ORACLE

ORACLE DOJO NR. 12

ULRIKE SCHWINN, MARKUS KISSLING

Oracle Database (In-)Memory-Technologien

Oracle Dojo ist eine Serie von Heften, die Oracle Deutschland B.V. zu unterschiedlichsten Themen aus der Oracle-Welt herausgibt.

Der Begriff Dojo [‘do:dʒo] kommt aus dem japanischen Kampfsport und bedeutet Übungshalle oder Trainingsraum. Als „Trainingseinheiten“, die unseren Anwendern helfen, ihre Arbeit mit Oracle zu perfektionieren, sollen auch die Oracle Dojos verstanden werden. Ziel ist es, Oracle-Anwendern mit jedem Heft einen schnellen und fundierten Überblick zu einem abgeschlossenen Themengebiet zu bieten.

Im *Oracle Dojo Nr. 12* beschäftigen sich Ulrike Schwinn, Senior Leitende Systemberaterin, und Markus Kissling, Leitender Systemberater, mit den wichtigsten In-Memory-Technologien der Oracle-Datenbank und vermitteln einen Überblick über deren vielfältige Einsatzmöglichkeiten.

ORACLE®

Inhalt

ORACLE

ORACLE DOJO NR. 12

ULRIKE SCHWINN, MARKUS KISSLING

Oracle Database (In-)Memory-Technologien

VORWORT DES HERAUSGEBERS

Vor einigen Jahren wurde ich gefragt, welche Oracle Datenbankfunktion mir am besten gefällt. Irgendwie eine komische Frage, dachte ich noch, doch schon war die Antwort ohne weiteres großes Nachdenken raus: „Den **Oracle Result Cache** finde ich einfach cool“. Bei der Nutzung der Result-Cache-Funktion werden die Ergebnisse einer SQL-Anfrage in einen Hauptspeicherbereich, den Result Cache, eingetragen und nachfolgende SQL-Statements nutzen diese Ergebnisse immer dann, wenn es passt. Es wird nicht mehr neu gelesen, optimiert oder gerechnet, sondern das bereits fertige Ergebnis angezeigt oder aber für weitere Operationen genutzt.

Schneller geht es nicht!

Einfach zu nutzen und hat bei passendem Anwendungsprofil erstaunlich positive Auswirkungen. Quasi eine Art Beschleunigungsknopf, eine Art Nachbrenner. Aber ein recht gut gehütetes Geheimnis! Der Result Cache ist eine von vielen Hauptspeicheroptimierungen, die Oracle im Laufe der Jahre in die Datenbank eingebaut hat, um die immer größeren (und billigeren) Hauptspeicher besser nutzen zu können. Die optimale Nutzung der Ressource „Hauptspeicher“ ist für Datenbanken schon immer von fundamentaler Bedeutung, nicht erst seit der In-Memory-Hype die Datenbankwelt durchgeschüttelt hat. Doch manchmal ist ein solches „durchschütteln“ ganz heilsam.

Unser Oracle In-Memory-Projekt, das einen integrierten Column Store zum Ziel hatte, hat davon auf alle Fälle profitiert und wurde dadurch enorm vorangetrieben. Heute kann die Oracle In-Memory-Option auf allen HW-/Betriebssystemen genutzt werden, auf denen die Oracle Enterprise Edition zur Verfügung steht. Es ist keine spezifische Hardware oder ein bestimmtes Betriebssystem notwendig.

Ich kann allen Lesern nur empfehlen diese Option zu testen. Das Ganze ist in wenigen Minuten aufgesetzt, die SQL-Befehle und die Anwendungen müssen nicht geändert werden, und die Ergebnisse liegen schnell auf dem Tisch. Und als Sahnehäubchen: In-Memory Column Store und Result Cache gemeinsam nutzen. Einfach unschlagbar!

Ich freue mich sehr, dass meine geschätzten Kollegen Ulrike Schwinn, Senior Leitende Systemberaterin, und Markus Kissling, Leitender Systemberater, in diesem *Dojo Nr. 12* die wichtigsten In-Memory-Technologien zusammengestellt und für Sie aufbereitet haben.

Ich wünsche Ihnen viel Spaß beim Lesen und beim Testen.

Ihr Günther Stürner
Vice President Sales Consulting

PS: Wir sind an Ihrer Meinung interessiert. Anregungen, Lob oder Kritik gerne an barbara.frank@oracle.com. Vielen Dank!

Zugriff auf alle Dojos: http://tinyurl.com/dojoonline

1 Einführung

Datenbank Memory effizient einzusetzen ist eine der „billigsten" Methoden, um Datenbanken zu tunen. Was ist günstiger als die vorhandenen Ressourcen richtig zu nutzen? *Dojo Nr. 12* erklärt einige wichtige Memory-Technologien der Oracle-Datenbank und soll einen Überblick über ihre vielfältigen Einsatzmöglichkeiten geben. Nicht nur das aktuelle Datenbank-Release, sondern auch vorangegangene Releases werden in den folgenden Abschnitten berücksichtigt. Die Verwendung ist in der Regel nicht isoliert zu betrachten: Zögern Sie nicht, mehrere Techniken gleichzeitig in Ihrer Datenbank zur Anwendung zu bringen.

Wie haben wir das Dojo aufgebaut? Wir starten zuerst einfach mit ein paar wesentlichen Fakten. Oracle bietet schon seit jeher spezielle Technologien an, um Daten effizient im Memory zu speichern. Effektive Einlagerung im Datenbank-Cache wird dabei „groß geschrieben", sodass beispielsweise „heiße Daten" im schnellen Zugriff sind. Um eine Verdrängung aus dem Buffer Cache zu verhindern, kann sogar ein spezieller Keep Pool reserviert werden. Ein weiterer separater Bereich, der nicht block- sondern segmentweise beim Full Table Scan die Daten vorhält, gehört zu den neuesten Entwicklungen in diesem Umfeld. Handelt es sich um eine eher kleinere Datenbank, die vollständig in den Cache

passen würde, ist sogar ein Full Database Caching im aktuellen Datenbank-Release möglich.

Damit nicht genug: Um bestimmte Applikationen beziehungsweise Abfragen noch besser bei der Ablage im Cache zu unterstützen, sind zusätzliche Techniken implementiert worden. So können sogar Resultate von sich wiederholenden Statementausführungen oder PL/SQL-Funktionen in einem speziellen Cache vorgehalten werden, um kostenintensive, sich wiederholende Abfragen und PL/SQL-Funktionen zu beschleunigen.

Geht man noch einen Schritt weiter und ist man an einer Optimierung von beispielsweise analytischen Abfragen interessiert, sollte man die Oracle Database In-Memory-Technologie ab Database Release 12c testen. Diese zeichnet sich nicht nur durch eine neue Form der Cache-Ablage aus – nämlich spaltenbasiert und zusätzlich komprimiert –, sondern profitiert zusätzlich von speziellen optimierten Statementzugriffen. Mit dieser Technik können außerordentliche Performancegewinne erzielt werden, wie Kunden uns nach einer ersten Testphase mit folgenden Worten bestätigten: „Das ist der helle Wahnsinn, ein absoluter Quantensprung. Solche Ergebnisse haben wir noch nie gehabt und das bereits auf den alten Rechnern!“

Zum Schluss vielleicht noch eine Anmerkung, die auf alle gerade genannten Techniken zutrifft: Keine zusätzliche Installation und kein zusätzlicher Layer ist erforderlich – die Funktionen

stehen in der Regel ohne großen Aufwand nach wenigen Schritten zur Verfügung. Die Verwendung von speziellen Tabellenattributen beziehungsweise Session-Eigenschaften oder nur die Bereitstellung über Initialisierungsparameter reichen im Normalfall aus. Alle weiteren Datenbankmechanismen werden wie gewohnt unterstützt. Für die Applikationen selbst ist übrigens in der Regel keine Anpassung erforderlich, alles ist völlig „transparent". Übrigens können Sie die beschriebenen Techniken auf allen Plattformen, auf denen die Oracle-Datenbank verfügbar ist, nutzen – egal ob Linux, Unix, Windows, HP UX, zLinux, AIX oder auch BS2000 als Betriebssystem verwendet wird. Möchte man die Oracle-Database-Cloud-Service-Angebote nutzen, wird man auch hier schnell fündig werden.

Bevor Sie sich mit den In-Memory-Techniken der Oracle-Datenbank beschäftigen, noch ein Tipp: Vergessen Sie nicht, zuerst das gesamte Datenbank-Memory im Verhältnis zum gesamten zur Verfügung stehenden Memory des Servers zu setzen und zu evaluieren. So stellen Sie sicher, dass das Datenbank-Memory ausreichend dimensioniert ist.

2 Ein paar Grundlagen

Starten wir mit ein paar grundsätzlichen Informationen, die Ihnen sicherlich aus Ihren 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 die unterschiedlichsten Aufgaben erfüllen. Um zwei Beispiele zu nennen: Der Database Buffer Cache stellt den Teil der SGA dar, der die Kopien der Datenbankblöcke zwischenspeichert. Alle User, die mit einer Instance verbunden sind, teilen sich diesen Bereich. Der Shared Pool hingegen setzt sich zusammen aus Library Cache für gemeinsame SQL- und PL/SQL-Bereiche, Dictionary Cache, um Informationen über Tabellen und User zu referenzieren, Result Cache für die Ergebnisse aus SQL-Statements und PL/SQL-Funktionen und weiteren Kontrollstrukturen.

