ORACLE

ORACLE DOJO NR. 8

ULRIKE SCHWINN

Komprimierung in der Datenbank

Tabellen, Large Objects, Indizes, RMAN, Data Pump, Netzwerk, Data Guard, Information Lifecycle Management

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

Der Begriff Dojo ['do:d30] 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. 8* beschäftigt sich Ulrike Schwinn, Leitende Systemberaterin bei Oracle Deutschland B.V., mit dem weiten Feld der Datenkompression innerhalb der Oracle-Datenbank. Diese fundierte Einführung, die mit einer Vielzahl praxiserprobter Tipps abgerundet wird, hilft Speicherplatz zu sparen (und damit Geld) und macht zudem, quasi als „Nebenwirkung", die meisten Anwendungen schneller!

ORACLE®

Inhalt

ORACLE

ORACLE DOJO NR. 8

ULRIKE SCHWINN

Komprimierung in der Datenbank

Tabellen, Large Objects, Indizes, RMAN, Data Pump, Netzwerk, Data Guard, Information Lifecycle Management

VORWORT DES HERAUSGEBERS

Wir leben in einer Zeit der massiven Datengenerierung. Immer mehr Daten fallen an, immer feingranularer werden Daten abgespeichert, immer länger werden Daten aufbewahrt. Mehr operative Daten bedeutet mehr Speicherplatz, der zur Verfügung gestellt werden muss und bedeutet – oft unterschätzt oder ignoriert –, dass mit noch größerer Sorgfalt an den Verfügbarkeitskonzepten wie auch an den Performancekonzepten gearbeitet werden muss. Waren gestern Systeme und Datenbanken im zweistelligen Gigabyte-Bereich eine bestaunenswerte Größenordnung, reden wir heute von ein-, zwei- oder dreistelligen Terabytes, und Petabytes sind nicht fern, bei manchen bereits Realität.

Es ist einfach eine Frage der Vernunft und vor allem der Kosten, diese großen Datenbestände, die zu verarbeiten sind, möglichst effizient und kompakt zu speichern.

Oracle hat die Problematik schon sehr früh erkannt und bereits mit Oracle 9i die ersten wichtige Schritte im Bereich Kompressionstechnologie innerhalb der Datenbank unternommen. Heute, mit Oracle12c, steht ein ganzes Arsenal von Möglichkeiten zur Verfügung, um die Datenbestände immens zu verkleinern, damit Kosten zu sparen und insgesamt die Performance und vor allem die Handhabbarkeit der

Anwendungssysteme zu erhöhen. Nun stehen unterschiedliche Kompressionstechnologien für unterschiedliche Zielobjekte (Tabellen, Indices, LOBs, Backups, Netzwerk) zur Verfügung, die – sinnvoll angewandt – eine signifikante Reduzierung des Platzbedarfs innerhalb einer Oracle-Datenbank herbeiführen.

Sinnvoller Einsatz ist das Stichwort für dieses Dojo. Es hat zum Ziel, in die unterschiedlichen Themengebiete der Datenkompression und der Daten-Deduplikation innerhalb einer Oracle-Datenbank einzuführen. Ulrike Schwinn, Leitende Systemberaterin bei Oracle Deutschland B.V., zeigt anhand vieler praxisorientierter Beispiele und wertvoller Tipps, wie man signifikant Speicherplatz und damit viel Geld einsparen kann.

Testen Sie einfach die unterschiedlichen Technologien oder nutzen Sie den Compression Advisor, um die Kompressionsraten in Ihrem System, bei Ihren Tabellen – mit und ohne LOBs – oder Ihren Backups festzustellen. Ich bin sicher, dass Sie überrascht sein werden, wie viel Speicherplatz Sie einsparen können.

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!

1 Einleitung

Schon seit vielen Jahren ist die Komprimierung von Daten ein wichtiger Bestandteil der Oracle-Datenbank und wird beständig weiterentwickelt. Dies zeigte sich besonders auch im Datenbankrelease **11g** mit der Einführung von neuen Techniken im Zusammenhang mit der neuen Option **Advanced Compression**. Die Komprimierung ist nun beispielsweise unabhängig vom Anwendungsworkload und zusätzlich um die Bereiche unstrukturierte Daten, Backup-Daten und Netzwerk-Komprimierung (im Data-Guard-Umfeld) erweitert worden.

In **Oracle Database 12c** sind sogar Eigenschaften zur Verbesserung des Storage Management ergänzt worden. Im Wesentlichen handelt es sich dabei um zwei neue Features – die **Heat Map** und die **automatische Datenoptimierung** (Englisch **A**utomatic **D**ata **O**ptimization). Die Heat Map „trackt" Veränderungen und Abfragen auf Zeilen- und Segmentebene und gibt einen detaillierten Überblick über den Zugriff auf die Daten. Die automatische Datenoptimierung verlagert und/oder komprimiert die Daten gemäß nutzerdefinierter Regeln (Englisch policies) basierend auf den Informationen, die sich aus der Heat Map ergeben. Beide zusammen helfen dabei, **Information Lifecycle Management**-Strategien in – und nicht außerhalb – der

Datenbank zu implementieren. Zusätzlich dazu bietet Oracle Database 12c eine Erweiterung der Netzwerkkomprimierung, eine Optimierung der Flashback Data Archives und des Online Partition MOVE.

Dieses Dojo gibt einen aktuellen Überblick über alle zur Verfügung stehenden Komprimierungsverfahren, die mit und ohne Lizenzierung der Advanced Compression Option zur Verfügung stehen, und illustriert die Handhabung mit zusätzlichen Tipps und Tricks. Da die einzelnen Kapitel nur die fachliche Komponente und nicht in allen Fällen die lizenzrechtliche Sicht beleuchten können, findet sich am Ende des Dojo eine aktuelle Liste der zu lizenzierenden Features.

Folgende Übersicht gibt zusätzlich einen kurzen Überblick über die umfassenden Möglichkeiten der Komprimierung in der Datenbank – aufgeteilt in die beiden Kategorien Basic und Advanced. Advanced steht dabei für die Features der Advanced Compression Option, Basic für die grundlegenden Komprimierungseigenschaften, die keine Lizenzierung der Advanced Compression Option erfordern.

Ebene	Basic Komprimierung	Advanced Komprimierung
Strukturierte Daten – Tabelle	Basic Compression	OLTP (Advanced Row) Compression
Unstrukturierte Daten – Securefiles		Advanced LOB Compression Advanced LOB Deduplication
Index	Bitmap Index komprimierter B*Index	
Partition	siehe Tabelle	siehe Tabelle
Tablespace	siehe Tabelle	siehe Tabelle
RMAN Backup	Algorithmus BASIC	weitere Algorithmen wie LOW, MEDIUM und HIGH
Data Pump	Metadaten	alle Daten
External Table		External Table Compression
Data Guard		Data Guard Redo Transport
Netzwerk		Advanced Network Compression
Flashback Data Archive		Optimized Flashback Data Archive

Nicht berücksichtigt ist die Tabellenkomprimierungsart HCC (kurz für **H**ybrid **C**olumnar **C**ompression), die unabhängig von der Advanced Compression Option auf

speziellem Storage wie Exadata Storage Systemen, Sun ZFS Storage Appliance (Oracle NAS Storage) oder Axiom Storage (Oracle SAN Storage) ihre Verwendung findet.

Ein Wort noch zum Setup und der Installation: Alle Komprimierungsarten stehen **ohne zusätzliche Installation** oder Aktivierung in der Datenbank zur Verfügung. Möchte man bestimmte Features verwenden, ist die Implementierung durch die vorgegebene Syntax erforderlich. Im Umkehrschluss bedeutet dies aber auch, dass die Advanced Compression Option nicht ausgeschaltet werden kann. Es gilt der Grundsatz: Werden die entsprechenden Funktionen nicht verwendet, ist auch keine Lizenzierung erforderlich – wie übrigens auch im Fall von anderen Optionen (zum Beispiel bei der Partitioning Option).

2 Tabellen mit strukturierten Daten

Daten in der Datenbank werden entweder als strukturierte oder als unstrukturierte Daten gespeichert. Bei strukturierten Daten handelt es sich beispielsweise um Ortsbezeichnungen, Kategorien, Namen usw., bei unstrukturierten Daten um Beschreibungen, Attachments oder Dokumente in unterschiedlichen Speicherformaten. Um eine effektive Speicherung zu gewährleisten, verwendet die Datenbank unterschiedliche Algorithmen bei der Komprimierung von Daten.

Im folgenden Abschnitt liegt der Fokus zuerst auf den strukturierten Daten. Drei Komprimierungsverfahren stehen dabei zur Verfügung:

- Basic (Table) Compression (auch Direct Load Compression)
- OLTP Compression (auch Advanced Row Compression)
- Hybrid Columnar Compression (auch Column Store Compression)

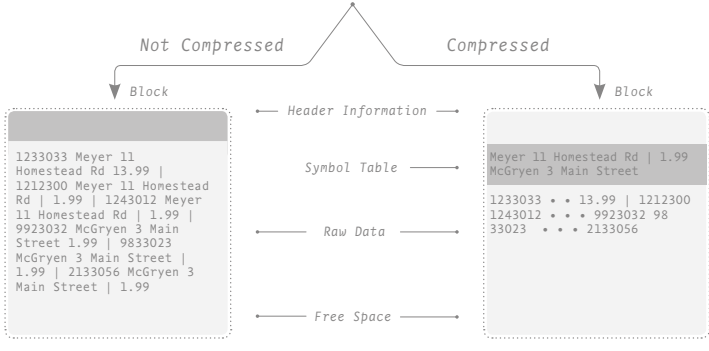
2.1 BASIC COMPRESSION

Bereits seit Oracle Version 9.2 ist es möglich, relationale Tabellendaten zu komprimieren. Dabei handelt es sich um die sogenannte Basic Compression – auch Direct Load

Compression genannt. Bei der Komprimierung der Daten werden Mehrfacheinträge im Datenblock nur einmal gespeichert. Die sich wiederholenden Werte werden in einer sogenannten „Symbol Table“ auf Blockebene gespeichert und durch einen Pointer im Datenteil des Blocks adressiert (siehe Abbildung 1).

Abb.1: Komprimierter und nicht-komprimierter Block im Vergleich

Invoice ID	Cust_Name	Cust_Addr	Sales_amt
1233033	Meyer	11 Homestead Rd	13.99
1212300	Meyer	11 Homestead Rd	1.99
1243012	Meyer	11 Homestead Rd	1.99
9923032	McGryen	3 Main Street	1.99
9833023	McGryen	3 Main Street	1.99
133056	McGryen	3 Main Street	1.99



Dieser Komprimierungsalgorithmus ist sehr effizient. Durch die Tabellenkomprimierung kann eine größere Anzahl Zeilen in einem Block gespeichert werden, daher werden weniger „Buffer Get"-Operationen beziehungsweise I/Os durchgeführt. Dies führt dazu, dass nicht nur der Speicherplatzbedarf auf der Platte und im Memory verringert wird, sondern auch die Performance der Tabellenzugriffe erhöht werden kann. Des Weiteren existieren keine funktionalen Einschränkungen bei der Nutzung von zusätzlichen Datenbank-Features. Für den Einsatz ist allerdings die Lizenzierung einer Enterprise Edition erforderlich.

Wichtig zu wissen ist, dass Basic Compression ausschließlich für Bulk-Load-Operationen genutzt werden kann. Dazu gehören folgende Operationen:

- Direct Path Load beim SQL*Loader
- `CREATE TABLE AS SELECT`
- Paralleler `INSERT`
- Serieller `INSERT` mit `APPEND` Hint und Subquery Klausel (auch Direct Path Insert)

Konventionelle DML-Operationen werden mit dieser Art der Komprimierung nicht unterstützt. Zusätzlich gibt es weitere Einschränkungen, die beachtet werden sollten: Operationen

wie DROP COLUMN auf Tabellen mit Basic Compression sind nicht erlaubt; generell ist Basic Compression auf Index Organized Tables (IOTs) nicht möglich. Genauere Informationen dazu sollten im Handbuch *Oracle Database SQL Language Reference* nachgelesen werden.

