Die Vielfalt von Bitmap-Indizes im DWH

Dominik Schuster
areto consulting gmbh
Köln

Schlüsselworte
Bitmap Indizes, Data Warehouse, Star-Schema

Einleitung

Bei großen und stetig wachsenden Datenmengen steigt auch der Bedarf an performanten und stabilen Methoden, Abfrageergebnisse zeitnahe erzielen zu können. Besonders Bitmap Indizes gelten in diesem Zusammenhang als wirkungsvolles Mittel, den gewaltigen Datenmengen Herr zu werden und finden in heutigen Data-Warehouse-Systemen (DWH-Systemen) bereits breite Anwendungsgebiete. Im Folgenden werden verschiedene Einsatzgebiete von Bitmap Indizes vorgestellt. Außerdem soll ein allgemeines Verständnis für die interne Speicherung und Verarbeitung dieser geschaffen werden. Zuletzt sollen neue Features aus der Oracle Datenbank 12c geschildert werden.

Grundkonzept der Bitmap Indizes

Bitmap Indizes erfüllen wie andere Indizes eine Zeiger-Funktion, bei der es darum geht einzelne Tupel oder ganze Bereiche an Tupel schneller ausfindig zu machen, als es über einen umfassenden Tabellenscan (full table scan, FTS) möglich wäre. Bei einem Bitmap-Index wird eine Bitmap, also eine Reihe von Einsen und Nullen, zu jeder Ausprägung eines zu indizierenden Attributs erstellt. Jede 1 in einer solchen Bitmap steht für die Existenz der Ausprägung an einer bestimmten Stelle eines Datenblocks und jede 0 für die nicht Existenz an dieser Stelle. Diese Speichermethode hat den großen Vorteil, dass mehrere dieser Bitmaps über logische Verknüpfungen kombiniert werden können und so die Ergebnismenge effizient eingeschränkt werden kann. So können bspw. Abbildung 1 zeigt exemplarisch eine mögliche AND-Verknüpfung der Bitmaps zweier Attribute einer Tabelle.

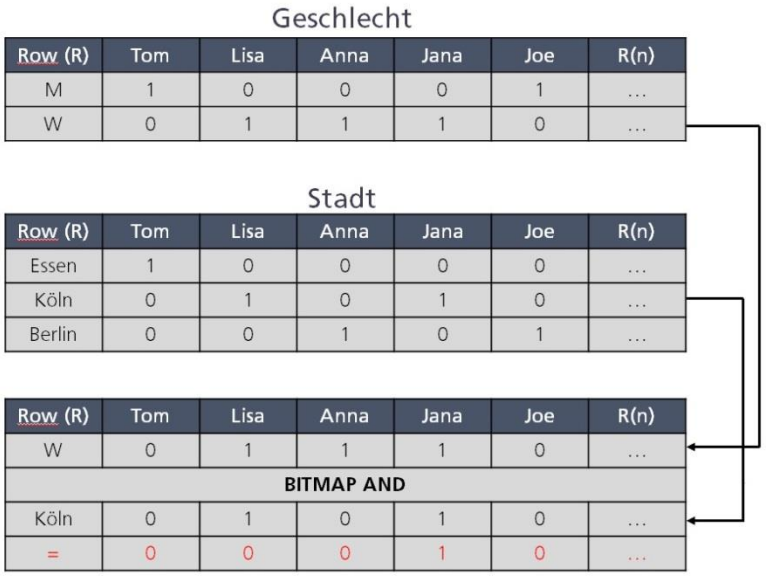

Abbildung 1 - BITMAP AND-Verknüpfung

Wie in Abbildung 1 dargestellt ist das Ergebnis einer solchen logischen Verknüpfung eine Ergebnis-Bitmap, aus welcher anschließend auf die betroffenen Tupel der Tabelle geschlossen werden kann. Bei der Auswahl einer geeigneten Zugriffsstrategie entscheidet der Optimizer kostenbasiert (cost based optimizer, CBO), ob und in welcher Kombination Bitmap Indizes bei der Abfrage verwendet werden.