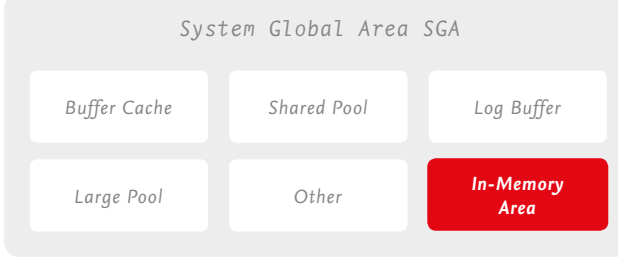


Abb. 1: Wichtige SGA-Komponenten im Überblick

Im Laufe der Jahre sind weitere Bereiche zur SGA hinzugefügt worden, um das Funktionsfeld zu erweitern. Abbildung 1 liefert einen Einblick über die wichtigsten Bestandteile der SGA.

Betrachten wir zuerst den Buffer Cache. Für die meisten Operationen innerhalb der Oracle-Datenbank wird der Buffer Cache zur Zwischenlagerung von Datenblöcken, die von Platte gelesen worden sind, verwendet. Ausgenommen sind dabei Operationen, die über einen Direct Path Read verarbeitet werden. Dies sind beispielsweise parallele oder serielle Zugriffe auf große Tabellen.

In welcher Form und wie lange werden die Daten im Buffer Cache gespeichert? Im Buffer Cache werden Daten in Datenbankblöcken gehalten. Die Informationen/Daten in den Blöcken liegen dabei standardmäßig im Zeilenformat vor; es wird weder komprimiert noch umsortiert. Eine Ausnahme dazu bildet der Column Store (siehe Kapitel 6: Oracle Database In-Memory). Da die Verwaltung der Buffer effizient sein muss, wird ein Least-Recently-Used-Algorithmus (kurz: LRU-Algorithmus) bei der Verwaltung der Blöcke im Buffer Cache verwendet. Blöcke, die ständig in Verwendung sind (Hot Blocks) werden an das obere Ende der Liste platziert, damit sie möglichst lange im Buffer Cache verweilen, selten verwendete Blöcke hingegen werden am unteren Ende der Liste gelagert. Neu mit Oracle Database 12c ist der

„temperaturbasierte“ Algorithmus auf Objektebene, der beim Automatic Big Table Caching Feature verwendet wird (mehr dazu im Kapitel 3: Automatic Big Table Caching).

Ein automatisches Memory-Management kann über den Initialisierungsparameter SGA_TARGET, mit dem die Gesamtgröße der SGA festgelegt wird, eingeschaltet werden. Automatisch werden dann die Memory-Bereiche wie Buffer Cache, Shared Pool, Large Pool und Java Pool allokiert. Empfehlenswert ist 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 VDB_CACHE_ADVISE, VMEMORY_CACHE_ADVISE beziehungsweise V$SGA_TARGET_ADVICE, die auch grafisch im Enterprise Manager Cloud Control oder Enterprise Manager Database Express zur Verfügung stehen, geben dabei die Auslastung für die aktuelle Buffergröße und für eine hypothetische Verkleinerung beziehungsweise Vergrößerung (auch Extrapolation) des Pools an. Folgendes Beispiel zeigt die Nutzung der Advisor View V$SGA_TARGET_ADVICE.

```
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
```

In unserem Beispiel würde eine Vergrößerung der SGA um das 1.5-Fache zu einer Verminderung der Datenbankzeit (DB Time) und der Physical Reads führen (siehe Zeile 5).

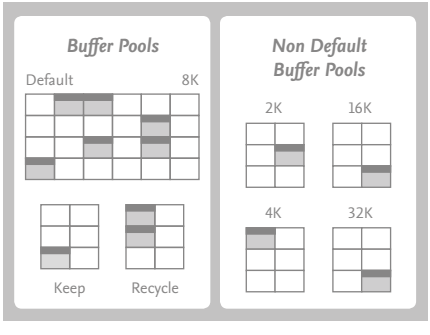


Abb. 2: Standard und Nicht-Standard Buffer Pool

Hat man einen guten Überblick über das Zugriffsverhalten (Workload) auf die Objekte, 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 Parametern DB_KEEP_CACHE_SIZE beziehungsweise 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 Ein- beziehungsweise Auslagerung der Blöcke im Keep oder Recycle Pool wird wie beim Default Pool über den LRU-Algorithmus vorgenommen.

Darüber hinaus gibt es generell die Möglichkeit, einen Non Default Buffer Pool einzurichten. Das ist immer dann erforderlich, wenn Tablespaces mit einer Nicht-Default-Buffergröße angelegt werden sollen, um beispielsweise für bestimmte Objekte eine bessere Compression Ratio zu erzielen. Vorab muss dann ein entsprechender Memory-Bereich mit Parametern der Form DB_CACHE_nK_SIZE (n steht dann für eine Zahl wie 32, 16 etc.) konfiguriert werden.

2.1 MONITORING UND TUNING

Speziell während des Tunings 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 Statistiken wie Physical Reads Hinweise darauf erhalten, ob und wie viele Plattenzugriffe für die Ausführung erforderlich waren.

Bezogen auf die Blöcke eines Objekts kann man auch die Fixed Table V$BH zur Analyse heranziehen. Sie gibt Auskunft über den Status 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. Die Tabelle selbst besteht aber aus 71424 Blöcken, wie man leicht aus DBA_SEGMENTS erkennen kann.

```
SQL> SELECT blocks FROM dba_segments WHERE segment_name='SALES_COPY';
   BLOCKS
---------
    71424
```

Womit hängt dies zusammen? Warum sind nicht alle Blöcke der Tabelle im Buffer Cache? Bei Full-Table-Scan-Operationen nutzt die Oracle-Datenbank einen internen Algorithmus, um zu entscheiden, ob Blöcke im Buffer Cache zwischengelagert werden oder ob Direct Path Reads durchgeführt werden. Bei einer Direct-Path-Read-Operation werden die Blöcke direkt unter Umgehung des Buffer Caches gelesen. Abhängig von der Größe des Objekts – falls ein Objekt als groß (LARGE) angesehen wird – wird ein Direct Path Read durchgeführt. Die Idee dabei ist, den Buffer Cache nicht unnötig mit wenigen großen Objekten völlig auszulasten und zusätzlich mit vielen kleinen Reads die fehlenden Blöcke in den Buffer Cache zu laden. Ist die Tabelle hingegen klein (SMALL) wird beim Zugriff der Buffer Cache verwendet. Intern wird dazu der Parameter _small_table_threshold verwendet; er legt die Grenze für die Größe von kleinen (SMALL) Tabellen fest. In der Regel sind dies ungefähr 2 Prozent der gesamten Größe des Buffer Cache. Soweit die wichtigsten Regeln für die Nutzung des Buffer Cache.

Möchte man sicherstellen, dass Tabellen im Buffer Cache zwischengelagert werden, kann man, wie im Abschnitt zuvor schon besprochen, auch den Keep Pool verwenden. Eine weitere Methode wäre den Parameter _small_table_threshold beispielsweise auf Session-Ebene zu setzen, um für nachfolgende Abfragen einen eigenen

Wert festzulegen. In unserem Beispiel wäre der Wert 72000 ausreichend, um zu bewirken, dass die Tabelle SALES_COPY über den Buffer Cache geladen wird.

```
SQL> alter session set "_small_table_threshold" = 72000;
```

In Oracle Database 12c werden darüber hinaus weitere interne Kriterien hinzugezogen, um zu bestimmen, welche Tabellen über den Cache zwischengelagert werden.

3 Automatic Big Table Caching

Um unabhängig von diesem gerade beschriebenen Standardverhalten des Buffer Cache bei parallelen und seriellen Full Table Scans zu sein, ist in Oracle Database 12c ein neuer Cache-Bereich im Buffer Cache eingeführt worden – der sogenannte **Automatic Big Table Cache** (kurz: ABTC).

Die Idee dahinter ist einfach: Ein gewisser Teil des Buffer Cache wird für die Speicherung großer Objekte oder Teile davon reserviert, sodass 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 des Big Table Cache 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 die **Temperatur der Objekte** (nicht Blöcke) bei der Nutzung des Big Table Caches die wesentliche 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 Set-up 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 die Verwendung des Features 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 Cache wird für Scans und damit für den Big Table Cache reserviert; 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                  4               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 im 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 infrage. 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:

- Für das parallele Arbeiten muss PARALLEL_DEGREE_POLICY den Wert AUTO oder ADAPTIVE (neu in 12c) haben.
- Wie schon erklärt muss DB_BIG_TABLE_CACHE_PERCENT_TARGET gesetzt sein, um den Anteil am Database Buffer Cache (in Prozent) festzulegen.

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

4 Die gesamte Datenbank im Cache? – Full Database Caching

Wie schon zu Beginn beschrieben, entscheidet die Datenbank selbst in Abhängigkeit von Größe und Art der Nutzung des Objekts, wie der Cache verwendet wird. Der Cache unterliegt dabei unterschiedlichen Algorithmen, was die Dauer und die Lagerung der Blöcke im Buffer Cache angeht. So werden beispielsweise die Blöcke nach einem Least-Used-Algorithmus im Cache gehalten, um die Verweildauer im Cache effizient zu gestalten oder sie können nach der Temperaturmethode ausgelagert werden. Ein Full Table Scan (parallel oder seriell) auf kleinen (SMALL) Tabellen wird dabei im Buffer Cache realisiert; ein Full Table Scan auf großen (LARGE) Tabellen hingegen über Direct-Read-Operationen. Bei mittelgroßen Tabellen bestimmt ein interner Algorithmus, ob über Direct Path Reads oder über Buffer Cache gearbeitet wird. Large Objects (auch LOBs) – also SecureFiles oder BasicFiles (in der 12c-Terminologie) – werden übrigens standardmäßig nicht im Cache gespeichert. Die Speicherung eines LOBs im Cache erfolgt nur unter der Bedingung, dass explizit vorab das Speicherattribut mit einem DDL-Befehl auf CACHE gesetzt wurde.