Tipp aus der Praxis: Auch Direct Path INSERTs unterliegen einigen Einschränkungen, die im Handbuch im Kapitel zum Kommando INSERT zu finden sind. Falls eine der Voraussetzungen bei der Verwendung nicht erfüllt wird, wird übrigens kein Fehler erzeugt, sondern automatisch auf eine konventionelle INSERT-Durchführung umgeschaltet. Die Konsequenz bei Verwendung der Basic-Compression-Eigenschaft wäre dann, dass keine Komprimierung erfolgen würde. Ein typisches Beispiel ist die Verwendung von Constraints. Laut Handbuch gilt Folgendes: "The target table cannot have any triggers or referential integrity constraints defined on it." Ein Direct Path INSERT schaltet also in einen konventionellen Load (INSERT) um, sobald ein referenzielles Constraint eingeschaltet ist. Das Constraint auszuschalten stellt dazu einen einfachen Workaround, eine einfache Lösung dar. Folgender Code-Auszug illustriert das Beispiel. Dabei besitzt Tabelle EMP1 ein Foreign Key Constraint, das im Fall 1 ausgeschaltet und im Fall 2 eingeschaltet wird.

Fall 1 mit Direct Load

```
SQL> alter table empl disable constraint FK_deptno;

SQL> insert /*+ append */ into empl select * from scott.emp2;

SQL> select * from table (dbms_xplan.display_cursor());

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
SQL_ID 8ttnnub4mO7rv, child number O
------------------------------------------------------------------------------
insert /*+ append */ into empl select * from scott.emp2

Plan hash value: 2748781111
------------------------------------------------------------------------------
| Id | Operation          | Name | Rows | Bytes | Cost (%CPU)| Time     |
------------------------------------------------------------------------------
| O | INSERT STATEMENT   |      |      |       |     3 (100)|          |
| 1 | LOAD AS SELECT     |      |      |       |            |          |
| 2 | TABLE ACCESS FULL  | EMP2 |  14  |   532 |      3  (0)| 00:00:01|
```

Fall 2 mit konventionellem Load

```
SQL> alter table empl enable constraint FK_deptno;

SQL> insert /*+ append */ into empl select * from scott.emp2;

SQL> select * from table (dbms_xplan.display_cursor());
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------
SQL_ID 8ttnnub4m07rv, child number 0
------------------------------------------------------------------------------
insert /*+ append */ into empl select * from scott.emp2
Plan hash value: 3956160932
------------------------------------------------------------------------------
| Id | Operation             | Name | Rows | Bytes |Cost(%CPU)| Time    |
------------------------------------------------------------------------------
| 0 | INSERT STATEMENT       |      |      |       | 3  (100) |
| 1 | LOAD TABLE CONVENTIONAL|      |      |       |          |
| 2 | TABLE ACCESS FULL      | EMP2 |  14  |  532  | 3    (0) |00:00:01|
```

Seit **11g Release 2** gibt es außerdem die **Hint-Erweiterung** `APPEND_VALUES`, die den Einsatz nun auch für Datenbankentwickler interessant macht. Mit dem neuen Hint können Bulk-Operationen auf die VALUES-Klausel ausgedehnt werden und nicht nur in Abhängigkeit einer Subquery-Klausel verwendet werden. Besonders profitieren können zum Beispiel umfangreiche Ladevorgänge in

PL/SQL-Anwendungen, die mit einer FORALL-Schleife und einem INSERT VALUES-Kommando ausgestattet sind.

Folgender Code-Ausschnitt zeigt ein Beispiel der Nutzung:

```
:
prod_ids NumList;
BEGIN
  FOR i IN 1..1000 LOOP prod_ids(i) := i;
  END LOOP;
  FORALL j IN 200..1000
           INSERT /*+ APPEND_VALUES */ INTO comp_tab
           VALUES (prod_ids(j),sysdate,...);
:
```

2.2 OLTP COMPRESSION (AUCH ADVANCED ROW COMPRESSION)

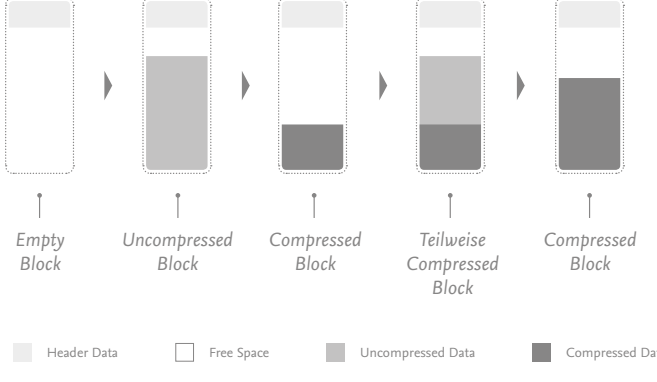
Mit Oracle 11g wurde die Basic-Compression-Methode um ein zweites Verfahren – das **OLTP-Compression**-Verfahren – erweitert. Mit diesem Verfahren werden nun alle DML-Operationen – auch konventionelle DML-Operationen – sowie ADD und DROP COLUMN-Kommandos unterstützt. Dies bedeutet, dass die Einschränkung auf Bulk-Load-Operationen entfällt und Komprimierung ohne Rücksicht auf die Art der Ladevorgänge garantiert werden kann. Diese neue Komprimierungsmethode wird als

OLTP Compression oder auch als **Advanced Row Compression** bezeichnet.

Hinweis: Dieses Verfahren steht mit der Advanced Compression Option zur Verfügung.

Um bei `INSERT`-Operationen eine möglichst hohe Performance zu erhalten, wird nicht nach jeder Schreiboperation komprimiert, sondern ein neuer Block bleibt so lange unkomprimiert, bis die Daten die PCTFREE-Grenze erreicht haben (siehe Abbildung 2).

Abb. 2: Komprimierung mit OLTP Compression



Hinweis: Tabellen mit OLTP-Compression-Einstellung besitzen den Defaultwert 10 für **PCTFREE**, wohingegen Tabellen mit Basic Compression als Defaulteinstellung den Wert 0 haben.

Die häufig gestellte Frage nach der **Höhe der Komprimierungsrate** lässt sich nicht allgemein beantworten. Je nach Art der Daten, beispielsweise dem Anteil an redundanten Informationen, der Datenbank-Blockgröße und der Art des Ladevorgangs, können die Komprimierungsraten stark variieren. So ist es mitunter vorteilhaft, eine große Datenbank-Blockgröße zu verwenden beziehungsweise mit vorsortierten Daten zu arbeiten, um die Komprimierungsrate zu erhöhen.

Tipp aus der Praxis: Um einen Eindruck zu bekommen wie redundant die Daten in den einzelnen Spalten sind, kann die Spalte NUM_DISTINCT aus DBA_TAB_COLUMNS hilfreich sein. NUM_DISTINCT gibt an, wie viele verschiedene Werte in einer Spalte vorkommen. Setzt man diese Information ins Verhältnis zu der Anzahl der Zeilen, bekommt man einen recht guten Überblick über den Grad der Redundanz. Um eine gute Ratio zu erhalten, sollte die zu komprimierende Tabelle gegebenenfalls nach den Spalten mit den höchsten Redundanzen umsortiert werden.

Wie wird Basic Compression oder OLTP Compression eigentlich eingeschaltet? Generell können beide Komprimierungsarten auf Tablespace-, Tabellen- oder Partitionsebene mit dem entsprechendem CREATE- beziehungsweise ALTER-Kommando eingeschaltet werden. Da sich die Syntax-Schlüsselwörter beim CREATE beziehungsweise ALTER TABLE-Kommando mehrfach geändert haben, schadet ein Blick in das aktuelle SQL-Language-Reference-Handbuch nicht. Die folgenden Beispiele geben einen kleinen Überblick dazu.

OLTP Compression in 11g Release 1:

```
CREATE TABLE sales_history(…) COMPRESS FOR ALL OPERATIONS;
```

OLTP Compression in 11g Release 2:

```
CREATE TABLE sales_history(…) COMPRESS FOR OLTP;
```

OLTP Compression (auch Advanced Row Compression) in 12c:

```
CREATE TABLE sales_history(…) ROW STORE COMPRESS ADVANCED;
```

Basic Compression (auch Direct Load Compression) in 11g Release 1:

```
CREATE TABLE sales_history(…) COMPRESS FOR DIRECT_LOAD OPERATIONS;
```

Basic Compression in 11g Release 2:

```
CREATE TABLE sales_history(…) COMPRESS [BASIC];
```

Basic Compression in 12c:

```
CREATE TABLE sales_history(…) ROW STORE COMPRESS BASIC;
```

Wie man leicht erkennen kann, folgt die Syntax der Bezeichnung der Algorithmen. Die Syntax ist aufwärtskompatibel, das bedeutet, dass alle Syntaxvarianten in 12c verwendet werden können.

Auf Tablespace-Ebene wird mit dem Keyword `DEFAULT` und der anschließenden Angabe der Komprimierungsart festgelegt, dass alle Tabellen, die in diesen Tablespace geladen werden, der Komprimierungsart des Tablespace folgen. Separate Tabellen- und Partitionsklauseln können dieses Verhalten überschreiben. Das folgende Beispiel zeigt die Default-Tablespace-Einstellung für OLTP-Komprimierung in 11g Release 2.

```
CREATE TABLESPACE ... DEFAULT COMPRESS FOR OLTP;
```

Möchte man im Nachhinein die Inhalte von existierenden unkomprimierten Tabellen in komprimierte Tabellen umwandeln, kann man die Daten mit folgenden Kommandos in einer Einschritt-Operation umschichten und gleichzeitig komprimieren.

```
ALTER TABLE ... MOVE [TABLE_COMPRESSION_CLAUSE]

ALTER TABLE ... MOVE PARTITION|SUBPARTITION COMPRESS
[TABLE_COMPRESSION_CLAUSE]
```

Allerdings erfordern diese Kommandos bis einschließlich 11g Release 2 DML-Sperren auf der Tabelle. Falls diese Operationen online ohne Sperren erfolgen sollen, muss man mit dem Package `DBMS_REDEFINITION` arbeiten. Wer partitions- beziehungsweise subpartitionsweise arbeiten kann, kann von der Neuerung in 12c profitieren. Die Syntaxerweiterung ONLINE ermöglicht die Ausführung ohne blockierende DML-Sperren (TM Lock). Gleichzeitige DML-Operationen, die normalerweise in 11g durch die entsprechenden Locks blockiert wären, sind nun möglich. Folgendes Beispiel zeigt, wie in 12c eine Partition ONLINE ohne Sperren in das OLTP-Komprimierungsformat überführt werden kann.

```
ALTER TABLE sales_big MOVE PARTITION sales_q4_2001 ROW STORE
COMPRESS ADVANCED ONLINE;
```

Tipp aus der Praxis: Nicht verwechseln sollte man die Operationen `ALTER TABLE MOVE COMPRESS` mit dem Kommando `ALTER TABLE ... COMPRESS`. Ohne Verwendung des Schlüsselworts MOVE wird nur die sogenannte Table Property

geändert, das heißt es werden Speichereinstellungen für die zukünftige Nutzung definiert. Nur nachträglich eingefügte Zeilen werden komprimiert abgelegt. Bestehende Zeilen werden nicht verändert.

Tabellenpartitionen, die USABLE Bitmap-Indizes enthalten, müssen gesondert behandelt werden, falls die Tabelleneigenschaft von unkomprimiert nach komprimiert geändert werden soll. Dokumentiert ist dieses Verhalten auch im *VLDB and Partitioning Guide*. Das Vorgehen sieht dabei folgendermaßen aus:

1) Setze Bitmap-Index UNUSABLE.

2) Verändere die Komprimierungseigenschaft der Tabelle.

3) Führe ein REBUILD des Index durch.

Missachtet man diese Regeln und wendet beispielsweise ein ALTER TABLE MOVE COMPRESS an, ohne die Indizes vorher UNUSABLE zu setzen, führt dies zu folgender Fehlermeldung:

```
SQL> alter table part_test move partition part1 compress;
alter table part_test move partition part1 compress
           *
ERROR at line 1:
```

```
ORA-14646: Specified alter table operation involving compression
cannot be performed in the presence of usable bitmap indexes
```

Wie die Fehlermeldung beschreibt, müssen die Indizes zuerst UNUSABLE gesetzt werden; erst dann lässt sich die Operation durchführen. Dieser Vorgang ist deswegen notwendig, da bei komprimierten Tabellensegmenten potenziell mehr Zeilen in einem Datenblock adressiert werden können. Das nachfolgende REBUILD des Index berücksichtigt diese veränderte Sachlage. Diese Regeln gelten übrigens nur für Bitmap-Indizes nicht für B*Indizes.