Bitmap Kompression

Da es sich weiterhin bei Bitmaps um Ketten von Einsen und Nullen handelt, können diese auch sehr Performant komprimiert werden. Die Komprimierung findet automatisch statt, indem Folgen von Nullen zusammengefasst werden. Lediglich die Einsen werden nicht komprimiert abgespeichert. Da sich einzelne Bitmaps immer auf die Vorkommnis einer bestimmten Ausprägung beziehen und diese mit steigender Selektivität immer seltener in der Tabelle vorkommt, kommen gleichzeitig wesentlich mehr Nullen als Einsen, in der entsprechenden Bitmap vor. Daraus ergibt sich die besondere Stärke dieses Komprimierungsverfahrens. Anhand von Abbildung 2 kann die Komprimierung einer Bitfolge angedeutet werden.

Bitfolge	Dezimal	Hexadezimal
00000000 00000000 00000000 00000010	30 x Null	x1E

Abbildung 2 - BITMAP-Komprimierung

Interne Struktur von Bitmap Indizes

Intern werden Bitmap Indizes in einer B-Tree ähnlichen Baumstruktur gespeichert, nur das einzelne Einträge auf der untersten Ebene gleich mehrere Werte umfassen. Abbildung 3 dient der Veranschaulichung einer Bitmap-Baumstruktur.