Neu in 12c ist ein automatischer **Full Database Caching Mode**. Falls das Memory ausreichend für die gesamte Datenbank ist und eigene interne Regeln erfüllt sind, werden alle Tabellen als kleine (SMALL) Tabellen angesehen und im Cache gelagert. Im Oracle Database Concepts Guide steht hierzu:

> *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 NO-CACHE attribute.*

Möchte man unabhängig von diesen Regeln sein, kann man diesen Mode auch forcieren. Das Konzept nennt sich dann Force Full Database Caching. Die Idee dahinter ist im Prinzip die gleiche wie beim automatischen Full Database Caching: Ist der Cache groß genug, um alle Objekte im Cache zu speichern, können alle Objekte im Cache gelagert werden – allerdings einschließlich der LOB-Objekte. Um den Full Database Mode zu nutzen, muss dieser zusätzlich im MOUNT-Stadium der Datenbank separat eingeschaltet werden. Dies bedeutet

auch, dass die Information im Control File gespeichert wird. Daher ist es sinnvoll, direkt nach dem Umsetzen des Modus ein Backup vom Control File zu erstellen.

```
SQL> startup mount;
SQL> ALTER DATABASE FORCE FULL DATABASE CACHING;
Database altered.
-- Ausschalten mit
-- ALTER DATABASE NO FORCE FULL DATABASE CACHING;
SQL> SELECT force_full_db_caching FROM v$database;
FOR
---
YES
```

In einer Multitenant-Umgebung bezieht sich diese Einstellung übrigens auf alle PDBs – eine „Container"-weite Entscheidung also.

Die Alert-Datei gibt dann Auskunft darüber, ob der Modus erfolgreich eingeschaltet werden konnte.

```
Completed: ALTER DATABASE  MOUNT
2015-12-15 02:56:03.396000 -08:00
alter database force full database caching
Completed: alter database force full database caching
```

Um Missverständnisse zu vermeiden: Nach der Konfiguration einer Datenbank im Force Full Database Mode werden die Objekte nicht automatisch in den Cache verlagert. Es muss zuerst ein Zugriff auf die Objekte erfolgen. Dabei werden auch LOBs mit der Einstellung NOCACHE geladen; dies ist beim normalen Caching nicht der Fall.

Bevor man Force Full Database Caching verwendet, sollte man sicherstellen, dass ausreichend Memory vorhanden ist. Bei Nutzung von MEMORY_TARGET oder SGA_TARGET kann sich die Größe des Buffer Cache ändern. Um dies zu verhindern gilt auch hier die Empfehlung, die eingangs schon gegeben worden ist: Verwenden Sie DB_CACHE_SIZE, um ein Minimum zu garantieren.

Häufig wird die Frage gestellt, was passiert, wenn die Datenbank wächst und die Speicherplatzanforderungen den Buffer Cache übersteigen? Wichtig zu wissen ist: Der Modus wird nicht automatisch umgeschaltet. Man wird wieder Physical Reads feststellen. Allerdings wird eine Information in der Alert-Datei wie folgt notiert:

```
Thread 1 opened at log sequence 44
Current log# 2 seq# 44 mem# 0: /home/oracle/app/oracle/oradata/test/
redo02.log
Successful open of redo thread 1
```

```
Fri Dec 18 02:20:54 2015
MTTR advisory is disabled because FAST_START_MTTR_TARGET is not set
Buffer Cache Force Full DB Caching mode on when DB does not fit in cache.
Turning off Force Full DB Caching advisable
```

Force Full Database Caching eignet sich hervorragend für kleine Datenbanken, deren genutzter Speicherplatz in den Buffer Cache passt. So kann man sehr einfach die Datenbank-Performance von Full Table Scans und Zugriffen auf Large Objects erhöhen. Keine Änderungen an den Abfragen oder Segmenten ist dazu erforderlich.

Bevor man diese Funktion verwendet, sollte man allerdings sicherstellen, dass ausreichend Memory vorhanden ist. Für RAC-Umgebungen bedeutet dies, dass der aktuell genutzte Datenbankspeicherplatz kleiner als der jeweilige (individuelle) Buffer Cache jeder Datenbank-Instance ist. Ist der Workload in der RAC-Umgebung ausreichend gut partitioniert (bzgl. Instance-Zugriff), kann der aktuell genutzte Datenbankspeicherplatz auch kleiner als 80 Prozent des kombinierten Buffer Caches sein.

Dieses Feature steht übrigens in allen Editionen zur Verfügung und ist somit auch gut geeignet für eher kleinere Datenbanken.

5 Der Ergebnis (Result) Cache

Seit Oracle Database 11g gibt es einen neuen Cache-Bereich im Shared Pool, der speziell für Ergebnismengen reserviert ist, den sogenannten Result Cache. Was steckt dahinter?

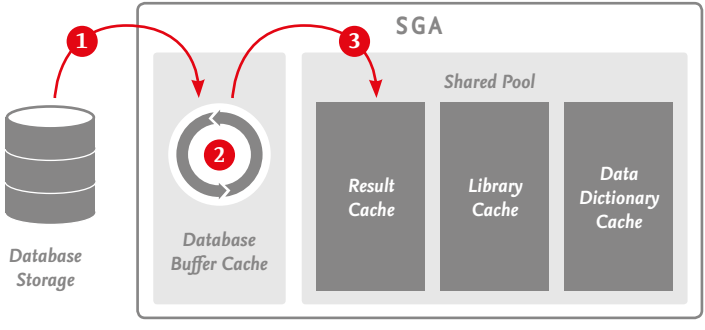


Abb. 3: Die Nutzung des Result Cache

Der Result Cache ist ein speziell für Ergebnisse aus SQL-Abfragen oder PL/SQL-Funktionen vorgesehener Cache. Die Datenbank verwaltet den Cache dabei völlig selbstständig und stellt dabei sicher, dass niemals veraltete Ergebnisse ausgegeben werden. Oder etwas genauer: Ergebnisse von Statements beziehungsweise Ergebnisse aus PL/SQL-Funktionen, die den Result Cache nutzen, werden bei der Ausführung im Result

Cache abgelegt und bei den Folgeausführungen wiederverwendet. Die Konsequenz ist eine drastisch reduzierte Ausführungszeit besonders bei Statements und PL/SQL-Funktionen, die sich häufig wiederholen und deren Ergebnisse nur mit aufwendigen Mitteln, das heißt mit hohem Ressourcenaufwand und Ausführungszeiten zu realisieren sind. Um konsistente Abfragen zu garantieren, wird bei Änderungen an den Tabellenwerten der Cache automatisch invalidiert. Der Result Cache kann übrigens client- oder serverseitig Verwendung finden. Eine clientseitige Verwendung setzt dabei die Nutzung von OCI Calls voraus.

Der Einsatz des Query beziehungsweise des PL/SQL Function Result Cache ist besonders in folgenden Fällen von Vorteil:

- bei langlaufenden und rechenintensiven SQL-Abfragen
- für rechenintensive PL/SQL-Funktionen
- für vorhersehbare SQL-Abfragen
- bei gleichbleibenden deterministischen Ergebnismengen
- für kleine Ergebnismengen
- bei geringer DML-Aktivität auf den zugrunde liegenden Tabellen

5.1 DIE KONFIGURATION

Wichtig zu wissen ist, dass der Query Result Cache und der PL/SQL Function Result Cache unterschiedlich konfiguriert werden. Für den ersten Fall sind Initialisierungsparameter beziehungsweise Hints oder Tabellenattribute erforderlich; der PL/SQL Function Result Cache wird vom Entwickler selbst in der entsprechenden PL/SQL-Funktion mitgegeben.

Folgende vier Initialisierungsparameter sind speziell zur Administration und Nutzung des Query Result Cache eingeführt worden:

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

Die Parameter sind gleich nach der Installation mit Default-Werten belegt. Somit kann der Cache sofort verwendet werden. Die Default-Werte lassen sich dynamisch mit dem ALTER SESSION- oder dem ALTER SYSTEM-Befehl verändern. Mit dem Parameter RESULT_CACHE_MAX_SIZE wird die Gesamtgröße des reservierten Bereichs für den Result Cache im Shared Pool festgelegt. Dabei wird der Speicherbereich 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. 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 Query Result Cache über die Session-Einstellung FORCE mit dem Parameter RESULT_CACHE_MODE, über das Tabellenattribut RESULT_CACHE (MODE FORCE) (bzw. MODE DEFAULT zum Ausschalten) oder feingranular über den Hint RESULT_CACHE.

```
SQL> ALTER TABLE <table_name> RESULT_CACHE (MODE FORCE);


SQL> SELECT result_cache FROM user_tables WHERE table_name like
'CUSTOMERS';
RESULT_
-------
FORCE
```

In der Session ließe sich Result Caching über folgenden Parameter einschalten:

```
ALTER SESSION SET RESULT_CACHE_MODE=FORCE;
```

Wichtig zu wissen ist, dass der Parameter RESULT_CACHE_MODE nur für den Top Level SELECT zum Tragen kommt; Sub Selects müssen separat über einen Hint beziehungsweise über das Tabellenattribut aktiviert werden.

Der PL/SQL Function Result Cache hingegen lässt sich nur innerhalb des PL/SQL Codes aktivieren und ist somit dem Entwickler vorbehalten. Folgendes Beispiel zeigt die einfache Verwendung im PL/SQL Code. In der RETURN-Syntax wird einfach nur das Schlüsselwort RESULT_CACHE mitgegeben.

```
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;
```

Bitte beachten Sie unbedingt ein paar wichtige Regeln:

- Der PL/SQL Function Result Cache kann nicht in anonymen Blöcken und mit Pipelined Table Functions verwendet werden.

- Dictionary Tables, Temporary Tables, Sequences oder nicht deterministische SQL Functions sind nicht referenzierbar.
- Die Funktion darf keine OUT oder IN-OUT-Parameter enthalten.
- IN-Parameter dürfen keine LOBs, REF CURSOR, Collections, Objekte oder Records darstellen.
- Der Return Typ ist kein LOB, REF CURSOR, Objekt oder Record.