Die Komprimierungseinstellungen der einzelnen Datenbanktabellen und Partitionen lassen sich über die zusätzlichen Spalten `COMPRESSION` und `COMPRESS_FOR` in `DBA_TABLES` und `DBA_TAB_PARTITIONS` beziehungsweise `DBA_TAB_SUBPARTITIONS` überprüfen. Für Tablespaces sind dies `DEF_TAB_COMPRESSION` und `COMPRESS_FOR`.

```
SELECT table_name, compression, compress_for FROM user_tables;

TABLE_NAME                COMPRESS COMPRESS_FOR
------------------------- -------- ------------
MY_CUSTOMERS_CPR          ENABLED  BASIC
ACO_CUSTOMERS             ENABLED  OLTP
...
```

Bis Oracle 11g ist zu bachten, dass Tabellen mit mehr als 255 Spalten, Index-Organized Tables, External Tables und Cluster Tables nicht komprimiert werden können. In 12c wurde die Spaltengrenze von 255 für Tabellen mit Advanced Row Compression allerdings aufgehoben. Im aktuellen Handbuch *Oracle Database SQL Language Reference* kann man die entsprechenden Einschränkungen nachlesen. Statements, die zu Row Chaining führen, können unter Umständen zu einer Vergrößerung der Tabelle führen. Die Master Note für OLTP Compression (Doc ID 1223705.1) gibt Aufschluss über den aktuellen Stand der Dinge und auf gegebenenfalls zu installierende Patches.

Der Einsatz von Komprimierung kann nicht nur zur Speicherreduktion, sondern auch zu einem Performancegewinn führen. Der Buffer Cache kann besser ausgenutzt werden, I/O intensive Operationen können somit beschleunigt werden. Allerdings kann es bei DML-Operationen auch zu höheren Redo- beziehungsweise CPU-Anforderungen kommen. In jedem Fall sollten vor dem Einsatz in der Praxis entsprechende Tests durchgeführt werden.

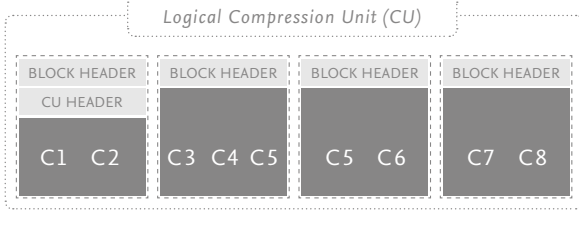
2.3 HYBRID COLUMNAR COMPRESSION (HCC)

Der Einsatz spezieller Storage wie Exadata-Storage-Systeme, Sun ZFS Storage Appliance (Oracle NAS Storage) oder

Axiom Storage (Oracle SAN Storage) ermöglicht die Verwendung von weiteren Komprimierungsalgorithmen, die unter dem Begriff **Hybrid Columnar Compression** zusammengefasst werden. Die Non-Exadata Storage-Systeme (ZFS und Pillar Storage) erfordern dabei eine Datenbankversion ab 11.2.0.3.

Oracle verwendet bei HCC eine Kombination aus zeilen- und spaltenbasierter Speicherung und die Verwendung von speziellen Komprimierungsalgorithmen. Die Datensätze werden in logische Compression-Einheiten (Logical Compression Unit) (siehe Abbildung 3) aufgeteilt und innerhalb einer Einheit nach Spalten sortiert und danach komprimiert. Die Spaltenwerte einer Gruppe werden dann im selben Datenblock abgespeichert. Auf diese Art und Weise kann eine sehr effiziente Komprimierung erfolgen, und es können hohe Komprimierungsraten – höher als im Fall von BASIC und OLTP Compression – erreicht werden.

Abb. 3: Logical Compression Unit



Vorgesehen sind diese Verfahren nur für Daten, die nicht häufig verändert werden. Sie sind – wie bei der BASIC Compression – ausschließlich für Bulk Loads implementiert.

Innerhalb von HCC gibt es unterschiedliche Komprimierungsverfahren. Diese können durch die Schlüsselwörter `ARCHIVE` und `QUERY` mit dem Zusatz `LOW` beziehungsweise `HIGH` angezeigt werden. Bei typischer Warehouse-Verwendung mit Low Concurrency empfiehlt sich der Einsatz von `QUERY`. Wie das Schlüsselwort `ARCHIVE` schon anzeigt, sollten Daten, die nicht mehr verändert werden und für eine Langzeitspeicherung vorgesehen sind, mit `ARCHIVE HIGH` gespeichert werden. Die Komprimierungsverfahren unterscheiden sich intern in der Verwendung unterschiedlicher Compression-Algorithmen und unterschiedlich großer Logical Compression Units.

Folgende Beispiele zeigen die Verwendung der Syntax. Auch hier, wie bei der BASIC und OLTP Compression, hat sich die Syntax in 12c verändert.

HCC mit Query (Default ist HIGH) 11g Release 2:

```
CREATE TABLE mass(...) COMPRESS FOR QUERY [LOW|HIGH]
```

HCC mit Query (Default ist HIGH) 12c:

```
CREATE TABLE mass(...) COLUMN STORE COMPRESS FOR QUERY[LOW|HIGH]
```

HCC mit Archive (Default ist LOW) 11g Release 2:

```
CREATE TABLE mass(...) COMPRESS FOR ARCHIVE [LOW|HIGH]
```

HCC mit Archive (Default ist LOW) 12c:

```
CREATE TABLE mass(...) COLUMN STORE COMPRESS FOR ARCHIVE[LOW|HIGH]
```

Wie zu erwarten, ist die Compression-Ratio bei der Verwendung von `ARCHIVE` höher als bei `QUERY`.

Tipp aus der Praxis: Um einen flexiblen Einsatz der Komprimierungsarten zu gewährleisten, werden in der Praxis häufig unterschiedliche Algorithmen für unterschiedliche Partitionen verwendet.

Am Ende dieses Kapitels soll noch auf die neuen ILM Features in 12c hingewiesen werden, die je nach Art der Verwendung der Daten eine automatische Komprimierung und/oder Verlagerung der Daten pro Partition oder Tabelle ermöglicht (siehe Informationen in Kapitel 6).

3 Index-Komprimierung

Eine Komprimierung im Index wird automatisch bei **Bitmap-Indizes** angewendet, die speziell in Warehouse-Anwendungen bei Spalten mit geringer Kardinalität zum Einsatz kommen.

Bitmap-Indizes speichern eine Bitmap für jeden Index-Schlüssel statt einer Liste von ROWIDS. Jedes Bit in dieser Bitmap adressiert eine ROWID und somit eine Zeile in der Tabelle. Bitmap-Indizes benötigen keine Komprimierungsalgorithmen, da diese Art von Indizes einen sehr geringen Speicherbedarf hat. Wegen des speziellen Locking-Verhaltens bietet die Verwendung von Bitmap-Indizes besonders bei lesenden Zugriffen Vorteile.

Weniger bekannt ist die Tatsache, dass auch „normale" B*Tree-Indizes komprimiert werden können. Im Gegensatz zum Bitmap-Index ist kein spezielles Locking erforderlich, sodass diese ohne Einschränkung im OLTP-Umfeld für unterschiedliche Anwendungen, so zum Beispiel auch im ERP-Bereich, eingesetzt werden können. Die sogenannte **Index Key Compression** ist bereits seit Oracle 9i verfügbar und steht für B*Tree-Indizes und IOTs (Index Organized Tables) zur Verfügung.

Das Prinzip der Index Key Compression beruht dabei auf der Eliminierung von sich wiederholenden Schlüsselwerten (auch Präfix genannt) eines **nonunique single column**- oder eines **unique multicolumn**-Index. Zusammengesetzte Schlüssel (unique multicolumn) werden dabei in einen Präfix- und einen Suffix-Anteil unterteilt, wobei der Suffix-Anteil den eindeutigen Teil des Index-Schlüssels repräsentiert.

Wenn sich Schlüsselwerte im Präfix-Anteil des Index wiederholen, werden diese Werte nur einmal gespeichert und vom Suffix referenziert. Bei „nonunique single column"-Schlüssel wird die ROWID-Information genutzt, um den Schlüssel eindeutig zu machen. Der Präfix-Anteil wird also durch die sich wiederholenden Werte repräsentiert; der verbleibende Anteil, die ROWIDS, stellen dann den Suffix-Anteil dar. Präfix- und Suffix-Werte befinden sich grundsätzlich im gleichen Block. Diese Speicherung kann zu einer starken Reduzierung der Index Leaf Pages und damit der Anzahl der I/O-Operationen bei einem Indexzugriff führen.

Die Komprimierung wird beim Erzeugen des Index (`CREATE INDEX`) oder mit einem `ALTER INDEX REBUILD`-Kommando eingestellt. Bei IOTs wird das `CREATE TABLE` beziehungsweise das `ALTER TABLE MOVE` Kommando verwendet.

```
CREATE INDEX i_name ON t1(col1,col2,col3,col4) COMPRESS 2;
ALTER INDEX i_name REBUILD [ONLINE] COMPRESS 1;
```

Die Klauseln `COMPRESS 2` und `COMPRESS 1` geben dabei die Anzahl der Präfixspalten an. Die Default-Präfixlänge für unique Indizes ist die Anzahl der Spalten minus 1, für nonunique Indizes beträgt dieser Wert die Anzahl der Spalten.

Index-Komprimierung lässt sich über die Standard Views wie `USER_INDEXES` monitoren.

```
SELECT index_name, owner, compression, prefix_length
FROM dba_indexes WHERE PREFIX_LENGTH is not null AND owner='US';

INDEX_NAME                OWNER      COMPRESS PREFIX_LENGTH
------------------------- ---------- -------- -------------
DR$TEXT_IDX$X             US         ENABLED              2
DR$TESTIDX$X              US         ENABLED              2
DR$MYTABLE_OTXML$X        US         ENABLED              2
DR$FILTER_TEST_IDX$X      US         ENABLED              2
DR$IDX_KOMMENTARE$X       US         ENABLED              2
CT_COMP                   US         ENABLED              1
```

Wie findet man nun die optimale Komprimierung und Präfixlänge? Zur Illustration werden folgende Index-Leaf-Einträge mit den zugehörigen ROWIDS betrachtet:

Arzt	Zahnarzt	AAAPvCAAFAAAAFaAAa
Restaurant	Italienisch	AAAPvCAAFAAAAFaAAa
Arzt	Internist	AAAPvCAAFAAAAFaAAl
Restaurant	Bayerisch	AAAPvCAAFAAAAFaAAm
Restaurant	Indisch	AAAPvCAAFAAAAFaAAq

Der Index wird nun mit der Klausel `COMPRESS 1` gespeichert. Die Schlüsselwerte „Restaurant“ (hier P 0) und „Arzt“ (hier P 1) werden im Leaf-Block genau einmal abgespeichert.

Folgende Tabelle zeigt das Ergebnis dieser Speicherung in einer schematischen Darstellung.

P 0	Restaurant
P 1	Arzt

P 0	Italienisch	AAAPvCAAFAAAAFaAAa
P 0	Bayerisch	AAAPvCAAFAAAAFaAAm
P 0	Indisch	AAAPvCAAFAAAAFaAAq
P 1	Zahnarzt	AAAPvCAAFAAAAFaAAa
P 1	Internist	AAAPvCAAFAAAAFaAAl

Die Güte der Komprimierungsrate kann stark variieren und ist zum Beispiel abhängig von der richtigen Anzahl der komprimierten Spalten, den Werteausprägungen und der Anordnung der Spalten im Index. Falls das Umsortieren der Spalten im Index möglich ist, kann dies zu höheren Komprimierungsraten führen. Um zu beurteilen, ob und welche Indexkomprimierung für die entsprechenden Indexkandidaten sinnvoll ist, kann das Kommando `ANALYZE INDEX` hilfreich sein.

```
ALTER INDEX ba_kat_ort VALIDATE STRUCTURE;
```

Bei der Ausführung wird der Index analysiert und das Ergebnis in der dynamischen Tabelle INDEX_STATS eingetragen. Die Spalte OPT_CMPR_COUNT gibt die optimale Key-Compression-Länge an, OPT_CMPR_PCTSAVE die Speichereinsparung in Prozent.