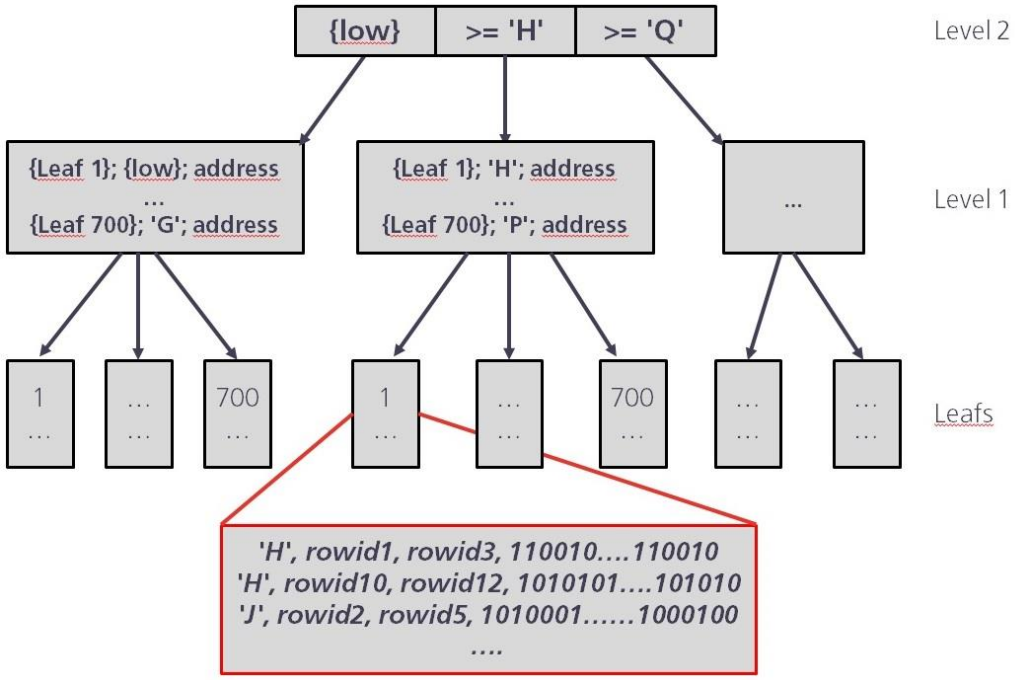

Abbildung 3 - Bitmap-Baumstruktur

Jedes Objekt der Baumstruktur wird in einem separaten Block gespeichert. Wie in Abbildung 3 dargestellt enthält jeder „Branch“-Block aus einem höheren Level Informationen über unmittelbar darunter liegende Branch-, bzw. Leaf-Blöcke. Wenn eine bestimmte Attributsausprägung wie bspw. ‚J‘ gesucht wird, muss die Baumstruktur lediglich vom höchsten Level bis zu den Leafs durchlaufen werden. Weiterhin gibt das Speichern in einer Baumstruktur alle anderen Verfahrens-Vorteile, wie bei dem lesen ganzer Index-Bereiche. Wie schon erwähnt enthalten Leafs bei Bitmap Indizes gleich mehrere Einträge, die sog. Bitmap-Stücke (Pieces), wobei jedes Piece wiederum Wertebereiche repräsentieren. Im Kontrast zum B-Tree-Leaf, bei dem auch mehrere Index-Einträge enthalten sind, aber diese jeweils nur eine Row repräsentieren, sind Bitmap-Leafs also extrem kompakt und komprimiert. Bitmap-Pieces enthalten den eigentlichen Wert, eine start-, und end-rowid und eine Bitmap, welche die Vorkommnis des Werts zwischen start- und end-rowid repräsentiert. Eine Bitmap wird aufgeteilt, wenn diese eine Größe von ca. einem halben Block erreicht und um zu kodierende Nullen auszusparen. Ein Leaf-Block kann weiterhin mehrere Pieces von unterschiedlichen Ausprägungen enthalten.

Kardinalität und Verteilung

Weiterhin ist die Kardinalität einer Spalte essentiell für die Entscheidung einen Bitmap Index einzusetzen. Eine Empfehlung von Oracle lautet, dass der Bitmap Index besonders Wirkungsvoll ist, wenn 1%, oder weniger der Ausprägungen unterschiedlich sind. Anders ausgedrückt, soll sich jede Ausprägung mindestens 100-mal wiederholen, wenn die Tabelle 10000 Rows hat. Dies sollte aber nicht zum vorzeitigen Ausschluss des Bitmap Index führen, da mit steigender Menge an unterschiedlichen Ausprägungen auch die Komprimierungsrate der Bitmaps steigen kann.
Eine weitere entscheidende Rolle spielt die Verteilung der einzelnen Ausprägungen in der Tabelle. Für die Messung der Verteilung der Werte eines Bitmap Index kann jedoch nicht der Clustering-Faktor in betracht gezogen werden, da dieser bei Bitmap Indizes immer gleich der Anzahl an Bitmap-Pieces in der Bitmap-Baumstruktur ist. Der Clustering-Faktor einer Spalte gibt im Groben an, mit welcher Häufigkeit im Index-Segment nebeneinander liegende Werte auch im Tabellen-Segment nebeneinander zu finden sind. Also wie stark die Tabelle nach dem Index geordnet ist. Besonders im Bereich der niedrigen Kardinalität braucht ein Bitmap Index auf einer schlecht verteilten Spalte bedeutend mehr Leaf-Blöcke um die gleiche Menge an Werten zu repräsentieren als es bei einer gut verteilten Spalte der Fall wäre. Dieser Sachverhalt kann mit Abbildung 4 verdeutlicht werden.

INDEX_NAME	BLEVEL	LEAF_BLOCKS	DISTINCT_KEYS	NUM_ROWS	CLUSTERING_FACTOR
T1_I1_SCATTERED_BMI	1	110	20	240	240
T1_I2_CLUSTERED_BMI	1	20	20	40	40
T1_I3_SCATTERED_BMI	1	125	25	250	250
T1_I4_CLUSTERED_BMI	1	17	25	50	50

Abbildung 4 - scattered vs clustered Data

Für das Beispiel in Abbildung 4 wurde eine Tabelle mit 20000 Rows erstellt. Um die unterschiedliche Verteilung zu simulieren wurden die scattered Spalten mithilfe der Mod-Funktion und die clustered Spalten mithilfe einer einfachen Teilung erstellt. Es kann gezeigt werden, dass die Bitmap Indizes auf den gut verteilten Spalten mit wesentlich weniger Leaf-Blocks auskommen.

