Deutschsprachige DB Community**:** http://blogs.oracle.com/dbacommunity_deutsch

Oracle White Paper
- Oracle Database In-Memory (July 2015)
- Parallel Execution with Oracle Database 12c Fundamentals (December 2014)

Handbücher
- Concepts Guide
- Database Performance Guide
- Database PL/SQL Language Reference
- Data Warehousing Guide

Kontaktadresse
Ulrike Schwinn
Oracle BU DB – Business Unit Database

ORACLE Deutschland B.V. & Co. KG
Riesstr 25, 80992 München
Telefon: +49 89 1430 1865
E-Mail Ulrike.Schwinn@oracle.com
Internet: http://blogs.oracle.com/dbacommunity_deutsch
https://twitter.com/uschwinn