Die folgende Abfrage zeigt das Ergebnis aus der INDEX_STATS-Tabelle nach Anwendung des ANALYZE INDEX Kommandos – angewendet auf einen unkomprimierten Index.

```
SELECT name, blocks, br_blks, lf_blks, opt_cmpr_pctsave,
opt_cmpr_count
FROM index_stats;

NAME       BLOCKS  BR_BLKS  LF_BLKS   OPT_CMPR_PCTSAVE  OPT_CMPR_COUNT
---------- ------- -------- --------- ----------------- --------------
BA_KAT_ORT    768        3       643                46              2
```

Die Komprimierung des Index mit COMPRESS 2 würde eine Einsparung von 46 Prozent des Speicherplatzes bewirken. In einem zweiten Test wird der Index mit COMPRESS 2 aufgebaut. Das Ergebnis in der Tabelle INDEX_STATS zeigt nun, dass es kein weiteres Einsparungspotenzial gibt. Die Blockanzahl hat sich – wie zu erwarten war – fast halbiert.

```
SELECT name, blocks, br_blks, lf_blks, opt_cmpr_pctsave,
opt_cmpr_count
```

```
FROM index_stats;
NAME       BLOCKS  BR_BLKS  LF_BLKS   OPT_CMPR_PCTSAVE  OPT_CMPR_COUNT
---------- ------- -------- --------- ----------------- --------------
BA_KAT_ORT    384        1       345                 0              2
```

Die Tabelle `INDEX_STATS` ist dynamisch und speichert nur den letzten Eintrag des `ANALYZE INDEX`-Kommandos. Um eine Historie zu speichern, sollte man eine Hilfstabelle anlegen und diese mit den entsprechenden Ergebnissen füllen. Ein Nachteil dieser Methode ist, dass das `ANALYZE INDEX`-Kommando weder ONLINE noch PARALLEL durchführbar ist.

Wichtiger Hinweis: Da bei der Ausführung ein DML Lock erfolgt, können DML-Operationen besonders bei der Verwendung von großen Indizes stark beeinträchtigt werden.
Ein alternatives Skript zur Indexanalyse liefert die MOS Note 989186.1. Allerdings wird hier keine Analyse über die optimale Key-Compression-Länge durchgeführt.

4 Tabellen mit unstrukturierten Daten

Unstrukturierte Daten vom Datentyp XML, CLOB und BLOB sind in der Regel sehr speicherintensiv. Aus diesem Grund ist eine Komprimierung dieser Daten sinnvoll. Mit Oracle Database 11g steht ein neuer Datentyp für die Speicherung von Large Objects in allen Editionen der Datenbank zur Verfügung, die sogenannten **SecureFiles**. Im Gegensatz zur „alten" LOB-Technologie, die ab

11g mit dem neuen Schlüsselwort BASICFILE – im Gegensatz zu SECUREFILE für die neue SecureFile-Speicherung – definiert werden kann, bieten SecureFiles verbesserte Performance, vereinfachtes Management und erweiterte Funktionen unter Nutzung von zusätzlichen Datenbank-Optionen. Der Initialisierungsparameter DB_SECUREFILE mit den Werten PERMITTED, NEVER, FORCE, ALWAYS und IGNORE steuert dabei die Verwendung. In 11g ist der Default PERMITTED – also erlaubt – in 12c hingegen PREFERRED – also Verwendung von SecureFiles, falls nichts anderes spezifiziert wurde.

Eine Eigenschaft von Oracle SecureFiles ist die Möglichkeit, Komprimierung einzuschalten. Dies erfordert allerdings den Einsatz der Advanced Compression Option. Dabei sind folgende Einstellungen bei der Komprimierung möglich:

- DEDUPLICATE: LOBs mit identischem Inhalt werden physikalisch nur einmalig gespeichert. Diese Einstellung ist besonders sinnvoll bei der Nutzung von großen LOBs, die mehrfach gespeichert werden wie beispielsweise E-Mail-Attachments.
- COMPRESS HIGH (beziehungsweise MEDIUM, LOW): Reduzierung des Speicherbedarfs von LOBs durch Komprimierung. Diese Komprimierung wird durch einen Standardalgorithmus durchgeführt und kann wahlweise mit einer

hohen, mittleren oder niedrigen Komprimierungsrate durchgeführt werden. Die LOB-Komprimierung mit Parametereinstellung `HIGH` hat dabei einen höheren CPU-Bedarf als die Komprimierung der LOBs mit der SecureFile-Standardkomprimierung `MEDIUM` oder mit der Komprimierung `LOW`.

Die LOB-Komprimierung ist unabhängig von der Tabellenkomprimierung und wird beim `CREATE TABLE` oder `ALTER TABLE` separat über die SecureFile LOB-Storage-Klausel angegeben.

```
CREATE TABLE nachrichten_text (dok_id NUMBER,...,text_info CLOB)
LOB (text_info)
STORE AS SECUREFILE(DEDUPLICATE COMPRESS HIGH DISABLE STORAGE IN
ROW)
```

Folgende Abfrage gibt Aufschluss über die Speicherung der LOBs.

```
SELECT table_name, column_name, compression, deduplication, in_row
FROM user_lobs WHERE securefile='YES';
TABLE_NAME          COLUMN_NAME     COMPRE DEDUPLICATION   IN_
------------------- --------------- ------ --------------- ---
DR$FT_NAMES1$I      TOKEN_INFO      NO     NO              YES
DR$FT_NAMES1$R      DATA            NO     NO              YES
NACHRICHTEN_TEXT    TEXT_INFO       HIGH   LOB             NO
...
```

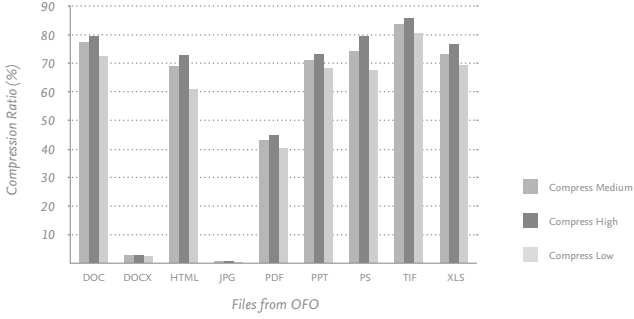
Unabhängig von der Komprimierung sollte man auch Überlegungen zur generellen LOB-Speicherung anstellen – beispielsweise zur in-line- beziehungsweise out-of-line-Speicherung der LOBs. Der Default ist in-line Speicherung (ENABLE STORAGE IN ROW). Das bedeutet LOBs, die kleiner als 4000 Bytes sind, werden im Tabellensegment selbst gespeichert; größere LOBs hingegen in einem zusätzlichen LOB-Segment. Manchmal kann diese „gemischte" Speicherung allerdings von Nachteil beim Zugriff sein. In diesen Fällen ist die out-of-line-Speicherung (DISABLE STORAGE IN ROW) – alle LOBs in einem zusätzlichen Segment zu speichern – vorzuziehen und kann sogar zu großen Performancesteigerungen führen.

Je nach gespeichertem Format (HTML, Text, ASCII, PDF, GIF usw.) sind die Komprimierungsraten unterschiedlich. Zweistellige Werte bei der **Komprimierungsrate** können dabei allerdings nichts Ungewöhnliches sein. Speziell bei I/O-intensiven Applikationen kann der Einsatz von Komprimierung von Vorteil sein. Ein Beispiel dafür liefert die 11g-Installation bei der FIZ CHEMIE in Berlin, die eine Performancesteigerung sowie zusätzliche Speicherplatzeinsparung erreichen konnte. Dabei wurde mittels einer Mark-up-Abfrage, die summarisch betrachtet das ressourcenintensivste Statement der getesteten Anwendung darstellt, das Laufzeitverhalten anhand einer Tabelle mit ca. 200000

Zeilen analysiert. Allein durch Einsatz der I/O-optimierten SecureFiles wurde bereits eine Performancesteigerung von 30 Prozent erzielt. In Verbindung mit Advanced Compression ergab die Testreihe dann eine radikale Verbesserung der Laufzeit. Diese wurde von ursprünglich 59 Minuten auf 12 Minuten reduziert – und das bei einer gleichzeitigen Verringerung des Speicherplatzes auf 25 Prozent des ursprünglichen Speicherbedarfs. Andere Beispiele des Einsatzes sind Daten aus dem SPATIAL-Bereich. Auch hier kann die Komprimierung zu großen Speichereinsparungen führen.

Umgekehrt macht es keinen oder wenig Sinn, schon vorkomprimierte Formate zu komprimieren. Folgende Übersicht gibt einen Überblick über die möglichen Kompressionsraten bei Verwendung unterschiedlicher Formate.

Abb. 4: Mögliche Compression Ratio bei Nutzung unterschiedlicher Formate



Um eine gute Komprimierungsrate zu erzielen, ist die richtige Wahl der Blockgröße entscheidend. Bei Large Objects lassen sich mithilfe der GETLENGTH-Funktion unkomprimierte LOB-Größen abfragen. Handelt es sich bei den Daten nicht schon um stark komprimierte Formate wie beispielsweise das GIF-Format, kann eine Halbierung der durchschnittlichen LOB-Größe die ideale Blockgröße für die komprimierten Lobs darstellen.

Um die Ratio zu evaluieren, kann man entweder USER_SEGMENTS (siehe Beispiel) für die komprimierte beziehungsweise unkomprimierte Version der Tabelle abfragen oder eine blockgenaue detaillierte Speicheraufteilung mit dem Package DBMS_SPACE berechnen.

```
SELECT bytes/(1024*1024) groesse FROM user_segments
WHERE segment_name='SECURE_TABLE'
UNION
SELECT bytes/(1024*1024) FROM user_segments
WHERE segment_name IN
(SELECT segment_name FROM user_lobs WHERE table_name='SECURE_
TABLE')
    GROESSE
    ----------
    34
    3463
```

```
SELECT bytes/(1024*1024) groesse FROM user_segments
WHERE segment_name='COMPRESS_TABLE'
UNION
SELECT bytes/(1024*1024) FROM user_segments
WHERE segment_name IN
(SELECT segment_name FROM user_lobs WHERE table_name='COMPRESS_
TABLE');
    GROESSE
    ---------
    19
    472.25
```

Tipp aus der Praxis: Hat man vorab die Möglichkeit, auf die Daten mit dem *zip*-Kommando zuzugreifen, kann man sich schon einen ungefähren Eindruck von der möglichen Komprimierungsrate verschaffen.

In 12c bietet der Compression Advisor zusätzliche Möglichkeiten im Voraus zu prüfen, welche Kompressionsraten bei LOB Compression zu erwarten sind.

Wie migriert man nun diesen Datentyp? Da kein `ALTER TABLE MODIFY`-Kommando zur Migration zur Verfügung steht, bietet sich entweder eine Online-Migration mit dem Paket `DBMS_REDEFINITION` oder eine Neuanlage der Tabelle und Kopie der Daten an.

Nicht ganz unerwähnt soll das Package UTL_COMPRESS bleiben, das mit Oracle Version 10g eingeführt wurde. Mit UTL_COMPRESS lassen sich RAW, BLOB oder BFILE-Daten komprimieren beziehungsweise dekomprimieren. Das Resultat von UTL_COMPRESS entspricht dabei dem der Werkzeuge *compress* in Unix-Umgebungen und *zip* in Windows-Umgebungen. UTL_COMPRESS erfordert allerdings im Unterschied zur automatischen Komprimierung und Dekomprimierung der SecureFile LOBs eine vollständige Programmierung in PL/SQL für die Komprimierung und die Dekomprimierung der Daten und ist unabhängig von der Speicherung in der Datenbank. Ausführliche Beispiele dazu finden sich in der MOS Note 249974.1.

5 Compression Advisor

Wie kann man den Grad der Speicherplatzeinsparung feststellen? Naheliegend ist, neue Segmente mithilfe der neuen Speichereinstellung zu erstellen und dann den Quotient aus nicht komprimierten und komprimierten Objekten – die sogenannte Compression Ratio – zu berechnen. Eine Alternative ist die Nutzung des Compression Advisors. Seit **Oracle Database 11g Release 2** steht standardmäßig der Compression Advisor in der Datenbank zur Verfügung. Ohne zusätzliche Installation ist dieser Advisor über das

Package DBMS_COMPRESSION **in jeder Edition** sofort einsetzbar.

Hat man allerdings noch keinen Zugriff auf eine 11g-Release-2-Installation, kann man Unterstützung durch eine zusätzliche Package-Installation erhalten. Download und Nutzungsbeschreibung dazu finden sich auf OTN (siehe Punkt 12: Weitere Informationen). Im Unterschied zur 11g-Release-2-Funktionalität muss man bei der Nutzung mit einigen Einschränkungen rechnen. So ist dieses Package beispielsweise nicht für partitionierte Tabellen geeignet. Außerdem werden nicht alle Komprimierungsalgorithmen unterstützt.