Bitmap Conversion
Damit die Arbeitsweise von Bitmap Indizes besser verdeutlicht werden kann, sollte auf die Bitmap Conversion from/to rowid eingegangen werden. Dieser Vorgang wird bei Umwandlungen von Bitmaps zu den eigentlichen Rows in der Tabelle und umgekehrt verwendet. Um diese Umwandlung durchführen zu können braucht der Optimizer eine Reihe von Informationen. Dazu zählen die Start- und End-rowid der Bitmap, die eigentliche Bitmap und die Anzahl an Rows per Block. Da sich diese Anzahl jedoch von Block zu Block und Tabelle zu Tabelle unterscheiden kann, wird zu jeder Tabelle der sog. Håkan faktor gespeichert. Der Håkan factor einer Tabelle gibt an, wie viele Rows maximal in einen Block passen. Dafür wird im default das Minimum an bytes pro Spalte genommen, zusammen addiert und die (Größe des Blocks – Block-Header) durch diese geteilt. Wenn ein Block jetzt weniger Rows aufweist als im Håkan factor vorgesehen war, werden die übrigen Stellen einfach mit Nullen aufgefüllt, welche gut komprimiert werden können. Das Problem an diesem Verfahren kann sein, dass bspw. durch diverse Varchar2-Spalten eine maximale Row per Block Anzahl errechnet wird, welche jedoch weit neben der realistischen maximalen Anzahl an Rows per Block liegt. Durch die vielen zusätzlichen Nullen, die fälschlicher Weise angehangen werden wird der Index unnötig vergrößert. Um den Håkan factor nachträglich anzupassen und die tatsächliche maximale Anzahl an Rows per Block errechnen zu können, kann der Befehl alter table … minimize records_per_block verwendet werden. Dabei führt die Datenbank einen FTS durch und überprüft, wie viele Rows tatsächlich maximal in einem Block stecken. Der Håkan factor wird anschließend auf diesen Wert angepasst, wodurch der Bitmap Index stark verkleinert werden kann. Nachdem der Håkan factor angepasst wurde kann es jedoch sein, dass dieser bei einem partition-exchange-load jeweils bei der neuen Partition und bei dem Ziel des exchange-loads unterschiedlich ist. Dies führt zu einen Datenbank Fehler, welcher dadurch umgangen werden kann, indem die Indizes währenddessen deaktiviert, oder gelöscht und neu erstellt werden.

Star Transformation
Im Analysebereich des Data Warehouse wird die gewünschte Abfragemenge bevorzugt über Einschränkungen innerhalb von Dimensionen erlangt. Da jedoch nur ein Join an einem Zeitpunkt getätigt werden kann, muss der Join vorzeitig auf die Faktentabelle geschehen, obwohl eine zusätzliche Einschränkung auf einer weiteren Dimension günstiger wäre. Bitmap Indizes haben den Vorteil, dass sie eben diese mehrfachen Einschränkungen realisieren können, ohne dafür Tabellen zu Joinen. Da diese aber die Fremdschlüsselattribute der Faktentabelle erstellt werden, müssen eben diese gesuchten Schlüssel-Ausprägungen im Vorhinein errechnet werden. Eine Abfrage kann zu diesem Zweck so formuliert werden, dass das Ergebnis einer Einschränkung auf einer Dimensionstabelle per Sub-Select errechnet wird und der Join umgangen werden kann. Mit Hilfe der Star Transformation kann der Optimizer eine Abfrage automatisch in die für die Indizes benötigte Form umwandeln.