5.2 DAS MONITORING

Wie kann man nun Result Caches überwachen? Woher weiß man, ob der Result Cache verwendet wird. Beim Query Result Cache liefert wie immer der Ausführungsplan einen schnellen Anhaltspunkt, wie folgendes Beispiel demonstriert:

```
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 für den Top-Select-Zweig erzeugt wird beziehungsweise bei wiederholter Ausführung auch verwendet wird. Handelt es sich dabei um eine wiederholte Ausführung, hat sich die Ausführungszeit stark reduziert, da beispielsweise keine Logical Reads (hier: consistent gets) mehr erforderlich sind (siehe Abb. 4):

Abb. 4: Ausführungsplan zu Result Cache

```
Execution Plan
----------------------------------------------------------

Plan hash value: 2024258137
-------------------------------------------------------------------------------------------------------
| Id  | Operation              | Name                       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |                            |    11 |   715 |     7  (15)| 00:00:01 |
|   1 |  RESULT CACHE          | 2a8h5ka5z4j5x6vumt9sz7jhbv |       |       |            |          |
|   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
```

Die View V$RESULT_CACHE_OBJECTS gibt darüber hinaus Aufschluss über die Nutzung der einzelnen Ergebnisse im Cache und ihre Abhängigkeiten. So kann man genau feststellen, wie häufig beispielsweise ein Cache-Objekt vom Typ Result oder Dependency verwendet worden ist, oder ob und wie häufig dieses invalidiert worden ist. Möchte man nun genau prüfen, welche Ergebnisse im Result Cache verwaltet werden, sollte man folgende Abfrage durchführen:

```
SQL> SELECT name, type, row_count, status, invalidations, scan_count
     FROM v$result_cache_objects;

NAME                 TYPE        ROW_COUNT STATUS      INVALIDATIONS   SCAN_COUNT
-------------------- ----------- --------- --------- --------------- ------------
HR.GET_DATUM         Dependency          0 Published               0            0
SCOTT.EMP            Dependency          0 Published               0            0
HR.BIGEMP            Dependency          0 Published               1            0
SELECT /*+ result_ca Result             12 Published               0            5
che */ COUNT(*) tota
l_count, SUM(salary)
total_sal
FROM hr.bigemp group
by department_id
ORDER BY department
```

```
NAME                    TYPE        ROW_COUNT STATUS     INVALIDATIONS   SCAN_COUNT
---------------------- ----------- --------- --------- --------------- -----------
"HR"."GET_DATUM"::8.    Result              1 Published               0
l"GET_DATUM"#27dda668
fe0cf492 #1
SELECT /*+ result_ca    Result             12 Invalid                 0           4
che */ COUNT(*) tota
l_count, SUM(salary)
total_sal
FROM bigemp group by
department_id ORDER
BY department_id
```

Die Ergebnisblöcke setzen sich offensichtlich aus der Abfrage SELECT /*+ result_cache */ COUNT(*) ... und der ausgeführten PL/SQL-Funktion HR.GET_DATUM zusammen. Dabei ist das Ergebnis der Abfrage offensichtlich vier beziehungsweise fünf Mal genutzt worden (siehe Spalte SCAN_COUNT). Invalidierungen, das heißt Änderungen an den abhängigen Objekten (hier an der Tabelle BIGEMP) sind erfolgt, wie in der Spalte INVALIDATIONS zu erkennen ist. Aus diesem Grund ist parallel eine weitere Zeile mit den Informationen des neuen Result Cache erzeugt worden. Diese besitzt den Status Published.

Änderungen an den Statements durch Hinzufügen der Schemabezeichnung oder einer WHERE-Klausel oder die Anwendung von verschiedenen Werten bei Bind-Variablen führen übrigens auch zur Erweiterung des Result Caches.

Neben den V$RESULT_CACHE_xxx Views gibt es zusätzlich das Package DBMS_RESULT_CACHE, das beim Testen beziehungsweise Monitoren hilfreich sein kann. So kann zum Beispiel mit DBMS_RESULT_CACHE.FLUSH() der Cache bereinigt (flush) werden beziehungsweise mit Prozedur INVALIDATE invalidiert werden. DBMS_RESULT_CACHE.MEMORY_REPORT() gibt zusätzlich einen Gesamtüberblick über die Verwendung der Blöcke.

5.3 OCI CLIENT SIDE RESULT CACHE

OCI-Treiber (wie OCCI, JDBC OCI und ODP.NET) können darüber hinaus den clientseitigen Result Cache verwenden. Dies ist ein Memory-Bereich innerhalb des Client-Prozesses, der die Resultate von SQL-Abfragen einer OCI-Applikation speichert. So können Roundtrips zum Server reduziert werden. Er existiert für jeden Client-Prozess und steht für alle Sessions innerhalb des Prozesses zur Verfügung. Client- und serverseitiger Result Cache sind dabei völlig getrennt und können unabhängig voneinander ein- und ausgeschaltet werden.

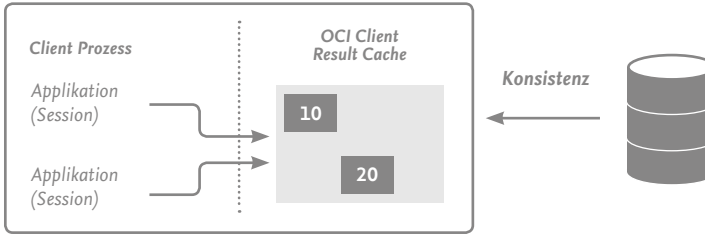


Abb. 5: Nutzung des clientseitigen Result Cache

Aktiviert wird der Client Result Cache über zwei zusätzliche Parameter – CLIENT_RESULT_CACHE_SIZE für die Größenordnung jedes Client-Prozesses und CLIENT_RESULT_CACHE_LAG für die sogenannte Lag Time (in Millisekunden) mit einem Default-Wert von 3000 (3 Sekunden). Die Nutzung kann, wie im Query Result Cache schon besprochen, über Hints, Tabellen-Annotationen oder Session-Einstellungen erfolgen.

Betrachten wir im Folgenden ein einfaches Beispiel mit CLIENT_RESULT_CACHE_SIZE von 32K. Der Default-Wert ist übrigens 0.

```
SQL> sho parameter client
```

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ---------------------
client_result_cache_lag              big integer 3000
client_result_cache_size             big integer 32K
```

Im OCI-Beispielcode (hier ein Ausschnitt) werden die Tabellen CUSTOMERS und SALES mehrfach in einer Schleife abgefragt.

```
stmt = conn.prepareStatement("select /*+ result_cache */ c.cust_id,
SUM(amount_sold) AS dollar_sales FROM sh.sales s,sh.customers c WHERE s.cust_
id=
c.cust_id and rownum < 1000 GROUP BY c.cust_id");
```

Überwachen lässt sich die Nutzung wie beim Server Result Cache über die Views V$RESULT_CACHE_OBJECTS und V$RESULT_CACHE_STATISTICS. Die spezielle View CLIENT_RESULT_CACHE_STATS$ (siehe unten) speichert darüber hinaus die Cache-Einstellungen und Memory-Nutzungsstatistiken für die Client Result Caches der einzelnen OCI-Client-Prozesse. Zu Beginn sieht das Ergebnis der Abfrage folgendermaßen aus.

```
SQL> SELECT stat_id, name, value, cache_id
     FROM client_result_cache_stat$
     ORDER BY cache_id, stat_id;
no rows selected
```

Nachdem das Programm einige Zeit läuft, erhalten wir folgendes Ergebnis.

```
SQL> SELECT stat_id, name, value, cache_id
     FROM client_result_cache_stat$
ORDER BY cache_id, stat_id;

   STAT_ID NAME                           VALUE   CACHE_ID
---------- ------------------------------ ------- ----------
         1 Block Size                         256      46447
         2 Block Count Max                   7936      46447
         3 Block Count Current                128      46447
         4 Hash Bucket Count                 1024      46447
         5 Create Count Success                 1      46447
         6 Create Count Failure                 0      46447
         7 Find Count                          50      46447
         8 Invalidation Count                   0      46447
         9 Delete Count Invalid                 0      46447
        10 Delete Count Valid                   0      46447
10 rows selected.
```

Der Cache mit CACHE_ID 46447 hat offensichtlich erfolgreich einen Result Cache gebildet. Find Count 50 gibt dabei an, dass der Cache 50 Mal verwendet worden ist. Falls Sie weitere Programme starten, erhalten Sie die Auflistung mit Informationen zu weiteren CACHE_IDs. Nachdem alle Client-Prozesse der Caches beendet worden sind, werden die Einträge von der Tabelle übrigens automatisch von einem Oracle-Hintergrundprozess gelöscht.

6 Oracle Database In-Memory

Die aktuellen Multicore-Prozessoren und Hauptspeicher-Größen im oberen Gigabyte- beziehungsweise Terabyte-Bereich ermöglichen einen neuen Architekturansatz bei Datenbanken und den darauf entwickelten Anwendungen. Dabei spielt der sogenannte In-Memory (IM) Column Store, der die Daten spaltenweise organisiert, die zentrale Rolle. Diesen haben die meisten Hersteller am Markt auf unterschiedliche Weise implementiert. Entweder erweitern sie dabei ihre bestehenden Datenbank-Produkte um einen derartigen IM Column Store oder sie haben komplett neue Produkte geschaffen. Jeder will dabei speziell im analytischen Umfeld die Datenverarbeitung von sehr großen Datenmengen selbst bei hoher Last beschleunigen, I/O-Zugriffe vermeiden und einen hohen Grad an Parallelisierung realisieren. Letztendlich geht es darum, Antwortzeiten zu erhalten, die früher nicht möglich waren.