Im Gegensatz zum „alten" Compression Advisor ist der Compression Advisor in 11g Release 2 auch für partitionierte Tabellen und für alle vorhandenen Komprimierungsalgorithmen (auch HCC) einsetzbar. Eine grafische Implementierung existiert allerdings noch nicht, sodass man sich mit der Syntax auseinandersetzen muss. Für partitionierte Tabellen bedeutet dies übrigens, dass man pro Partitionssegment eine Berechnung durchführen muss.

Folgende Fragestellungen können mit DBMS_COMPRESSION gelöst werden.

- Sind alle Blöcke meiner Tabellen komprimiert? Und welcher Komprimierungstyp wurde verwendet? Die Antwort liefert die Prozedur `GET_COMPRESSION_TYPE`. Über die einfache Eingabe einer ROWID kann der Advisor ermitteln, ob und welche Komprimierung verwendet wurde.
- Welche Komprimierung (Ratio) kann beim Einsatz der unterschiedlichen Komprimierungstypen erwartet werden? Hier kann die Prozedur `GET_COMPRESSION_RATIO` weiterhelfen. Nach Eingabe einer Komprimierungsart wird die Ratio des Segments ermittelt. Zusätzlich zu den Komprimierungstypen OLTP und BASIC können auch HCC-Komprimierungstypen wie QUERY LOW, QUERY HIGH, ARCHIVE LOW und ARCHIVE HIGH **ohne** Zugriff auf ein Exadata-Storagesystem getestet werden.

Auch Indizes und unstrukturierte Daten nehmen einen Anteil am gesamten Speicherplatz ein und sollten in die Berechnung einfließen. Der Compression Advisor liefert dazu allerdings keine Unterstützung in 11g Release 2. Erst ab 12c gibt es eine Erweiterung für unstrukturierte Daten.

Realisierte Compression-Projekte zeigen übrigens, dass die Berechnung der Ratio durch den Compression Advisor sehr realistische Werte und Annäherungen zur tatsächlichen Speicherung liefert.

5.1 BESTIMMUNG DER COMPRESSION RATIO

Im ersten Fall soll die Compression Ratio in 11g Release 2 bestimmt werden. Der Aufruf der Prozedur `GET_COMPRESSION_RATIO` benötigt einige IN-Parameter und liefert das Ergebnis über die vorgegebenen OUT-Parameter. Der Parameter `COMPTYPE` kann dabei als Package-Konstante oder als Zahl (siehe Abbildung 5) angegeben werden.

Abb. 5: Komprimierungstypen und Konstanten in 11g Release 2

Constant	Type	Value	Description
COMP_NOCOMPRESS	NUMBER	1	No compression
COMP_FOR_OLTP	NUMBER	2	OLTP compression
COMP_FOR_QUERY_HIGH	NUMBER	4	High compression level for query operations
COMP_FOR_QUERY_LOW	NUMBER	8	Low compression level for query operations
COMP_FOR_ARCHIVE_HIGH	NUMBER	16	High compression level for archive operations
COMP_FOR_ARCHIVE_LOW	NUMBER	32	Low compression level for archive operations

Der zu überprüfende Komprimierungstyp entspricht im nächsten Beispiel der OLTP-Komprimierung (siehe Wert 2).

Besitzt die Tabelle Partitionen, muss die Ratio pro Partition berechnet werden. In folgendem Beispiel wird die Partition COSTS_Q1_1998 der Tabelle COSTS des Users SH überprüft. Möchte man die Ratio für alle Partitionen einer Tabelle berechnen, kann dies über eine Programmerweiterung in PL/SQL erreicht werden.

Hinweis: Damit die Durchführung funktioniert, müssen als Voraussetzung ausreichende Zugriffsrechte und Platz im Tablespace SCRATCHTBSNAME bereitgestellt werden. Am Besten führt man die Prozedur daher mit einem User aus, der DBA-Privilegien besitzt. (Beachten Sie bitte die Securityvorgaben in Ihrem Unternehmen.)

```
set serveroutput on
declare
  b_cmp      PLS_INTEGER;
  b_uncmp    PLS_INTEGER;
  row_cmp    PLS_INTEGER;
  row_uncmp   PLS_INTEGER;
  cmp_ratio   NUMBER;
  cmp_str    VARCHAR2(200);
begin
     DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
 SCRATCHTBSNAME     => 'USERS',
 OWNNAME            => 'SH',
```

```
 TABNAME                => 'COSTS',
 PARTNAME               => 'COSTS_Q1_1998',
 COMPTYPE               => 2,
 BLKCNT_CMP             => b_cmp,
 BLKCNT_UNCMP           => b_uncmp,
 ROW_CMP                => row_cmp,
 ROW_UNCMP              => row_uncmp,
 CMP_RATIO              => cmp_ratio,
 COMPTYPE_STR           => cmp_str);
 dbms_output.put_line('# Blocks compressed   => '|| b_cmp);
 dbms_output.put_line('# Blocks uncompressed  => '|| b_uncmp);
 dbms_output.put_line('Ratio          => '|| cmp_ratio);
 dbms_output.put_line('Komprimierungstyp    => '|| cmp_str);
end;
/
# Blocks compressed    => 10
# Blocks uncompressed  => 20
Ratio                  => 2
Komprimierungstyp      => Compress For OLTP
```

Das Ergebnis zeigt, dass durch den Einsatz von OLTP-Komprimierung eine Ratio von 2 erreicht werden kann.

Ein Blick auf Abbildung 5 zeigt, dass auch **Hybrid Columnar Compression** (HCC) mit dem Compression Advisor überprüft werden kann. Der Test ist möglich, ohne ein entsprechendes

Storagesystem zu besitzen, und ab Version 11.2.0.2 auch ohne zusätzliche Installation von Patches. Eine Tabelle kann also ohne weiteren Aufwand auf das Einsparungspotenzial von vier weiteren Komprimierungstypen überprüft werden.

Hinweis: Die Größe der Tabelle sollte für den Test mit HCC Komprimierung geeignet sein und mindestens 1000000 Zeilen enthalten.

Wie funktioniert die Berechnung der Ratio? Generell werden zwei temporäre Tabellen im Tablespace erstellt, der über den Parameter SCRATCHTBSNAME benannt wird. Die eine Tabelle enthält ein Stichprobe aus komprimierten Blöcken die andere aus unkomprimierten Blöcken. Nach der Berechnung der Ratio werden diese beiden Tabellen wieder automatisch gelöscht. Um die Verwendung der Ressourcen zu begrenzen, ist bei der HCC-Komprimierung übrigens die Anzahl der verwendeten Zeilen für das Sampling auf 1000000 Zeilen limitiert.

In 12c gibt es einige Veränderungen an DBMS_COMPRESSION. Beispielsweise wurden Parameter wie TABNAME und PARTNAME in OBJNAME und SUBOBJNAME umbenannt und die Liste der Konstanten erweitert (siehe Abbildung 6).

Abb.6: Komprimierungstypen und Konstanten in 12c

Constant	Type	Value	Description
COMP_NOCOMPRESS	NUMBER	1	No compression
COMP_ADVANCED	NUMBER	2	Advanced compression level
COMP_QUERY_HIGH	NUMBER	4	High compression level for query operations
COMP_QUERY_LOW	NUMBER	8	Low compression level for query operations
COMP_ARCHIVE_HIGH	NUMBER	16	High compression level for archive operations
COMP_ARCHIVE_LOW	NUMBER	32	Low compression level for archive operations
COMP_BLOCK	NUMBER	64	Compressed row
COMP_LOB_HIGH	NUMBER	128	High compression level for LOB operations
COMP_LOB_MEDIUM	NUMBER	256	Medium compression level for LOB operations
COMP_LOB_LOW	NUMBER	512	Low compression level for LOB operations
COMP_RATIO_LOB_MINROWS	NUMBER	1000	Minimum required number of LOBs in the object for which LOB compression ratio is to be estimated
COMP_BASIC	NUMBER	4096	Basic compression level
COMP_RATIO_LOB_MAXROWS	NUMBER	5000	Maximum number of LOBs used to compute the LOB compression ratio

COMP_RATIO_MINROWS	NUMBER	1000000	Minimum required number of rows in the object for which HCC ratio is to be estimated
COMP_RATIO_ALLROWS	NUMBER	-1	To indicate the use of all the rows in the object to estimate HCC ratio
OBJTYPE_TABLE	PLS_ INTEGER	1	Identifies the object whose compression ratio is estimated as of type table

Das Aufruf von DBMS_COMPRESSION sieht in 12c dann folgendermaßen aus:

```
...
    DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
  SCRATCHTBSNAME        => 'USERS',
  OWNNAME               => 'SH',
  OBJNAME               => 'COSTS',
  SUBOBJNAME            => 'COSTS_Q1_1998',
  COMPTYPE              => 2,
  BLKCNT_CMP            => b_cmp,
  BLKCNT_UNCMP          => b_uncmp,
  ROW_CMP               => row_cmp,
  ROW_UNCMP             => row_uncmp,
  CMP_RATIO             => cmp_ratio,
  COMPTYPE_STR          => cmp_str);
...
```

Neu in 12c ist der Einsatz für unstrukturierte Daten. Laut Abbildung 5 können dabei die Level LOW (512), MEDIUM (256)und HIGH (128) für SecureFile komprimierte Daten überprüft werden.

Das folgende Beispiel zeigt eine mögliche Implementierung. Mit `COMPTYPE` 128 wird der Compression Level HIGH überprüft; die LOBs im Beispiel sind in der Spalte TEXT gespeichert.

```
set serveroutput on
declare
  b_cmp       PLS_INTEGER;
  b_uncmp     PLS_INTEGER;
  lob_cnt     PLS_INTEGER;
  cmp_ratio   NUMBER;
  cmp_str     VARCHAR2(200);
begin
     DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
 SCRATCHTBSNAME    => 'USERS',
 TABOWNER          => 'SH',
 TABNAME           => 'BASIC_LOB',
 LOBNAME           => 'TEXT',
 PARTNAME          => '',
 COMPTYPE          => 128,
 BLKCNT_CMP        => b_cmp,
```

```
 BLKCNT_UNCMP       => b_uncmp,
 LOBCNT             => lob_cnt,
 CMP_RATIO          => cmp_ratio,
 COMPTYPE_STR       => cmp_str);
end;
/
Sampling percent: 2.5
Uncomp blocks: 1246  Comp blocks: 638
Number of lobs sampled: 4980
compression ratio: 1.9
PL/SQL procedure successfully completed.
```

Von den ungefähr 200 000 gespeicherten LOBs der Tabelle BASIC_LOB wurden ca. 5000 LOBs überprüft. Die Ratio ist circa 2.

5.2 BESTIMMUNG DES KOMPRIMIERUNGSTYPS

Mithilfe der Funktion `GET_COMPRESSION_TYPE` kann sogar im Nachhinein der Komprimierungstyp einer Zeile bestimmt werden. Dazu ist die Angabe der ROWID einer Tabelle erforderlich. Das Ergebnis ist eine Zahl, die den jeweiligen Komprimierungstyp angibt (siehe Tabelle mit Komprimierungstypen). Folgendes Beispiel zeigt eine Anwendung.

```
SELECT rowid FROM sales WHERE rownum<10;
ROWID
------------------
AAAR+EAAFAAAAzDAAT
AAAR+EAAFAAAAzDADc
AAAR+EAAFAAAAzDADd
AAAR+EAAFAAAAzDADe
AAAR+EAAFAAAAzDADf
AAAR+EAAFAAAAzDADg
AAAR+EAAFAAAAzDADh
AAAR+EAAFAAAAzDADi
AAAR+EAAFAAAAzDAED
SELECT
dbms_compression.get_compression_type('SH','SALES','AAAR+EAAFAAAAzDAED')
typ FROM dual;
       TYP
----------
         2
```

Der Wert 2 entspricht der OLTP Komprimierung.

5.3 RÜCKRECHNEN VON KOMPRIMIERTEN DATEN

Nun stellt sich die Frage, ob man diese Berechnung auch dazu verwenden kann, eine **Rückrechnung** durchzuführen? Das heißt, man gibt ein komprimiertes Segment vor und möchte berechnen, wie groß das unkomprimierte Segment war oder – bei

dauerndem Verzicht auf die Komprimierung – wie groß das Segment werden würde. Auch dies kann mit der Prozedur `GET_COMPRESSION_RATIO` bewerkstelligt werden, indem man als Eingabewert den **aktuellen** Komprimierungstyp angibt. Da immer eine unkomprimierte Version berechnet und ausgegeben wird, können wir auf diese Weise die ursprüngliche Segmentgröße abschätzen. Dies wird im Folgenden kurz demonstriert.