Typische Abfrage:
```
select p.name, sum(s.menge * s.wert) Umsatz
from verkauf v, produkt p
where v.produkt_id = p.id
and p.typ = ‘verderblich’;
```

Transformiert:
```
select p.name, sum(s.menge * s.wert) Umsatz
from verkauf v, produkt p
where v.produkt_id IN (
        select *
        from produkt
        where typ = ‘verderblich’);
```

Abbildung 5 dient dem Verständnis dieses Vorgangs.

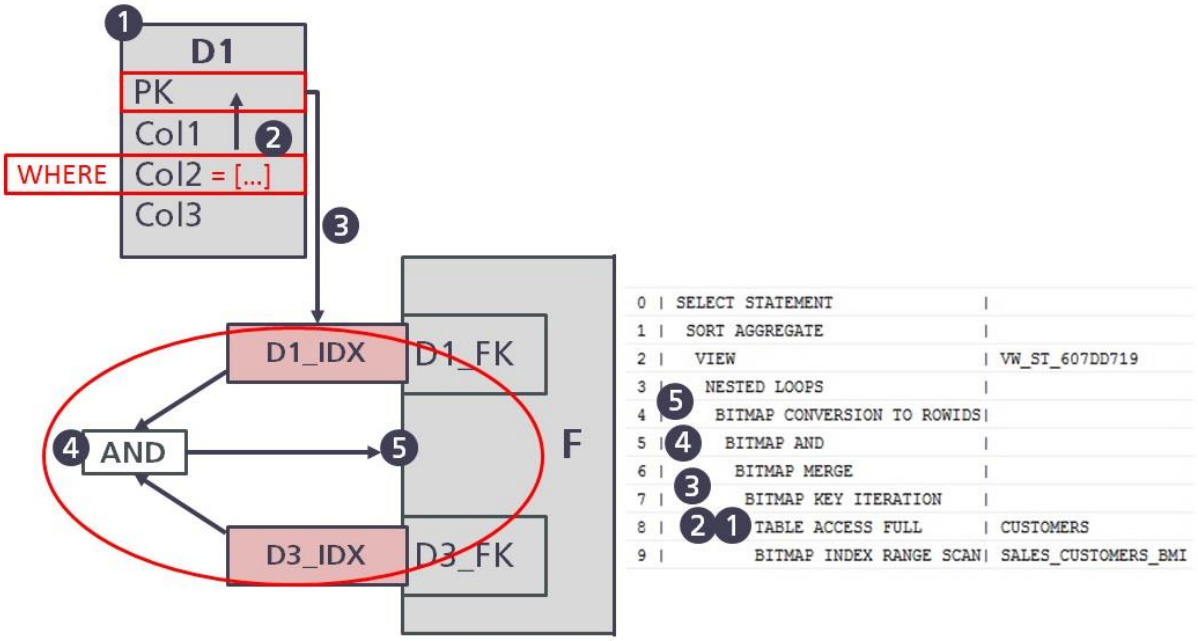

Abbildung 5 - Star Transformation

Durch die Transformation der Query ist es dem CBO möglich, die Abfragemenge mithilfe der Bitmap Indizes effektiv einzuschränken. Um die automatische Transformation der Query zu aktivieren, muss der Parameter „star_transformation_enabled“ explizit auf „TRUE“ bzw. auf „DISABLE_TEMP“ gesetzt werden. DISABLE_TEMP führt ähnlich wie TRUE die Starquery Transformation aus, nur das das Zwischenspeichern in temporären Tabellen deaktiviert ist. Nach der erfolgreichen Transformation der Query kann es nötig sein, mit der Ergebnismenge der Faktentabelle zurück auf die Dimension zu Joinen, wenn zusätzliche Informationen aus diesen ausgegeben werden sollen, oder nach einem weiteren Attribut der Dimension gruppiert werden soll. Aus diesem Grund kann es sinnvoll sein, die vorherige Ergebnismenge des Sub-Selects in einer temporären Tabelle zwischen zu speichern, welche dann ausgelesen werden kann. Anhand der Query erkennt der CBO ob eine temporäre Speicherung der Ergebnismenge sinnvoll sein könnte und kann diese bei der Option TRUE anwenden.