Oracle nutzt diese hardwareseitigen Trends ausgesprochen elegant: die Oracle-Datenbank wird einfach um einen IM Column Store erweitert. In den Genuss kommen alle Anwender, die die Oracle Database 12c Enterprise Edition in der Version 12.1.0.2 mit der neuen Oracle Database In-Memory-Datenbank-Option einsetzen.

Innerhalb der Oracle-Datenbank stellt die neue Oracle Database In-Memory-Option die einzigartige Dual-Format-Architektur

zur Verfügung, die bei analytischen Workloads neue Größenordnungen bei der Performance-Verbesserung ermöglicht. Mithilfe der Vector Register der CPU-Kerne können Milliarden von Rows pro Sekunde verarbeitet werden, da sich hier die Taktung der CPU direkt auswirkt. Ebenso können durch das Eliminieren von analytischen Indizes, die bei Änderungen nicht mehr gepflegt werden müssen, bei Unternehmens-Workloads wie bei CRM, HCM und ERP signifikante Steigerungen erreicht werden. Auch SAP/Oracle-Kunden profitieren von Oracle Database In-Memory, da seit Juli 2015 SAP-Anwendungen auf Basis der SAP NetWeaver 7.x Technologie-Plattform zertifiziert sind (siehe SAP Note 2178980).

Das duale Format, das mit der Oracle Database In-Memory-Option eingeführt wurde, stellt sowohl das Row- als auch das Column-Format bereit und umgeht so die Kompromisse, die bei reinen Single-Format In-Memory-Datenbanken zwangsläufig existieren. Während das bekannte Row-Format ideal im Rahmen der Transaktionsverarbeitung im OLTP-Umfeld ist, hat das Column-Format seine Vorteile im OLAP-/Analytics-Bereich. Bei reinen In-Memory-Datenbanken müssen fehlende Formate in der Regel über komplexe und teure Workrounds kompensiert werden. Beispiele dafür wären: Merge-Prozesse, die einen hohe Ressourcenaufwand benötigen und Temp-Bereiche, die den

Hauptspeicher reduzieren. Anders als bei herkömmlichen In-Memory-Datenbanken ist bei Oracle Database In-Memory die Datenbankgröße nicht an den verfügbaren DRAM gebunden: Die zahlreichen Optimierungen auf allen Ebenen innerhalb der Speicherhierarchie (DRAM, Flash, Disk, als auch über Maschinengrenzen innerhalb eines RAC-Clusters hinweg) ermöglichen Datenbanken in einer annährend unbegrenzten Größe. Des Weiteren ermöglicht der Smart Flash Cache des Exadata Storage Server einen übergreifenden Speicherzugriff über die Grenzen von Hauptspeicher, Flash und Festplatte.

6.1 DIE SCHLÜSSELEIGENSCHAFTEN DER ORACLE DATABASE IN-MEMORY-OPTION

Das Columnar-Format ist gut geeignet für OLAP, Reporting und Analytics, da analytische Abfragen typischerweise auf eine kleine Anzahl von Spalten innerhalb einer großen Anzahl von Zeilen und damit größere Datenmengen zugreifen. Die Benutzerzahl in diesem Umfeld ist eher klein. Auf der anderen Seite bedient das Row-Format OLTP-Workloads besser. Hier werden kleinere Datenmengen verarbeitet, mit einer großen Anzahl von Columns innerhalb weniger Rows. Es arbeiten viele Benutzer gleichzeitig. Dies ist der Grund dafür, dass der Row-Store von den meisten OLTP-Datenbanken verwendet wird – auch von reinen In-Memory-Datenbanken wie Oracle TimesTen. Man er-

kennt, dass weder das eine noch das andere Format für alle Workloads geeignet ist. Dies ist der Grund dafür, dass Oracle Database In-Memory das **duale Format** eingeführt hat, auf das simultan zugegriffen werden kann und somit das Beste aus beiden Welten vereint: sowohl den In-Memory Column Store (IM Column Store) als auch den In-Memory Row Store (bekannt als Buffer Cache).

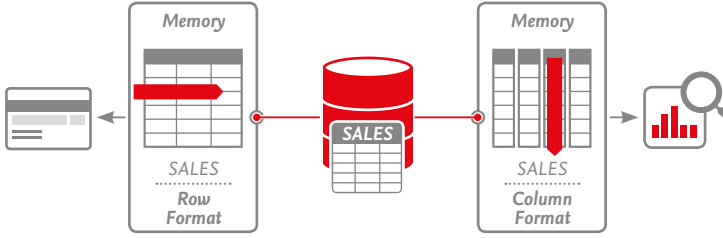


Abb. 6: Die Dual-Format-Architektur

Diese duale Repräsentation bedeutet nicht, dass sich die Speicheranforderungen verdoppeln. Der Oracle-Datenbank Buffer Cache wurde über die letzten Jahrzehnte derart optimiert, dass selbst ein extrem kleiner Buffer Cache hohe Hit-Raten bei vergleichsweise großen Datenbanken ermöglicht. Die Größe des Buffer Cache kann weiter reduziert werden,

da mit dem IM Column Store auch analytische Indizes ersetzt werden können. Der IM Column Store kann auch als eine Art In-Memory Index-Container angesehen werden – mehr dazu im Abschnitt 6.3 zum Storage Index.

Bei Oracle Database In-Memory muss **nicht die komplette Datenbank** in den IM Column Store passen. Es sollten nur die wichtigen Tabellen und Partitionen (z. B. die Bestell-Partition des letzten Monats aus der gesamten Bestell-Tabelle, die mehrere Jahre umfassen kann) in den IM Column Store geladen werden. Die restlichen Daten werden dann bei Bedarf aus dem Buffer Cache bedient. So kann der Hauptspeicher sinnvoll für die Daten verwendet werden, auf die häufig zugegriffen wird und bei denen der IM Column Store einen Vorteil bietet (große Tabellen und passende Abfragen). Zudem stehen mit den Engineered Systems wie Exadata und SuperClusters Systeme zur Verfügung, bei denen die Daten über mehrere Storage Tiers verteilt werden können: über High Capacity Festplatten, PCI Flash Cache, der extrem hohe IOPS liefert, sowie DRAM mit der geringstmöglichen Antwortzeit.

Der IM Column Store befindet sich im Data Access Layer innerhalb der Database Engine und stellt dem Optimizer alternative Zugriffsmethoden für extrem schnelle Table Scans zur Verfügung. Die einzige Stelle, die sich auf den Optimizer und den Query Processor auswirkt, sind die unterschiedlichen Kosten-

funktionen bei den In-Memory Scans. Der IM Column Store kommt zum Einsatz, falls die Kosten innerhalb der In-Memory-Statistiken des IM Columns Stores günstiger sind als die Kosten des Row Stores. Deshalb müssen die klassischen Statistiken stets aktualisiert werden. Die direkte Implementierung des IM Column Store in die Database Engine bewirkt, dass **sämtliche Features und Betriebsprozesse** der Oracle-Datenbank wie beispielsweise Database Recovery, Disaster Recovery, Backup, Replikation, Storage Spiegelung und das Clustern von Knoten transparent mit dem IM Column Store zusammenarbeiten.

Beim neuen In-Memory-Columnar-Format handelt es sich um ein **reines In-Memory-Format**. Der Gebrauch hat keinen Einfluss auf das Row-Format. Der IM Colum Store lässt sich im Rahmen von DML-Operationen sehr leicht pflegen. Da keinerlei Persistierung erfolgt, sind nur sehr geringe Datenänderungen innerhalb der In-Memory-Komponenten notwendig. Die Größe des IM Column Store wird über einen einzigen Initalisierungsparameter gesteuert.

6.2 DIE EINFACHE HANDHABUNG

Die In-Memory Area wird in der aktuellen Version über den statischen Initialisierungsparameter INMEMORY_SIZE aktiviert. Dieser Parameter steht standardmäßig auf „0“ und

bedeutet, dass die In-Memory Area nach dem Erzeugen einer Oracle-Datenbank-Instanz nicht aktiviert ist. Um mit der neuen Option arbeiten zu können, muss einfach nur die INMEMORY_SIZE auf einen Wert größer 100 MB gesetzt werden.

Zum Kennenlernen der Oracle Database In-Memory-Option genügt eine relativ kleine In-Memory Area, die leicht vom Hauptspeicher der bestehenden Server-Hardware verwendet werden kann. Im nachfolgenden Beispiel wird auf einem Server mit 128 GB RAM eine 55 GB große In-Memory Area innerhalb der System Global Area (SGA) allokiert. Dazu muss anschließend ein Neustart der Oracle-Datenbank-Instanz erfolgen.

Der Wert von SGA_TARGET sollte ebenfalls entsprechend erhöht werden, da sich ansonsten die übrigen Pools innerhalb der SGA verkleinern können. In die In-Memory Area passt dann beispielsweise ein Data Mart mit einer sehr großen Faktentabelle (in unserem Beispiel mit über 600 Millionen Zeilen) und mehreren Dimensionstabellen und man hat noch genügend Platz für weitere Kandidaten.