Als Beispiel dient die Tabelle CUSTOMERS_BIG, die unkomprimiert vorliegt. Um eine Kontrolle über die Güte des Ergebnisses zu haben, wird die Segmentgröße vor der Komprimierung abgefragt.

```
SELECT segment_name, bytes/1024/1024, blocks FROM user_segments
WHERE segment_name='CUSTOMERS_BIG';
SEGMENT_NAME         BYTES/1024/1024     BLOCKS
-------------------- --------------- ----------
CUSTOMERS_BIG                     24       3072
```

Danach wird eine OLTP-Komprimierung durchgeführt und die Segmentgröße nach der Komprimierung überprüft.

```
ALTER TABLE customers_big MOVE COMPRESS FOR oltp;
SELECT segment_name, bytes/1024/1024, blocks FROM user_segments
WHERE segment_name='CUSTOMERS_BIG';
```

```
SEGMENT_NAME          BYTES/1024/1024     BLOCKS
--------------------- --------------- ----------
CUSTOMERS_BIG                      13       1664
```

Im nächsten Schritt führen wir die Rückrechnung mithilfe des Compression Advisors durch. Der Parameter `COMPTYPE` erhält den Wert `DBMS_COMPRESSION.COMP_FOR_OLTP` – dies entspricht der OLTP-Komprimierung, die wir auf die Tabelle angewendet haben. Statt der Konstanten `DBMS_COMPRESSION.COMP_FOR_OLTP` kann natürlich auch die Funktion `DBMS_COMPRESSION.GET_COMPRESSION_TYPE` verwendet werden.

```
set serveroutput on
declare
  b_cmp       PLS_INTEGER;
  b_uncmp     PLS_INTEGER;
  row_cmp     PLS_INTEGER;
  row_uncmp   PLS_INTEGER;
  cmp_ratio   NUMBER;
  cmp_str     VARCHAR2(200);
begin
DBMS_COMPRESSION.GET_COMPRESSION_RATIO (
  SCRATCHTBSNAME    =>'USERS',
  OWNNAME           =>'SH',
  TABNAME           =>'CUSTOMERS_BIG',
```

```
  PARTNAME             =>'',
  COMPTYPE             =>DBMS_COMPRESSION.COMP_FOR_OLTP,
  BLKCNT_CMP           =>b_cmp,
  BLKCNT_UNCMP         =>b_uncmp,
  ROW_CMP              =>row_cmp,
  ROW_UNCMP            =>row_uncmp,
  CMP_RATIO            =>cmp_ratio,
  COMPTYPE_STR         =>cmp_str);
dbms_output.put_line('# Blocks compressed     => '||  b_cmp);
dbms_output.put_line('# Blocks uncompressed   => '||  b_uncmp);
dbms_output.put_line('Ratio                   => '||  cmp_ratio);
dbms_output.put_line('Compression-Typ         => '||  cmp_str);
end;
/
# Blocks compressed     => 1529
# Blocks uncompressed   => 2877
Ratio                   => 1.8
Compression-Typ         => "Compress For OLTP"
```

Wir erhalten als Ergebnis für die unkomprimierte Tabelle (siehe „Blocks uncompressed“) 2877 Blöcke. Das Resultat entspricht zwar nicht zu 100 Prozent der ursprünglichen Größe, kann aber eine gute Hilfestellung bei einer Abschätzung liefern.

6 Information Lifecycle Management

Information Lifecycle Management (kurz ILM) ist kein neues Schlagwort, sondern ein gängiger Begriff um den „Lebenszyklus“ von Daten zu beschreiben. ILM hat zum Ziel, die Speicherung von Informationen entsprechend ihrem Wert und ihrer Nutzung optimal auf dem jeweils kostengünstigsten Speichermedium – im besten Fall automatisch – zu platzieren; so die Definitionen im Web.

Die Techniken zur Realisierung von ILM wie Partitionierung, Speicherplatzeinsparung durch Komprimierung, Verlagerung von Daten und Virtual Private Database (kurz VPD) für die unterschiedlichen Sichtweisen auf die Daten sind schon lange Bestandteil der Oracle-Datenbank und werden mit jedem Release weiterentwickelt.

Allerdings musste eine Automatisierung bisher über eigene Programme oder spezielle Werkzeuge implementiert werden, nachdem die Daten vorab manuell kategorisiert worden sind. Neu in Oracle Database 12c ist nun die **vollständige Integration** von ILM Features in die Datenbank. Keine zusätzlichen Werkzeuge oder Skripte zur Implementierung sind notwendig.

Wichtige Voraussetzung für die Verwendung dieser neuen Technologie ist die Lizenzierung der **Advanced Compression Option**. Im Wesentlichen handelt es sich dabei um zwei

neue Features – die **Heat Map** und die **automatische Datenoptimierung** (Englisch **A**utomatic **D**ata **O**ptimization). Die Heat Map „trackt" Veränderungen und Abfragen auf Zeilen und Segmentebene und gibt einen detaillierten Überblick über den Zugriff auf die Daten. Die automatische Datenoptimierung verlagert und/oder komprimiert die Daten gemäß nutzerdefinierter Regeln (Englisch policies) basierend auf den Informationen, die sich aus der Heat Map ergeben.

Wichtig zu wissen ist, dass in Release 1 noch einige Einschränkungen existieren, die unbedingt bei der Nutzung berücksichtigt werden müssen. Die ILM-Funktionen stehen beispielsweise im Moment nur in einer Nicht-Multitenant-Architektur (auch NON CDB) zur Verfügung.

Bevor über eine Automatisierung der optimalen Datenablage nachgedacht werden kann, müssen die Daten kategorisiert werden. Im Klartext bedeutet dies, dass die Daten je nach Zugriffsstatistik in unterschiedliche Kategorien eingeteilt werden – natürlich automatisch ohne Interaktion durch den Nutzer oder DBA. In Oracle Database 12c ist diese Funktion über das sogenannte **Heat Map Feature** implementiert. Heat Maps sollen einen Überblick über die Aktivitäten auf den unterschiedlichen Objekten geben. Dabei wird nicht nur die Segmentebene berücksichtigt, sondern

es wird sogar die aktuellste Veränderung auf Blockebene mitdokumentiert. Die Aktivitäten bestehen aus Lese- und Schreiboperationen. Sogar Table beziehungsweise Index Lookups werden mitgeschrieben. Damit kein verfälschtes Bild entsteht, werden operative Eingriffe, wie Statistikmanagement, Verlagerungen usw. nicht vermerkt.

Wie funktioniert nun das Heat Map Feature? Eingeschaltet wird die Funktion über einen einzigen dynamischen Initialisierungsparameter HEAT_MAP (Werte ON bzw. OFF). Die Defaulteinstellung ist dabei OFF.

```
ALTER SYSTEM set HEAT_MAP=ON;
```

Nach Aktivierung der Heat Map über den Initialisierungsparameter werden automatisch alle Zugriffe über einen speziellen In-Memory-Zugriff geloggt. Zugriffe auf Objekte im SYSTEM und SYSAUX Tablespace werden dabei ausgelassen.

Der Zugriff auf die Heat Map kann dann über die Standardschnittstellen wie Data Dictionary Views, Fixed Tables oder PL/SQL Packages erfolgen. Eine grafische Schnittstelle in Cloud Control 12c ist geplant. Folgende Beispiele sollen einen kleinen Einblick in die verschiedenen Verwendungsweisen geben.

Einen ersten aktuellen Einblick liefert beispielsweise die View V$HEAT_MAP_SEGMENT. Sie zeigt Realtime-Informationen über die Segmentzugriffe. Das folgende Listing zeigt einen Auszug zur Trackzeit 15:19 Uhr am 11. Juli. Die Objekte, deren Spalten den Wert YES beinhalten, sind aktuell im Zugriff.

```
SELECT h.object_name, o.object_type, h.track_time, h.segment_write
write, h.segment_read read, h.full_scan, h.lookup_scan
FROM v$heat_map_segment h join dba_objects o
ON (o.object_id=h.obj#);

OBJECT_NAME       OBJECT_TYPE     TRACK_TIME       WRI REA FUL LOO
----------------- --------------- ---------------- --- --- --- ---
PRODUCTS          TABLE           11.07.2013 15:19 YES NO  NO  NO
DEPARTMENTS       TABLE           11.07.2013 15:19 NO  NO  NO  NO
CONFIG$           TABLE           11.07.2013 15:19 NO  NO  NO  NO
TAB_300_300       INDEX           11.07.2013 15:19 NO  NO  NO  NO
TAB_300           TABLE           11.07.2013 15:19 NO  NO  YES NO
TAB_300_OLTP      TABLE           11.07.2013 15:19 NO  NO  NO  NO
TAB_300_OLTP      TABLE           11.07.2013 15:19 NO  NO  NO  NO
...
```

Mit ALL_, DBA_, und USER_HEAT_MAP_SEGMENT kann man den letzten Zugriff auf die Segmente anzeigen. Folgendes Codebeispiel listet die Zugriffe auf einige der Objekte des Users SH auf. Wie zu erkennen ist, werden

Lese- und Schreibzugriffe dokumentiert und darüber hinaus die Full Table Scans beziehungsweise die zugehörigen Index Scans aufgelistet.

```
SELECT object_name, segment_write_time write, full_scan, lookup_scan
FROM dba_heat_map_segment
WHERE owner ='SH' AND object_name!='SALES' ORDER BY 2,3;
OBJECT_NAME       WRITE              FULL_SCAN          LOOKUP_SCAN
----------------- ------------------ ------------------ ------------------
TAB_300_OLTP      10.07.2013 17:21   10.07.2013 17:21
TAB_300           10.07.2013 17:21   11.07.2013 15:01
TABLE_300_SELECT  10.07.2013 18:22
PRODUCTS          11.07.2013 15:01   26.06.2013 12:30   25.06.2013 12:48
PRODUCTS_PK       11.07.2013 15:01                      11.07.2013 15:01
CUSTOMERS_PK                                            25.06.2013 22:48
...
```

Aus Gründen der Übersichtlichkeit wurde die Spalte SUBOBJECT_NAME nicht selektiert. Im Fall von Zugriffen auf partitionierte Tabellen, wie zum Beispiel im Fall der Tabelle SALES, ist dies natürlich zur genauen Segmentanalyse unbedingt erforderlich.

Interessiert man sich nur für die Objekte und Tablespaces, die am häufigsten im Zugriff waren, kann man die Views DBA_HEATMAP_TOP_OBJECTS beziehungsweise DBA_HEATMAP_TOP_TABLESPACES verwenden. Im nächsten Beispiel sind die drei top Tablespaces unter Angabe der jeweiligen Objektanzahl und des verwendeten Speicherplatzes aufgelistet.

```
SELECT tablespace_name, segment_count, allocated_bytes, min_
writetime, max_writetime
FROM dba_heatmap_top_tablespaces;
TABLESPACE_NAME SEGMENT_COUNT ALLOCATED_BYTES MIN_WRITE MAX_WRITE
--------------- ------------- --------------- --------- ---------
ADOTEST                    16       135266304
EXAMPLE                   334       190775296 18-JUN-13 05-JUL-13
USERS                      20      5235408896 21-JUN-13 10-JUL-13
```

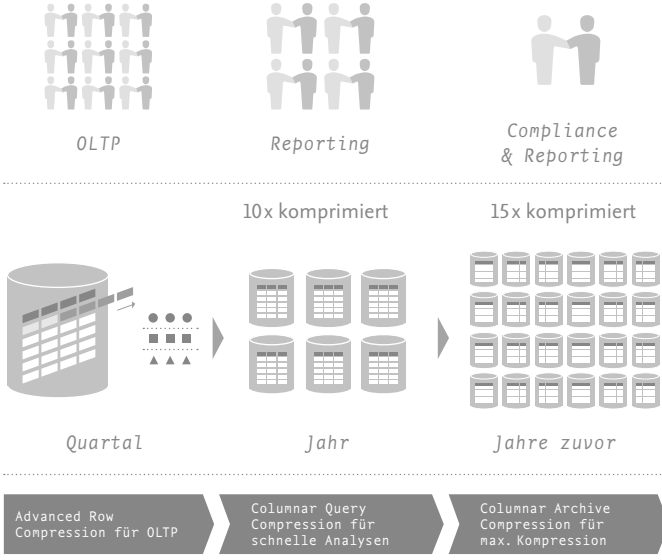
So wird beispielsweise der erste beziehungsweise letzte Schreibzugriff auf Objekte im entsprechenden Tablespace mitgeschrieben.