Bitmap Join Index

Bei einem Bitmap Join Index werden die Bitmaps zu Ausprägungen aus Dimensionstabellen erstellt und repräsentieren somit die Rows einer Tabelle (bspw. Faktentabelle), die als Ergebnis einer Einschränkung auf das indizierte Attribut der Dimension ausgegeben werden. Dies wird anhand von Abbildung 6 verdeutlicht.

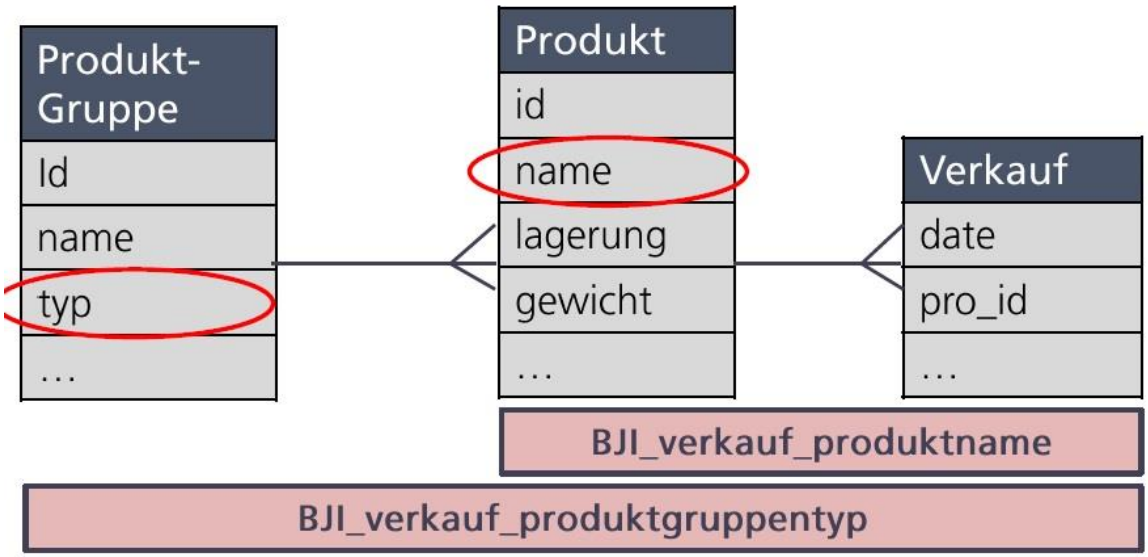

Abbildung 6 - Bitmap Join Index

Wie in Abbildung 6 angedeutet, können Bitmap Join Indizes auch über Snowflake-Schemata angelegt werden und somit einen zweifachen Join abbilden. Zur Erstellung eines BJI wird als zu indizierende Tabelle die Faktentabelle angegeben, aber mit dem Zielattribut der Dimensionstabelle. Weiterhin muss der gesamte Join aufgeführt werden.
BJI sind bei Filterungen auf die indizierten Dimensionsattribute performant, da durch diese Joins, sowie Transformationen der Query durch die Star Transformation vermieden werden können. BJI sind jedoch unflexibel. Bei Filterung auf andere Attribute können diese überhaupt nicht verwendet werden. Dadurch eignen sie sich eher als optionales Feature, kommen jedoch nicht an den Nutzen von normalen BMI heran.