Die Formel für die Bestimmung von SGA_TARGET lautet wie folgt:

Im Single-Instance-Umfeld: sga_target = sga_target + inmemory_size

Im RAC-Umfeld: sga_target = (sga_target + inmemory_size) *1.1

Die folgenden Kommandos fassen die Aktivierung in SQL*PLUS zusammen. Daneben gibt es selbstverständlich auch die Möglichkeit, die Aktivierung über Oracle Enterprise Manager Cloud Control oder Oracle Enterprise Manager Database Express durchzuführen.

```
SQL> alter system set inmemory_size = 55G scope=spfile;
System altered.

SQL> startup force;
ORACLE instance started.
Total System Global Area 1.1811E+11 bytes
Fixed Size                  7660072 bytes
Variable Size            9663679960 bytes
Database Buffers         4.9124E+10 bytes
Redo Buffers              260771840 bytes
In-Memory Area           5.9056E+10 bytes
Database mounted.
Database opened.

SQL> show sga
Total System Global Area 1.1811E+11 bytes
Fixed Size                  7660072 bytes
Variable Size            9932115416 bytes
Database Buffers         4.8855E+10 bytes
Redo Buffers              260771840 bytes
In-Memory Area           5.9056E+10 bytes
```

Der IM Column Store wird aus den persistenten Inhalten des Row Stores (sprich den auf Platte gespeicherten Oracle-Blöcken) durch einen Populate-Mechanismus aufgebaut. Ein entscheidender Mehrwert ist der Umstand, dass der IM Column Store auch nur mit Teilen der Datenbank gefüllt werden kann. Zum Befüllen wird die neue INMEMORY-Klausel des CREATE/ALTER Statements verwendet, mit der die gewünschten Kandidaten für die Speicherung innerhalb des IM Column Store ausgewählt werden. Diese Befehle markieren die Objekte im Data Dictionary für die Verwendung innerhalb des IM Column Stores. Dementsprechend wird zum Entfernen eines Objekts aus dem IM Column Store die NO INMEMORY Klausel angewendet. Folgende Datenbank-Objekte können mit INMEMORY markiert und in den IM Column Store geladen werden:

- Tablespaces
- Tabellen
- Spalten einer Tabelle
- Partitionen einer Tabelle
- Subpartitionen innerhalb einer Partition
- Materialized Views

Nachfolgende Beispiele veranschaulichen die Verwendung der neuen INMEMORY-Klausel:

```
ALTER TABLE region INMEMORY;

ALTER TABLE historie NO INMEMORY;

CREATE TABLE verkaeufe
PARTITION BY LIST
   (PARTITION p1 … INMEMORY,
    PARTITION p2 … NO INMEMORY);

ALTER TABLE kunden INMEMORY
NO INMEMORY (unterschrift);
```

Das letzte Beispiel zeigt, dass es auch möglich ist, eine oder mehrere Spalten eines Objektes von der Speicherung im IM Column Store auszuschließen. Die Tabelle „Kunden“ enthält eine elektronische Unterschrift, die in einem BLOB abgespeichert ist. Diese kommt bei den analytischen Abfragen innerhalb der Applikation nicht zur Verwendung und wird deshalb nicht in den IM Column Store abgespeichert. Dies trägt auch dazu bei, den Hauptspeicher höchst effizient für den IM Column Store zu verwenden.

Standardmäßig und falls keine Ladepriorität im Rahmen des CREATE- /ALTER-Table-Befehls angegeben wurde, wird der

IM Column Store durch einen Pool von Hintergrundprozessen befüllt. Die Anzahl der Prozesse wird über den Initialisierungsparameter INMEMORY_MAX_POPULATE_SERVERS gesteuert. Falls kein Parameter angegeben wird, greift die standardmäßige Verwendung der Hälfte aller CPU-Kerne (bei 32 Kernen also 16). Beim Laden werden einfach die Oracle-Blöcke um 90 Grad gedreht und in den IM Column Store geladen.

Sind die Daten dann in IM Column Store enthalten, berücksichtigt der Optimizer beim erneuten Aufruf den IM Column Store und greift darauf zu.

Mit der PRIORITY-Klausel und den entsprechenden Einstellungen kann die **Reihenfolge der Beladung** des IM Column Stores beeinflusst werden, also wie die Objekte im Anschluss an das Hochfahren der Datenbank geladen werden. Folgende Einstellungen stehen zur Verfügung: CRITICAL, HIGH, MEDIUM, LOW und NONE. Objekte, die mit CRITICAL markiert sind, werden zuerst geladen, dann folgen die Objekte mit HIGH, MEDIUM und LOW. NONE bedeutet, dass die Objekte beim ersten Zugriff in den IM Column Store geladen werden. Dies hat für den Erstaufrufer zur Folge, dass er noch nicht vom IM Column Store unmittelbar profitiert, sondern erst derjenige, der nach erfolgter Befüllung das Objekt aufruft.

Ein entscheidender Punkt soll hier noch erwähnt werden: Es ist zu erkennen, dass für die Nutzung des IM Column Stores

die Daten nicht über komplizierte selbstgeschriebene Laderoutinen und Replikationswerkzeuge in eine „reine" In-Memory-Datenbank „migriert" werden müssen. Bei Oracle Database In-Memory stehen alle Daten immer zur Verfügung, werden über den Zusatz INMEMORY für die Nutzung innerhalb des IM Column Store markiert und anschließend in den Column Store geladen. Die Pflege der Objekte im Column Store wird vollständig von der Oracle-DB übernommen und ist der Pflege eines klassischen Indexes nicht unähnlich.

Die Speichernutzung der In-Memory Area kann über folgende Abfrage auf V$INMEMORY_AREA ermittelt werden:

```
SQL> select * from v$inmemory_area;

POOL        ALLOC_BYTES USED_BYTES POPULATE_STATUS
----------- ----------- ---------- -----------------
1MB POOL    4.7195E+10  3145728    DONE
64KB POOL   1.1787E+10  393216     DONE
```

Wie bereits erwähnt befüllen die Worker-Prozesse (z. B. ORA_W001_IM) den IM Column Store und die darin enthaltenen In-Memory Column Units (IMCU) mit den Daten aus den Oracle-Blöcken von Platte. Diese umfassen eine großen Anzahl von Zeilen aus einem oder mehren Table Extends, die in einzelnen Spalten repräsentiert werden. Die

Größe ist abhängig von der Zeilengröße, dem Datentyp und dem verwendeten Komprimierungsalgorithmus (siehe unten). Jede Spalte – auch Rowids - werden in einer separaten fortlaufenden Compression Unit (Column CU) gespeichert. Charakteristisch ist die feste Größe von 1 MB je Column Unit. Im Vergleich dazu haben die Oracle-Blöcke auf Platte, die beim Zugriff im Buffer Cache landen, eine Größe von 8K (bei OLTP-Applikationen) beziehungsweise bis zu 32K (bei OLAP).

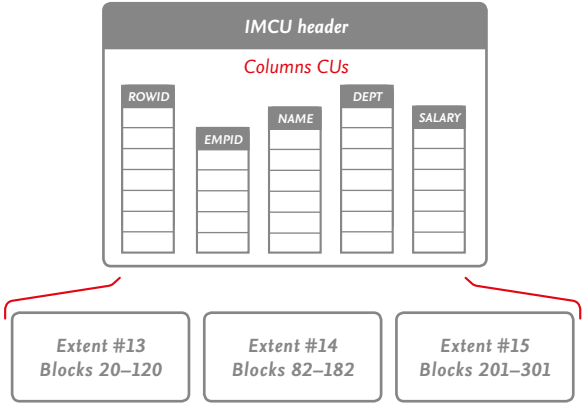


Abb. 7: Column Compression Units und Extends

Die Metadaten des IM Column Stores landen in einem Snapshot Metadata Pool bestehend aus 64K großen Snapshot Metadata Units (SMUs). SMUs speichern außer den Metadaten auch Informationen über Transaktionen ab. Die vorangegangene Abfrage auf V$INMEMORY_AREA zeigt diese beiden Pools mit den 1MB und 64K Units an.

Der IM Column Store verwendet ein reines In-Memory-Format, ist statisch und folgt keinem Least-Recently-Used-Algorithmus. In der aktuellen Version ist deshalb der DBA oder Anwender für die Befüllung des IM Column Stores verantwortlich. Der IM Column Store ist geeignet für einheitliche Zugriffe (über möglichst viele bzw. alle Zeilen einer Tabelle). Dies ist typisch für analytische Abfragen. Der Buffer Cache hingegen ist vorgesehen für nicht einheitliche Zugriffe (z. B. nur einige Zeilen einer Tabelle). Dies ist typisch für OLTP-Abfragen, die B-Tree-Indizes verwenden. Der IM Column Store ist zusätzlich mit neuen Komprimierungsalgorithmen und Optimizer-Zugriffsmethoden ausgestattet.

Die neuen Komprimierungsalgorithmen des IM Column Stores sind für die Hauptspeicher-Verarbeitung optimiert und sind unabhängig von den Komprimierungsalgorithmen, die in der Advanced Compression Option für die Speicherung auf Disk enthalten sind. Diese verfolgen das Ziel, möglichst effizient mit dem Speichermedium Disk umzugehen.

Wie bereits beschrieben, wird jede Spalte in fortlaufenden Compression Column Units (CU) abgespeichert. Der Spalten-Vektor selbst wird komprimiert und der Anwender kann aus diversen Compression Levels abhängig vom Anwendungsfall auswählen. Die Auswahl erfolgt über die MEMCOMPRESS-Unterklausel.

MEMCOMPRESS	Beschreibung
DML	Minimale Komprimierung, optimiert für DML-Performance
QUERY LOW	Optimiert für beste Abfrage-Performance (Default)
QUERY HIGH	Ebenfalls für Abfrage-Performance optimiert, zusätzlich verbesserte Speichernutzung
CAPACITY LOW	Ideale Balance zwischen Abfrage-Performance und Speichernutzung
CAPACITY HIGH	Optimiert für beste Speichernutzung

Tabelle 1: Komprimierungsarten

Standardmäßig wird QUERY LOW verwendet, falls kein anderer Komprimierungsalgorithmus angegeben wird. Innerhalb einer partitionierten Tabelle können für die einzelnen Partitionen sogar unterschiedliche Komprimierungslevels verwendet werden, je nach Zugriffsart und -häufigkeit.

6.3 DIE NEUEN OPTIMIZER-ZUGRIFFSMETHODEN

Oracle Database In-Memory stellt neue Optimizer-Zugriffsmethoden speziell für die In-Memory-Verarbeitung zur Verfügung. Zu nennen sind hier In-Memory Scans, In-Memory Joins sowie In-Memory-Aggregationen und In-Memory-Gruppierungen.