6.1 AUTOMATISCHE DATENOPTIMIERUNG

Um zu erklären, was mit automatischer Datenoptimierung gemeint ist, soll folgendes Beispiel einen möglichen Anwendungsfall skizzieren. Zuerst werden Daten mit Bulk-Load-Operationen oder anderen konventionellen Methoden in die Tabellen geladen und unterliegen dabei starken Veränderungen. Zu diesem Zeitpunkt ist eine Komprimierung noch nicht erwünscht, somit werde die Daten unkomprimiert gespeichert. Nach einer gewissen Zeit erfolgen nur noch vereinzelte Veränderungen über OLTP-Transaktionen. Nun kann es sinnvoll sein, die Daten aus Platzgründen in das OLTP-Komprimierungsformat zu konvertieren. Nach

einiger Zeit werden die Daten nur noch selten genutzt; nun könnten sie auf ein anderes Speichermedium ausgelagert werden – vielleicht sogar auf ein Speichermedium wie ZFS oder Pillar, die eine höhere Komprimierungsrate durch HCC-Komprimierung erlauben oder einfach nur auf ein beliebiges anderes Low Cost Storage-Medium.

Abb.7: Beispiel für die automatische Datenoptimierung



Das oben erläuterte Szenario lässt sich ab Oracle Database 12c einfach in der Datenbank abbilden und kann sogar in automatisierter Form ablaufen. Wichtiges Hilfsmittel zur Implementierung von automatischer Datenoptimierung sind die sogenannten **Policies**. Eine Policy-Spezifikation beinhaltet dabei eine Aktion wie zum Beispiel **Compression** oder **Storage Tiering** und zusätzlich eine Bedingung, unter der die Aktion ausgeführt werden soll. Policies können auf Segment- oder Row-Ebene implementiert werden. Folgendes Beispiel zeigt eine Segment Level Policy (Schlüsselwort: `SEGMENT`), die automatisch eine Tabelle in das OLTP-Komprimierungsformat umwandelt, nachdem 20 Tage keine Veränderungen (Schlüsselwort: `NO MODIFICATION`) erfolgt sind.

```
ALTER TABLE sh.sales
ILM ADD POLICY ROW STORE COMPRESS ADVANCED SEGMENT
AFTER 20 DAY OF NO MODIFICATION;
```

`ROW STORE COMPRESS ADVANCED` ist dabei übrigens die neu eingeführte Syntax für OLTP Compression in 12c.

Das nächste Beispiel zeigt eine ROW Policy. Oracle evaluiert in regelmäßigen Abständen die Blöcke der ORDERS Tabelle. Jeder Block, der den Anforderungen – in unserem Fall `1 DAY OF NO MODIFICATION` – genügt, wird komprimiert, um wieder ausreichend Platz zu schaffen.

```
ALTER TABLE sh.customers
ILM ADD POLICY ROW STORE COMPRESS ADVANCED ROW
AFTER 1 DAY OF NO MODIFICATION;
```

Diese Policy ermöglicht höchst mögliche Performance beim Laden von Daten und zusätzlich die Vorteile der Platzersparnis durch die Komprimierung. Im Gegensatz zum Segment-Policy-Beispiel muss nicht darauf gewartet werden, bis eine ganze Partition den Anforderungen genügt.

Nun stellt sich die Frage, wann diese Policies evaluiert werden? Da die Aktionen automatisch im Hintergrund ablaufen sollen, bietet sich das Maintenance Window des Default Maintenance Plans zur Evaluierung und Ausführung an. Möchte der DBA eingreifen, ist dies natürlich auch über die entsprechenden Kommandos möglich.

Zusätzlich zu den gerade illustrierten Aktionen über Komprimierung gibt es auch die Möglichkeit der Verlagerung auf einen anderen Storage (auch **Storage Tiering**). Die Implementierung ist einfach, wie in folgendem Beispiel zu sehen ist.

```
ALTER TABLE sh.sales ILM ADD POLICY TIER TO adotest;
```

Diese Art von Policy wird durch den Füllgrad des entsprechenden Tablespaces getriggert. Ist der Quell-Tablespace

(hier USERS) annähernd voll, werden die Daten in den Tablespace ADOTEST verlagert. Die Eigenschaft „voll“ kann dabei durch den DBA beeinflusst werden. In unserem Beispiel ist der Tablespace bei 70 Prozent voll, wie die eingestellten Parameter zeigen.

```
SELECT SUBSTR(name,1,32) name, value
FROM dba_ilmparameters WHERE name LIKE 'TBS%';
NAME                          VALUE
------------------- ----------
TBS PERCENT USED              70
TBS PERCENT FREE              30
```

Policies können natürlich ein- und ausgeschaltet oder auch gelöscht werden. Ein Monitoring ist wie beim Heat Map Feature über die entsprechenden Views wie zum Beispiel DBA_ILMDATAMOVEMENTPOLICIES und DBA_ILMOBJECTS möglich. Folgendes abschließendes Beispiel zeigt einen Ausschnitt aus den im Kapitel benutzten Policies.

```
OBJECT_NAME   SUBOBJECT_NAME  ACTION_TYPE SCOPE   TIER_TABLESPACE
------------ --------------- ----------- ------- ---------------
CONDITION_TYPE         DAYS
---------------------- ----
CUSTOMERS                     COMPRESSION ROW
LAST MODIFICATION TIME    1
```

```
SALES          SALES_1995       STORAGE      SEGMENT ADOTEST
                                0
SALES          SALES_1996       STORAGE      SEGMENT ADOTEST
                                0
SALES          SALES_H1_1997    STORAGE      SEGMENT ADOTEST
                                0
```

Die neuen ILM Features ermöglichen eine **automatische** Datenkomprimierung beziehungsweise eine automatisierte Verlagerung der Daten auf ein anderes Storage-Medium. Wichtige Voraussetzung ist dabei die Nutzung beziehungsweise das Einschalten des **Heat Map** Features auf Systemebene. Die Operationen erfolgen dabei im Hintergrund und können im Fall von Partitionen sogar vollständig **online** ausgeführt werden.

Dieses Kapitel konnte nur einen kleinen Ausschnitt der Funktionalität demonstrieren. So ist es zum Beispiel auch möglich, eigene Funktionen zu definieren, um den Zeitpunkt des Storage Tierings oder der Komprimierung festzulegen. Wer mehr darüber erfahren möchte, dem sei geraten Handbücher, Blogs und White Paper zu konsultieren.

7 Data Pump, External Tables und RMAN

Nicht nur den aktuellen Datenbestand in der Datenbank effizient abzuspeichern muss ein Ziel bei der Speicherverwaltung sein, sondern auch die Backupgröße optimal zu verwalten. Daher bietet Oracle ab 11g eine Komprimierung von Tabellendaten beim Data Pump Export und eine verbesserte RMAN-Komprimierung über die Standardwerkzeuge an.

Der **Oracle Recovery Manager** (kurz RMAN) ist das Werkzeug für Backup und Recovery von Oracle-Datenbanken, das optimale Performance und effizienten Platzverbrauch durch File Multiplexing und zusätzliche Komprimierung ermöglichen kann. Seit Oracle Database 10g ist dabei eine **Backup Compression** mit dem BZIP2-Algorithmus möglich. So können Backup-Sets komprimiert werden, bevor sie auf die Platte geschrieben werden. Bei der Nutzung dieses Backups ist kein zusätzlicher separater Dekomprimierungsschritt mehr notwendig. Folgende Syntax zeigt die Anwendung von Komprimierung beim RMAN Backup.

```
RMAN> BACKUP AS COMPRESSED BACKUPSET DATABASE;
```

Allerdings kann die Anwendung von Komprimierung die Dauer des Backups um ein Vielfaches verlängern. Aus

diesem Grund wurden ab 11g mit der Advanced Compression Option weitere Algorithmen zur Verfügung gestellt, um die Geschwindigkeit beim Backup zu erhöhen. In 11g Release 1 handelte es sich dabei um den ZLIB-Algorithmus. Ab 11g Release 2 änderte Oracle seine Strategie und veröffentlicht nun nicht mehr die Namen der verwendeten Algorithmen. Die Bezeichnungen der Algorithmen lauten `BASIC`, `LOW`, `MEDIUM` und `HIGH`. Einstellbar ist der gewählte Algorithmus über das folgende RMAN-Kommando. Die Defaulteinstellung ist `BASIC`, die der 10g-Variante entspricht.

```
RMAN> CONFIGURE COMPRESSION ALGORITHM 'BASIC|LOW|MEDIUM|HIGH';
```

Auskunft über die zur Verfügung stehenden Algorithmen und den Anwendungsfall gibt die View `V$RMAN_COMPRESSION_ALGORITHM`. `LOW` bietet beispielsweise maximale Geschwindigkeit bei der Komprimierung, `HIGH` hingegen die höchst mögliche Compression Ratio.

```
SELECT algorithm_name name, algorithm_description,
algorithm_compatibility compatibility
FROM v$rman_compression_algorithm;

NAME       ALGORITHM_DESCRIPTION                 COMPATIBILITY
---------- ------------------------------------- -------------
BASIC      good compression ratio                9.2.0.0.0
LOW        maximum possible compression speed    11.2.0.0.0
```

```
MEDIUM   balance between speed and compression ratio   11.0.0.0.0
HIGH     maximum possible compression ratio            11.2.0.0.0
4 rows selected
```

Hinweis: Außer für den Algorithmus `BASIC` muss die Advanced Compression Option lizenziert sein.

Wie viel schneller ein Backupset komprimiert beziehungsweise recovered werden kann, hängt von der Umgebung und dem gewählten Algorithmus ab. Falls ein I/O-Bottleneck vorliegt aber CPU zur Verfügung steht, kann der Algorithmus `HIGH` zu guten Ergebnissen führen. `HIGH` verwendet mehr CPU bei hoher Platzeinsparung und vermindert somit die Anzahl der I/Os zum Schreiben des Backups. Auf der anderen Seite kann bei einer Begrenzung der CPU-Ressource der Einsatz von `LOW` oder `MEDIUM` sinnvoller sein. Komprimierte Daten in der Datenbank werden übrigens bei Einsatz der RMAN-Komprimierung weiter minimiert. Der Einsatz kann also auch in diesem Fall sinnvoll sein. Es zeigt sich, dass mit diesen Algorithmen Platzeinsparungen bis zu 80 Prozent sogar bei höherer Backup-Performance (ca. 60 Prozent) erzielt werden können. Weitere Ergebnisse aus Kunden POCs finden sich im White Paper „Oracle Advanced Compression Helps Global Fortune 500 Company" (siehe Punkt 12: Weitere Informationen). Ein Test des Backup-Szenarios sollte daher vorab immer in Betracht gezogen werden.

Auch der **Data Pump Export** profitiert von Komprimierungsmöglichkeiten. Bislang wurden automatisch nur die Metadaten komprimiert; mit 11g und der Advanced Compression Option können nun auch die Tabellendaten komprimiert werden. Beim Data Pump Export unter Angabe der Option *compression=all* werden automatisch die Tabellendaten komprimiert; beim Import ist keine weitere Angabe beziehungsweise kein weiterer Dekomprimierungsschritt nötig. Dabei bleibt die Komprimierung vollständig applikationstransparent; somit gibt es keine Einschränkungen bei der Data-Pump-Funktionalität. Auch hier sind Komprimierungen von 75 Prozent keine Besonderheit und können mit Werkzeugen wie GNU *zip* verglichen werden. Auch hier finden sich Beispiele im schon erwähnten „White Paper Oracle Advanced Compression Helps Global Fortune 500 Company". Folgendes Listing zeigt die Nutzung mit dem Data Pump Export. Mögliche Einstellungen für den Parameter `COMPRESSION` sind `METADATA_ONLY`, `DATA_ONLY` und `ALL`. Für die Einstellung `ALL` und `DATA_ONLY` ist die Lizenzierung der Advanced Compression Option nötig.

```
expdp compression=all directory=dumpdir dumpfile=back_compl.dmp ...
```

Normalerweise werden die Tabellen mit der gleichen Compression-Einstellung wie beim Export importiert. Neu mit

12c ist die Möglichkeit, diese Einstellung zu ändern. Unabhängig von der Einstellung im Export beziehungsweise von der Einstellung im Tablespace der Zieldatenbank können somit Tabellen mit eigenen Compression-Einstellungen erzeugt werden. Dazu ist ein neuer Metadaten `TRANSFORM`-Parameter nötig. Folgendes Beispiel zeigt eine Implementierung. Der Tablespace der Zieldatenbank besitzt keine Compression-Einstellung. Die Tabelle wird im folgenden Beispiel mit der Eigenschaft Basic Compression importiert.

```
impdp dumpfile=sh.dmp directory=home tables=sh.cust_copy
TRANSFORM = TABLE_COMPRESSION_CLAUSE:\"COMPRESS BASIC\"
```

Die Compression-Klauseln (`TABLE_COMPRESSION_CLAUSE`) entsprechen dabei der Tabellen-Compression-Klauseln im `CREATE` oder `ALTER TABLE`-Kommando.

