Partitionierung für mehr Performance

Sebastian Winkler
CarajanDB GmbH
Erftstadt

Schlüsselworte

Oracle Partitioning, Data Warehouse, VLDB, Archivierung, Zugriffsoptimierung, Range, List, Hash

Einleitung

Mit der Partitionierung bietet Ihnen die Oracle Datenbank ein einfach zu verwendendes Feature um Tabellen und Indexe in einzelne Datensegmente aufzuteilen. Das soll nicht nur deren Verwaltung sondern vor allem Performance und Verfügbarkeit steigern. Dazu unterstützt Oracle Partitioning eine Reihe von verschiedenen Partitionierungsoptionen wie die populäre Range Partition sowie die List, Hash oder Interval Partition und eine lange Liste von Mischformen daraus. Die größte Herausforderung bei der Implementierung heißt aber nicht wie baue ich eine entsprechende Partitionierung in mein Data Warehouse am geschicktesten ein, sondern vielmehr: Welcher Partitionierungstyp ist zunächst mal der Richtige für meine Anwendung und welche Spalte soll der Partition Key sein? Des Weiteren stellt sich die Frage, ob eine Partitionierung überhaupt Sinn macht und die versprochene Performancesteigerung für die eigene Datenbank überhaupt zu erwarten ist. Dieser Vortrag soll einen Überblick über die verschiedenen Möglichkeiten geben, welche die Partitionierung bietet, wie man die beste Strategie zur Partitionierung auswählt. Dazu wird zunächst das Konzept, dass hinter der Partitionierung steht beleuchtet, anhand von konkreten Code-Beispielen rücken die einzelnen Partitionierungstypen in den Fokus. Wie wendet man diese an, für welche Anwendung sind diese geeignet und welchen Nutzen bringen sie. Es folgt eine Einordnung nach Zielen, denn verschiedene Partitionierungsstrategien können miteinander kombiniert werden. Was ist mein wichtigstes Ziel - Performance, Verfügbarkeit, Archivierung und Bereinigung oder Optimierung meines Backup? Schließlich bietet die Partitionierung seit ihrer Einführung mit Version 8 kontinuierlich neue Funktionen, so auch mit 12c. Beispielsweise können Partitionen inzwischen online verschoben werden. In Kombination mit Oracle Advanced Compression erhält man ein Werkzeug zur Einrichtung des vielbeschworenen Information Lifecycle Management. Am Ende steht die Beantwortung der Frage: In welcher Form kann die Partitionierung für uns von Nutzen sein.

Warum Partitionieren?

Das Stichwort heißt große Datenbanken oder sehr große Datenbanken. Was vor 10 bis 20 Jahren als groß galt ist heute zunehmend der Normalfall. In der Datenbanklehre spricht man von Very Large Database kurz VLDB. In der Praxis bertrifft dies meist das sogenannte Data Warehouse. Dabei bedeuten größere Datenbanken vor allem größere Herausforderungen im Betrieb, im Hinblick auf die Menge der Daten, das Transaktionsvolumen, die Aktivität, Parallele Userzugriffe, steigende Abfragekomplexität, Antwortzeiten und nicht zuletzt Verfügbarkeit.

Hierfür sind Standartechniken allein nicht mehr ausreichend. Partitioinierung ist aus diesem Blickpunkt eine der wichtigsten Techniken um diesen Herausforderungen zu begegnen. Partitionierung steht dabei zum einen als Vorteil für die Administration (sehr) großer Datenbanken. Durch die Möglichkeit des Löschens, Hinzufügens, Verschiebens und Archivierens von Partitionen vereinfacht sie die Administration und macht sie beherrschbar. Als weiteren wichtigen Vorteil unterstützt sie den Oracle

Optimizer bei einer effizienteren Abfrage-Bearbeitung durch sogenanntes Partition Pruning, partitionsweise Joins und Parallelisierung.

Partitionierung arbeitet auf Segementebene. Ein einzlnes Segment wird nach einer bestimmten Vorschrift in mehrere Segmente aufgeteilt. Ein einzlnes Segment stellt dabei Tabellen oder Indizes dar, welche nach dem sogenannten Partitionierungskriterium in die entsprechenden Partitionen aufgeteilt wird. Dieses Kriterium wird beim Anlegen der Tabelle festgelegt. Als Partitionierungskriterium können grundsätzlich Wertebereiche, Wertelisten oder Hash-Werte verwandt werden. Neue Daten werden schließlich automatisch einsortiert.

Entscheidungshilfe

Es ergeben sich mehrere Vorteile. Die Partitionen sind physikalisch unabhängig, d.h., es ist eine Ablage und Verwaltung in separaten Tablespaces mit unterschiedlichen Speicherparametern (z.B. PCTFREE, COMPRESSION) möglich. Partitionierung ist Transparent, d.h. aus Anwendungssicht sind alle Daten über den Namen der partitionierten Tabelle les- und schreibbar. Der Optimizer kennt die physikalische Aufteilung der Tabelle und kann entsprechend optimierte Zugriffspfade generieren. Einer (partitionierten) Tabelle zugehörige Indizes lassen sich ebenfalls, abhängig oder unabhängig von dieser, partitionieren. Kleinere Partitionssegmente ermöglichen dabei eine bessere Verwaltbarkeit und effizienteren Indexaufbau.