Attribute Clustering (12c)
Ein neues Feature, welches Oracle im letzten Datenbank Release (12.1.0.2) vorgestellt hat, ist das Attribute Clustering. Es dient dazu Daten in einer Tabelle physisch nach einer oder mehreren Spalten zu sortieren. Dabei können maximal 10 Attribute für die Sortierung verwendet werden, welche sich nicht unbedingt in derselben Tabelle befinden müssen. Es kann zusätzlich über Attribute in umliegenden Dimensionen sortiert werden. Syntaktisch wird nach einem create oder alter Befehl das Argument „clustering by“ angefügt, welches die Tabellensortierung ankündigt.

create table sales (...)
clustering by linear order (attribut1, attribut2, ...)
yes on load yes on data movement
without materialized zonemap;

Der Sortiervorgang der Tabelle (oder auch einzelnen Partition) kann auf den Zeitpunkt eines direct path load, oder auf einen move-Befehl gesetzt werden.
Dadurch wird ein hohes Maß an Flexibilität gewährleistet, da der Zeitpunkt der Sortierung nach Auslastung, sowie nach Zweckmäßigkeit verschoben wird. Beim Attribute Clustering gibt es zwei unterschiedliche Modi*: by linear order*, *by interleaved order*. Linear order ist die standardmäßige ascending order, welche auch bei einem order by regulär durchgeführt wird. Interleaved order speichert die Werte in einer Z-Sort-Ordnung, welche sich besonders dafür eignet, wenn alle, oder nur Teile der Attribute als Abfragekriterium verwendet werden.

Attribut Clustering kann partitionsweise an- und ausgeschaltet und sogar bei seinen geclusterten Attributen angepasst werden. Dies kann u.a. von Vorteil sein, wenn sich bevorzugte Filter mit dem Alter der Daten ändern. Wie vorher dargestellt, profitieren BMI stark von gut sortierten Daten. Neben dem Fakt, dass Tabellen besser komprimiert werden können, brauchen BMI weniger Leaf-Blocks. So kann Abfrageperformance gewonnen und Zeit bei der Pflege dieser gespart werden.

Fazit

Bitmap Indizes sind nach wie vor geeignetes Mittel, Abfragen zu beschleunigen. Besonders die Fähigkeit diese ad hoc kombinieren und Ergebnismengen stark einschränken zu können, kann Abfragen extrem beschleunigen. Durch ihre Eigenschaft, aus Nullen und Einsen zu bestehen können diese weiterhin extrem effizient gespeichert und abgefragt werden. Mit eingeschalteter Star Transformation kann der Optimizer die Abfrage so umformen, dass Joins verhindert und Vorteile des Bitmap Index bestmöglich eingesetzt werden können. Auch Bitmap Join Indizes können zur Reduzierung von Joins führen, sollten aber durch einen Mangel an Flexibilität eher optional verwendet werden. Außerdem entscheidet der CBO, ob er schneller an Abfrageergebnisse kommen kann, wenn andere Indizes in eine Bitmap on-the-fly umgewandelt werden. Durch die bitmap-to-row Transformation kann aus der Ergebnis-Bitmap wieder auf die einzelnen Zeilen der Relation geschlossen werden.

Zusammenfassend kann gesagt werden, dass die Entscheidung für einen Bitmap Index von verschiedenen Kriterien abhängt. Ausprägungen eines Attributs sollten sich mindestens einmal wiederholen. Weiterhin sollte der clustering-factor, der darüber bestimmt, aus wie vielen Einsen und Nullen eine Bitmap besteht und wie viele Mitmap-Stücke letztendlich zu einem Attribut gehören, möglichst niedrig sein. Dieser kann mithilfe des Oracle 12c-Features Attribut Clustering performant und flexibel reduziert werden. Durch manuelle Anpassung des Håkan faktors kann die Größe alle BMI einer Tabelle reduziert werden, wobei diese bei einem Partition Exchange Load deaktiviert oder gedroppt werden müssen.

Kontaktadresse:
Dominik Schuster
areto consulting gmbh
Schanzenstraße, 6-20
D- 51063 Köln

Telefon: +49 221 66 95 75-0
Telefax: +49 221 66 95 75-99
E-Mail dominik.schuster@areto-consulting.de
Internet: www.areto-consulting.de