Bei Abfragen auf den IM Column Store werden die vorhandenen Vektor-Register der CPU-Kerne verwendet. Über SIMD-Instruktionen (Single Instruction Multiple Data Values) werden mehrere Werte mit einer einzigen CPU-Instruktion gleichzeitig verarbeitet und durch die unmittelbare Nutzung der CPU-Taktung ergeben sich sehr hohe Scan-Raten.

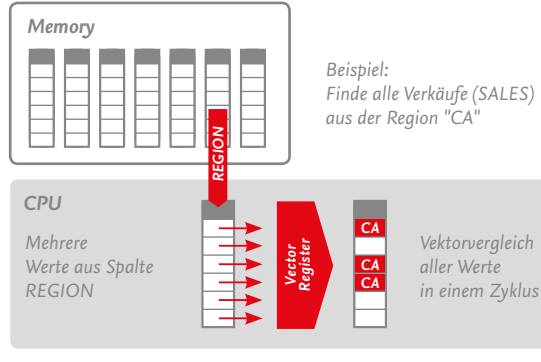


Abb. 8: Nutzung von SIMD-Instruktionen

Mit dem neuen In-Memory Storage Index lassen sich die Daten, auf die zugegriffen wird, zusätzlich enorm reduzieren. Dieser wird automatisch für jede Spalte im IM Column Store angelegt und gepflegt. Die darin enthaltenen Wertebereiche (Minimum und Maximum der Werte in einer IM Column Unit) werden mit den Prädikaten innerhalb einer WHERE-Klausel abgeglichen und überspringen beim Scannen – ähnlich beim Pruning bei Partitionen – nicht relevante Bereiche der IM Column Units. Des Weiteren dient das Metadaten-Dictionary als Basis für ein Dictionary Pruning bei bestimmten Operationen wie Equality, IN-List und einigen Range-Prädikaten. Hier wird im Dictionary überprüft, ob der Wert tatsächlich in einer bestimmten In-Memory Column Unit vorhanden ist. Auf diese Weise wird sichergestellt, dass nur relevante Column Units gescannt werden.

In-Memory Joins sind ebenfalls neu und tragen erheblich zur Performancesteigerung bei. Dabei werden Joins zwischen der Faktentabelle und den Dimensionstabellen zu einem Filter-Scan umgewandelt. Im ersten Schritt werden beim Scannen der Dimensionstabelle Bloom Filter mit den relevanten Werten erzeugt, die dann im zweiten Schritt beim Scannen der Faktentabelle angewendet werden. Dadurch lässt sich die Anzahl der zu verarbeitenden Werte gegenüber einem klassischen Join erheblich reduzieren.

Sie arbeiten auch sehr gut mit Buffer-Cache-Abfragen oder gemischten Abfragen (sowohl über den IM Column Store als auch den Buffer Cache) zusammen.

Das folgende SQL-Statement veranschaulicht im Ausführungsplan die Erzeugung von einem Bloom-Filter (:BF0000). Dieser Bloom-Filter wird beim Scannen der große Faktentabelle (LINEORDER) angewendet. Das hier auf den IM Column Store zugegriffen wird, erkennt man anhand des neuen Schlüsselwortes INMEMORY in der Operation TABLE ACCESS INMEMORY FULL. Interessant ist auch die Anwendung der Prädikate in den Filtern und die Verwendung der Parallelisierung (hier im Beispiel mit 16 CPU-Kernen). Die Statistiken belegen, dass der komplette Zugriff über das Memory erfolgt – ohne Physical Reads.

```
SQL> SELECT SUM(lo_extendedprice * lo_discount) revenue
     FROM lineorder l, date_dim d
     WHERE l.lo_orderdate = d.d_datekey
     AND l.lo_discount BETWEEN 2 AND 3
     AND l.lo_quantity < 24
     AND d.d_date='December 24, 1996';

SQL> SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                          | Name        | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |             |     1 |    43 |  4280  (23)|
|   1 |  SORT AGGREGATE                    |             |     1 |    43 |            |
|   2 |   PX COORDINATOR                   |             |       |       |            |
|   3 |    PX SEND QC (RANDOM)             | :TQ10001    |     1 |    43 |            |
|   4 |     SORT AGGREGATE                 |             |     1 |    43 |            |
|*  5 |      HASH JOIN                     |             | 20286 |   851K|  4280  (23)|
|   6 |       JOIN FILTER CREATE           | :BF0000     |     1 |    25 |    15   (0)|
|   7 |        PX RECEIVE                  |             |     1 |    25 |    15   (0)|
|   8 |         PX SEND BROADCAST          | :TQ10000    |     1 |    25 |    15   (0)|
|   9 |          PX SELECTOR               |             |       |       |            |
|* 10 |           TABLE ACCESS FULL        | DATE_DIM    |     1 |    25 |    15   (0)|
|  11 |       JOIN FILTER USE              | :BF0000     |    50M|   861M|  4257  (23)|
|  12 |        PX BLOCK ITERATOR           |             |    50M|   861M|  4257  (23)|
|* 13 |         TABLE ACCESS INMEMORY FULL | LINEORDER   |    50M|   861M|  4257  (23)|
-------------------------------------------------------------------------------------------

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

   5 - access("L"."LO_ORDERDATE"="D"."D_DATEKEY")
  10 - filter("D"."D_DATE"='December 24, 1996')
  13 - inmemory("L"."LO_DISCOUNT"<=3 AND "L"."LO_QUANTITY"<24 AND "L"."LO_DISCOUNT">=2 AND
              SYS_OP_BLOOM_FILTER(:BF0000,"L"."LO_ORDERDATE"))
       filter("L"."LO_DISCOUNT"<=3 AND "L"."LO_QUANTITY"<24 AND "L"."LO_DISCOUNT">=2 AND
              SYS_OP_BLOOM_FILTER(:BF0000,"L"."LO_ORDERDATE"))
```

```
Degree of Parallelism is 16 because of table property
Statistics
----------------------------------------------------------------------------------
  39  recursive calls
   0  db block gets
 320  consistent gets
   0  physical reads
   0  redo size
3940  bytes sent via SQL*Net to client
 574  bytes received via SQL*Net from client
   4  SQL*Net roundtrips to/from client
   1  sorts (memory)
   0  sorts (disk)
  34  rows processed
```

Komplexe Report-Abfragen werden durch die neue Optimizer-Transformation namens Vector Group By bei der Berechnung von multidimensionalen Aggregaten erheblich beschleunigt. Diese Transformation ist vergleichbar mit der bekannten Star-Transformation, die vor allem im Bereich Data Warehouse zum Einsatz kommt. Auch hier können Abfragen auf den Buffer Cache oder bei gemischten Abfragen auf IM Column Store und Buffer Cache zum Einsatz kommen.

6.4 BEST PRACTICES

Oracle Database In-Memory oder genauer ein IM Column Store im Allgemeinen ist kein Allheilmittel, um alle Performanceprobleme zu lösen. Analytische Abfragen und Reporting profitieren am meisten; Lookup-Abfragen, die nur einen oder wenige Datensätze zurückgeben, fahren mit den bewährten B-Tree-Indizes besser. Durch den im IM Column Store enthaltenen IM Storage Index ist es möglich, auf analytische Indizes, die für analytische Abfragen (Summierungen, Gruppierungen usw.) und Reporting verwendet werden, weitestgehend zu verzichten. Der DBA wird entlastet, da er sich nicht mehr um die Anlage derartiger Indizes kümmern muss. Die Fachabteilung kann jegliche Ad-hoc-Abfragen sofort ausführen, ganz abgesehen von der Platzersparnis und den Performance-Steigerungen, die dadurch im operativen Betrieb durch den Wegfall der Pflege dieser Indizes zusätzlich entstehen. Ein absolutes Highlight von Oracle Database In-Memory stellt das Reporting auf die operativen Daten (sogenannte Mixed Workloads) dar. Diese sind die beliebtesten Use Cases bei den Proof of Concepts (PoCs) unserer Kunden. Wo früher separate Datenbanken für das Reporting hochgezogen werden mussten, kann heute dank Oracle Database In-Memory EINE Datenbank für alles verwendet werden.

Folgende Regeln sollten Sie bei der Verwendung in Betracht ziehen:

- Die Anwendung sollte idealerweise schon auf Oracle 12c migriert sein und die neuen Möglichkeiten im Rahmen der Optimierungen sollten berücksichtigt werden.
- Für den produktiven Einsatz sollte immer der aktuelle Bundle Patch von support.oracle.com heruntergeladen und eingespielt werden – siehe dazu Doc ID 1937782.1 (12.1.0.2 Bundle Patches for Engineered Systems and DB In-Memory). Der Titel mag auf den ersten Blick verwirrend erscheinen, da hier auch die Rede von Engineered Systems ist. Der Bundle Patch betrifft auch die normalen Datenbanken.
- Für die Bestimmung des Platzbedarfes kann das PL/SQL Package DBMS_COMPRESSION (Compression Advisor) verwendet werden. Man erhält eine recht genaue Compression Ratio und damit den Speicherbedarf von den Objekten, die in den IM Column Store geladen werden sollen.
- Der In-Memory Advisor ist übrigens schon in Oracle 11.2.0.3-Umgebungen nutzbar. Mit seiner Unterstützung kann man aufgrund des Workloads erfahren, welchen Mehrwert Oracle Database In-Memory in der eigenen Umgebung voraussichtlich liefern wird. Er ist verfügbar über einen My Oracle Support Download. Die Skripte zum Anlegen der Kandidaten liefert das Tool gleich mit.