Mit Oracle Version 10g sind **External Tables** um das Schreiben/Entladen von Daten erweitert worden. Dabei werden die Daten aus der Datenbank in eine binäre Datei geschrieben und können dann einfach und schnell auf einem anderen System zur Verfügung gestellt werden. Wie bei der ursprünglichen Verwendung von External Tables ist auch hier der Zugriff auf ein logisches Datenbank-Directory notwendig. Das Entladen der Daten aus der Datenbank erfolgt dann mit der erweiterten External-Table-Syntax. Die Daten werden durch

das SELECT Statement definiert und in einer binären Datei im logischen Directory abgelegt.

Neu mit 12c ist die Verwendung von COMPRESSION [ENABLED {BASIC|LOW|MEDIUM| HIGH} | DISABLED]. Hiermit wird angegeben, ob und wie die Daten komprimiert werden sollen, bevor sie in der binären Datei abgelegt werden. Folgendes Beispiel zeigt eine Verwendung.

```
CREATE TABLE sh.ext_sales_products_basic
  ORGANIZATION EXTERNAL
   (
     TYPE oracle_datapump
     DEFAULT DIRECTORY home
     ACCESS PARAMETERS (compression enabled basic)
     LOCATION ('sales_prod.exp_low'))
   REJECT LIMIT UNLIMITED
AS SELECT p.prod_name, s.quantity_sold, s.cust_id FROM sh.sales s
JOIN sh.products p USING (prod_id);
```

Wie aus der Syntax zu ersehen ist, können zusätzliche Attribute wie BASIC, LOW, MEDIUM und HIGH bei der Spezifizierung der Komprimierung mitgegeben werden.

BASIC bietet dabei eine gute Kombination aus Ratio und Geschwindigkeit. LOW ist geeignet bei Umgebungen mit

eingeschränkten CPU-Ressourcen. Die Compression Ratio wächst normalerweise von `LOW` nach `HIGH` – in Abhängigkeit von den zur Verfügung stehenden CPU-Ressourcen.

Hinweis: Dieses Feature benötigt die Lizenzierung der Advanced Compression Option.

8 Netzwerkkomprimierung

Netzwerkperformance ist häufig begrenzt durch die Bandbreite und das Datenvolumen. Erhöhen der Netzwerkperformance bedeutet also entweder Vergrößerung der Bandbreite oder Minimieren des Datenvolumens. Komprimierung der Daten, die über das Netzwerk transferiert werden, ist somit eine wichtige Option, um Performance zu erhöhen.

Oracle bietet diese Möglichkeit im Data-Guard-Umfeld ab 11g und in 12c sogar generell über die entsprechenden Netzwerkeinstellungen.

Im ersten Fall können die Redo-Daten, die über den **Data Guard Redo Transport** Service übermittelt werden, komprimiert werden. Redo-Komprimierung kann zu geringeren Redo-Transfer-Zeiten und somit schnelleren Redo-Gap-Resolution-Zeiten und weniger Netzwerkbelastung führen.

Vorteile bringt Redo Compression sicherlich bei niedriger Netzwerkbandbreite und hohen Erwartungen an die Recoveryzeiten. Besonders deutlich wird dies bei Datenbanken mit hoher Redo-Rate und eher niedriger Netzwerkbandbreite. Dabei sollte man natürlich nicht vergessen – wie auch übrigens bei der Komprimierung von unstrukturierten Daten –, dass ausreichend CPU-Ressourcen für die Durchführung der Komprimierung zur Verfügung stehen.

Wie wird die Redo-Komprimierung nun aktiviert? Ab 11g Release 2 wird die Einstellung entweder direkt bei der Angabe des Parameters LOG_ARCHIVE_DEST_n mitgegeben oder über den Data Guard Broker.

```
LOG_ARCHIVE_DEST_n='SERVICE=o1 COMPRESSION=ENABLE'
```

Mit Data Guard Broker kann folgende Syntax verwendet werden.

```
DGMGRL> edit database 'o1 SET PROPERTY 'RedoCompression'= ENABLE;
```

Tipp aus der Praxis: Gibt es Möglichkeiten die Einsparungen beziehungsweise die Compression Ratio zu bestimmen? Der Compression Advisor liefert hier (noch) keine Implementierung. Da der hier verwendete Algorithmus allerdings dem *gzip* mit Level 1 entspricht, kann ein einfacher Test mit einer archivierten Log-Datei die Frage beantworten.

Neu in 12c ist die Möglichkeit, das SQL*Net-Datenvolumen zwischen Client und Server zu komprimieren. Die Konfiguration erfolgt dabei über die Einstellung der neuen SQL*Net-Parameter `SQLNET.COMPRESSION`, `SQLNET.COMPRESSION_LEVELS` und `SQLNET.COMPRESSION_THRESHOLD`. Die Verwendung kann auf verschiedenen Ebenen wie Connection (z. B. connect string), Service (tnsnames.ora, ldap.ora) oder Datenbank (sqlnet.ora) statt finden. Auch hier ist die Voraussetzung die Lizenzierung der Advanced Compression Option.

Folgende Beispiele zeigen die Verwendung.

`SQLNET.COMPRESSION` schaltet die automatische Komprimierung ein. Der Default ist OFF.

```
SQLNET.COMPRESSION=on
```

`SQLNET.COMPRESSION_LEVELS` bestimmt die Ebene der Komprimierung. Der Default ist LOW.

```
SQLNET.COMPRESSION_LEVELS=(high)
```

`SQLNET.COMPRESSION_THRESHOLD` bestimmt ab welcher Größenordnung (in Bytes) die Komprimierung durchgeführt wird. Der Wert gibt die minimale Größe an. Der Default ist 1024 Bytes.

```
SQLNET.COMPRESSION_THRESHOLD=1024
```

Unabhängig von der Einstellung der Komprimierung kann natürlich weiterhin der SDU-Wert als Einstellung verwendet werden. Zur Erinnerung SDU (kurz für session data unit) ist die Größe der Datenpakete, die über das Netzwerk versendet werden. Mögliche Werte rangieren zwischen 512 und 2097152 Bytes.

9 Optimierung für Flashback-Data-Archive-Tabellen

Daten werden gespeichert und zum Teil lange aufbewahrt. Mitunter werden die Daten nach ihrer ersten Speicherung geändert, vielleicht sogar mehrfach. Je nach gesetzlicher oder betrieblicher Vorgabe müssen die Veränderungen nachverfolgbar sein. Damit sind zugleich Mechanismen gefordert, die sicherstellen, dass die Folge der Versionen lückenlos ist. Und implizit bedeutet das zusätzlich, dass die Versionen auch vor Löschen und Verändern geschützt sein müssen. Flashback Data Archives lösen diese Frage, denn sie bieten nicht nur einen wirksamen Mechanismus zum Versionieren von Datensätzen, sondern sie schützen diese Versionen auch vor Veränderung und löschen sie schließlich sogar automatisch nach Ablauf ihrer Aufbewahrungsfrist.

Ursprünglich sind die Archive mit **Oracle Database 11g Release 1** auch unter dem Namen **Total Recall** eingeführt wurden – was übersetzt so viel heißt wie: das perfekte Gedächtnis. Diese Option ist allerdings seit Ende Juni 2012 nicht mehr im Lizenzumfang enthalten (siehe Hinweis am Ende dieses Kapitels).

Wie funktioniert nun Flashback Data Archive? Automatisch werden Änderungen an Tabellen auf Zeilenebene „getrackt" und eine Historie auf Zeilenebene – den sogenannten Archiven oder Flashback Data Archives (auch FDA) – zur Verfügung gestellt. Ein wesentlicher Unterschied zwischen den Möglichkeiten der bekannten UNDO-Mechanismen und denen der Archive ist folgender: Der normale UNDO-Mechanismus der Datenbank sammelt für alle Veränderungen in der Datenbank UNDO-Informationen und ist in produktiven Umgebungen in der Regel auf einige Stunden oder maximal Tage begrenzt. Mit FDA und der Verbindung zu einzelnen Tabellen werden hingegen viel effizientere Möglichkeiten geboten, auf einen größeren Zeitraum von selektiven Daten zu zugreifen.

Dabei kann das Volumen dieser Flashback Data Archives natürlich mit der Zeit sehr groß werden. Um die Speicherung zu optimieren, lohnt es sich daher eine Optimierung beim `CREATE FLASHBACK ARCHIVE` beziehungsweise `ALTER`

FLASHBACK anzugeben. Diese ist nicht als Defaulteinstellung wirksam und muss zusätzlich mit der Advanced Compression Option lizenziert werden. Folgendes Beispiel zeigt wie die Optimierung eingeschaltet werden kann.

```
CREATE FLASHBACK ARCHIVE test_archive
TABLESPACE users
QUOTA 300 M
OPTIMIZE DATA
RETENTION 1 DAY;
```

Hinweis: Ohne den Einsatz dieser Optimierung das heißt ohne die Klausel OPTIMIZE DATA (Default) beziehungsweise bei Verwendung von NO OPTIMIZE DATA steht Flashback Data Archive ab 11.2.0.4 in jeder Edition zur Verfügung.

10 Lizenzierung

Für einige der im Dojo erwähnten Features ist die Lizenzierung der Enterprise Edition (Stand September 2013) nötig. Dabei handelt es sich beispielsweise um folgende Funktionen:

- Online Index Rebuild
- Online Table Redefinition
- Online Datafile Move
- Basic Table Compression
- Bitmapped Index, bitmapped Join Index und Bitmap Plan Konvertierungen

Zusätzlich dazu ist für die Nutzung der folgenden Features die Lizenzierung der Advanced Compression Option (Stand September 2013) nötig:

- Advanced Row Compression (auch OLTP Compression)
- Advanced LOB Compression
- Advanced LOB Deduplication
- RMAN Backup Compression
  (`RMAN DEFAULT COMPRESS` benötigt keine Lizenz)
- Data Pump Data Compression
  (`COMPRESSION=METADATA_ONLY` benötigt keine Lizenz)

- Heat Map
- Automatic Data Optimization
- Data Guard Redo Transport Compression
- Advanced Network Compression
- Optimierung für Flashback Data Archive History Tables
- Storage Snapshot Optimization
- Online Move Partition (für alle komprimierten Formate)

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

11 Fazit und Ausblick

Komprimierung ist in der Datenbank ein nicht mehr wegzudenkendes Feature, das in jedem Release weiterentwickelt wird. Um herauszufinden, ob Komprimierung in der eigenen Umgebung sinnvoll ist, sollte bei strukturierten Daten im ersten Schritt unbedingt der Compression Advisor genutzt werden. Nicht vergessen sollte man dabei auch die unstrukturierten Daten, da hier sehr viel Speicherplatz verwendet werden kann. Ab 12c bietet der Compression Advisor zusätzlich ein Interface für Large Objects. Tests sollten allerdings immer eingeplant werden, da es zu Veränderungen an Ausführungsplänen, Query und DML Performance kommen könnte.

12 Weitere Informationen

- My Oracle Support Notes:
  - Script to investigate a b-tree index structure (DOC ID 989186.1)
  - Master Note for OLTP Compression (Doc ID 1223705.1)
  - Using the new UTL_COMPRESS Oracle Supplied Package (Doc ID 249974.1)

- OTN Download für Oracle Advanced Compression Advisor: *http://www.oracle.com/technetwork/database/options/compression/compression-advisor-095705.html*

- Deutschsprachige Tipps der DBA Community: *http://blogs.oracle.com/dbacommunity_deutsch*

- Oracle White Paper:
  - Oracle Advanced Compression Helps Global Fortune 500 Company
  - Oracle Advanced Compression with Oracle Database 12c

- Handbücher:
  - VLDB and Partitioning Guide
  - SQL Language Reference
  - PL/SQL Packages and Types Reference
  - Database Licensing Information

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

ORACLE®

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