- Für einen Proof of Concept (PoC) liefern meistens auch die Kollegen, die die Anwendung kennen, die besten Kandidaten für den IM Column Store. Wie oben geschildert, ist der Aufwand für den Aufbau eines Test-Cases minimal und kann quasi über die Mittagspause erfolgen. Zudem liefern AWR- und Statspack-Reports im Rahmen der Top-SQL-Statements mit den längsten Laufzeiten, einen guten Start für einen Test mit den eigenen Daten. Es reicht aus, die zu verwendenden Tabellen INMEMORY zu setzen.
- Das SQL Plan Management ist ebenfalls eine hervorragende Möglichkeit, um den Nutzen von Oracle Database In-Memory feststellen zu können. Die als Basis zu verwendende SQL Plan Baseline sollte aus einer bestehenden Umgebung mit Oracle 11g beziehungsweise Oracle 12c stammen – ohne aktivierter Oracle Database In-Memory Option.
- Einfache Tests lassen sich mit den Statement Hints INMEMORY beziehungsweise NO_INMEMORY durchführen. Der Session-Parameter INMEMORY_QUERY erlaubt mit dem Wert ENABLE beziehungsweise DISABLE die Ausführung von Statements mit und ohne Verwendung des In-Memory Column Stores.
- Ab Enterprise Manager Cloud Control 12c Release 4 lässt sich sogar eine In-Memory Object Page über die Menüpunkte Administration → In Memory Central erreichen.

Hier können Einstellungen der In-Memory-Datenbank, der Compression Factor und weitere Eigenschaften der einzelnen Objekte im Column Store eingesehen werden. Zusätzlich können die Zugriffe auf die Column-Store-Objekte grafisch in einer Heat Map angezeigt werden.

Nun stellt sich die Frage, welche Zugriffe und Operationen am meisten von dem Einsatz der In-Memory Option profitieren können. Aus der Herleitung sollte klar geworden sein, dass beim eigentlichen Datenzugriff über Analystics und Reporting und den Wegfall von analytischen Indizes ein großer Mehrwert zu erwarten ist.

Das größte Potenzial haben dabei Zugriffe, die über eine größere Anzahl von Rows gehen und dabei wenige Ergebnisse zurückgeben. Je weniger Rows verwendet werden, umso besser. Das Selektieren aller Columns einer sehr breiten Tabelle führt allerdings zu einem Re-Tupling, das sehr kostenintensiv ist. Man sollte sich daher die folgenden einfachen Regeln merken: Je weniger Rückgabewerte und je selektiver, umso besser.

Folgendes Schaubild stellt die Bereiche dar, die grundsätzlich im Rahmen der Beurteilung der Anwendungsperformance betrachten werden sollten. In diesem Kontext kann auch im Vorfeld schon der Nutzen von Oracle Database In-Memory eingeschätzt werden.

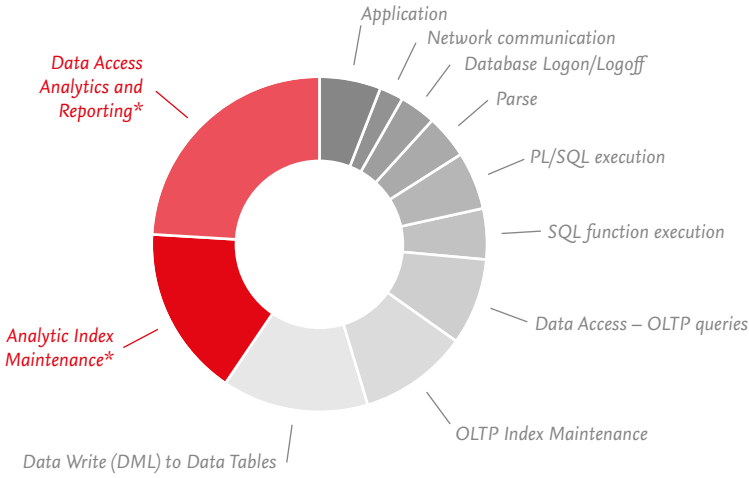


Abb. 9: Bereiche und effiziente Nutzung durch die In-Memory-Option

Es gibt somit durchaus Bereiche, die weniger von dem Einsatz der In-Memory-Option profitieren. Dazu gehören beispielsweise die satzweise Verarbeitung zur Layout-Kontrolle, der Netzwerkverkehr, Parse-Zeiten oder komplexe PL/SQL-Codes.

7 Fazit und Ausblick

Wir haben uns im vorliegenden Dojo auf die bekannten und häufig eingesetzten Aspekte der Memory-Techniken konzentriert und wollen keinen Anspruch auf Vollständigkeit erheben. Da die meisten Techniken nur kurz beschrieben wurden, empfiehlt sich die nachfolgende Lektüre von Handbüchern, exzellenten White Papers und Blogs (siehe Kapitel 9: Weitere Informationen). Darüber hinaus ist zu erwarten, dass sich die Technologie weiterentwickeln wird, sodass in Zukunft sicher noch Erweiterungen und Neuigkeiten in diesem Bereich zu erwarten sind. So wird es im Bereich Oracle Database In-Memory mehr Performance durch größere Unterstützung von Operationen und Ausdrücken geben, eine vereinfachte Verwaltung durch verbesserten Advisor, Einbindung in das Automatic Data Management und dynamisches Memory Management (In-Memory Area) geben, um nur einige Beispiele zu nennen.

Zusammenfassend sollte folgendes deutlich werden: Alle Techniken können und sollen ergänzend Verwendung finden, damit optimale Performance für Ihre Datenbankanwendungen erreicht werden kann. Beispielsweise spricht nichts dagegen die Oracle Database In-Memory-Technik mit der Result-Cache-Technik zu kombinieren, falls Ihre Applikationen davon profitieren können. Die Abfragen müssen in

der Regel auch nicht angepasst werden – einzige Ausnahme bildet der PL/SQL Result Cache. Tabellenattribute, Session-Eigenschaften oder einfach nur das Bereitstellen einer Technik reichen im Normalfall zur Nutzung aus. Testen lassen sich dabei viele Features ganz einfach über dynamische Session-Parameter. Hat man also einen Workload zur Verfügung – als SQL-Skript oder gar als SQL Tuning Set – ist ein schneller und einfacher Test ohne großen Aufwand möglich.

Alle weiteren Datenbankoptionen und -mechanismen wie zum Beispiel im Bereich Security, Hochverfügbarkeit usw. werden übrigens 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 und der Anwendungsentwickler ändert sich nicht, kein Mehraufwand ist zu erwarten – die komplette Oracle-Infrastruktur bleibt ohne Modifikation erhalten.

Wir haben absichtlich nur wenige oder gar keine Performancezahlen angegeben, da die Performance wie bei vielen Features abhängig vom eigenen Workload und der Umgebung ist. Also am Besten ausprobieren und sich selbst vergewissern!

8 Lizenzierung und Database Cloud Services

Für einige der in diesem Dojo erwähnten Features ist die Lizenzierung der Enterprise Edition (Stand Januar 2016) nötig. Dabei handelt es sich beispielsweise um folgende Funktionen:

- OCI Client Side Query Cache
- Query Result Cache
- PL/SQL Function Result Cache

Zusätzlich ist für die Nutzung der folgenden Features der Oracle Database In-Memory (Stand Januar 2016) nötig:

- In-Memory Column Store
- In-Memory Aggregation
- Fault Tolerant In-Memory Column Store (zusätzlich mit Exadata oder Supercluster)

Für folgendes Feature ist das Oracle Diagnostics und Tuning Pack erforderlich:

- Oracle Database In-Memory Advisor

Zur Verifizierung des aktuellen Stands wird die Lektüre des Handbuchs Oracle Database Licensing Information 12c Release 1 (12.1) empfohlen.

Bei den Database Cloud Services können die Standard oder Enterprise Edition Services zum Einsatz kommen. Möchte man die Packs verwenden, ist die High-Performance-Variante erforderlich. Für die Database In-Memory Technology sollte die Extreme Performance oder die Exadata Cloud Service Variante gewählt werden. Genauere Informationen zu den Database Cloud Services sind auf cloud.oracle.com zu finden.

9 Weitere Informationen

My Oracle Support Notes:

- In-Memory Advisor Download (MOS DOC ID 1965343.1)
- 12.1.0.2 Bundle Patches for Engineered Systems and DB In-Memory (MOS DOC ID 1937782.1)
- Oracle Database In-Memory Option (DBIM) Basics and Interaction with Data Warehousing Features (MOS Doc ID 1903683.1)

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:

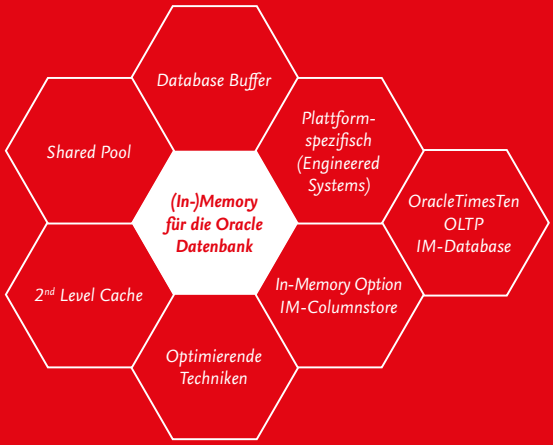
- Oracle Database In-Memory, July 2015
- Oracle Database In-Memory Advisor Best Practices, February 2015

- When to Use Oracle Database In-Memory, July 2015
- Parallel Execution with Oracle Database 12c Fundamentals, October 2010

Handbücher:

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

NOTIZEN



Copyright © 2016, Oracle. All rights reserved. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Herausgeber: Günther Stürner, Oracle Deutschland B.V.
Design: volkerstegmaier.de // Druck: Stober GmbH, Eggenstein

ORACLE®

Zugriff auf die komplette
Oracle Dojo-Bibliothek unter
http://tinyurl.com/dojoonline

ORACLE®