Zu Optimierung von Zugriffen kann man das bereits erwähnte Partition Pruning nutzen. Dabei wird ausgenutzt, dass eine Partition gegenüber dem logischen Objekt eine kleinere Einheit darstellt. Je nach WHERE-Klausel profitiert jetzt eine Table-Scan-Operation durch das ausklammern bestimmter Partitionen. Weiterhin kann man von flacheren Index-Bäumen profitieren, da sie Partitionsspezifisch aufgebaut auch flacher ausfallen können. Weniger Ebenen reduzieren so den I/O-Aufwand bei Zugriffen. Joins lassen sich auf korrespondierende Partitionen durch partitionsweise Tabellenoperationen reduzieren. Partitionsweise Joins lassen sich zudem parallel auf den physisch getrennten Daten ausführen. Eine Optimierung von Verwaltungsaufgaben bedeutet eine Vereinfachung und Verkürzung der Verwaltung durch Partitionen die einzeln hinzugefügt, aufgeteilt und gelöscht werden können. Ein Datenaustausch zwischen Tabellen und Partitionen ist über effiziente Switch-Operationen möglich – Stichwort: EXCHANGE PARTITION und das ohne Locking.

Alle Vorteile lassen sich prinzipiell verwirklichen. Ihr konkreter Nutzen hängt jedoch vom Einzelfall ab. Voraussetzung ist eine sorgfältige Analyse vor der Implementierung. Dazu sollte zunächst eine priorisierung der Optimierungsziele erfolgen. Steht allein eine effizientere Verwaltung oder eher eine Optimierung der Zugriffe im Vordergrund. Desweiteren sollte die Größe der Objekte in Betracht gezogen werden. Eine Partitionierung einer Tabelle mit wenigen Zeilen ist wenig sinnvoll. Die Art der Zugriffe spielt ebenso eine Rolle. OLTP Anwendungen profitieren durch sehr selektive Indexzugriffe generell weniger von einer Partitionierung. Batch-Anwendungen oder DWH im Allgemeinen mit Scans auf große Bereiche profitieren von dichteren Partitionsbereichen und flacheren Indexstrukturen. Als letzten Punkt sei die Art der Datenverwaltung zu nennen. Lässt sich bspw. das Partitionierungskriterium der Tabelle mit dem Löschzyklus einer Rolling-Window-Anwendung synchronisieren bietet sich auch eine dahingehende Partitionierung an. Partitionierung ist generell zu empfehlen, wenn alle zu einer Priorisierung passenden Kriterien erfüllt werden. Sie ist zu empfehlen bei großen Tabellen im zwei-stelligen Gigabyte-Bereich oder wenn große Datenmengen über viele Jahre hinweg zur Verfügung stehen müssen –Stichwort: In Database Archiving.

Partitionierungsarten

Zunächst gibt es drei Standardtechniken. Partitionierung nach Wertebereichen, nach Wertelisten oder nach Hash-Werten. Als Erweiterungsformen gibt es die Intervall-, Referenz- und die Virtual-Column-

Based-Partitionierung, letztere also eine Partitionierung die sich auf mit 11g eingeführte virtuelle Spalten bezieht. Schließlich gibt es noch die Zusammensetzung verschiedener Partitionierungstypen die Composite- oder Sub-Partitionierung. Hierbei werden zwei Arten hierarchisch miteinander kombiniert. Mit den Standardtechniken lassen sich jeweils alle Partitionierungstechniken miteinander kombinieren, von Range-List bis Hash-Range oder auch List-List. Interval Partitioning lässt sich ebenfalls mit den drei Standardtechniken ausführen. Neu mit 12c ist die Möglichkeit des Interval-Reference-Partitioning.

Die Maximale Anzahl Partionen pro Segment hat sich seit 10gR2 von 1024K-1 also etwas über 1 Million Partitionen sowie Sub-Partionen nicht geändert. Davor, also bis 10gR1 waren es noch 64K-1 also etwas über 65.000 Partionen. Es bestimmt also eher die Händelbarkeit, die genaue Anzahl und Granularität der Partitionen. Einzelne Partitionen sollten nicht über den 2-stelligen GB-Bereich hinauswachsen, dies bestimmt zugleich die Granularität des Partitionierungskriteriums z.B. Woche, Monat, Quartal oder Jahr. Die richtige Partitionierungsspalte sollte nach dem initalen Einfügen des Datensatzes unveränderlich oder mindestens selten veränderlich sein. Seit 11g können wie bereits erwähnt virtuelle Spalten genutzt werden. Beispielsweise ein Teilstring einer Spalte oder funktionale Ausdrücke aus mehreren Spalten. Vorsicht, ein nachträgliches Ändern einer solchen Spalte kann eine andere Partition bedeuten und eine andere Partition bedeutet ein physisches Bewegen der Daten.

Range Partitioning ist die Partitionierung nach Wertebereichen und ist besonders geeignet für zeitbezogene Daten. Zum Beispiel Woche, Monat oder Quartal für eine Auftragsdatumsspalte. Sie ist die wohl häufigste Variante in der Praxis. Seit 11g wird eine Interval-Partitionierung als Spezialfall unterstützt. Hiermit werden neue Partitionen bei Bedarf automatisch erstellt. List Partitioning ist die Partitonierung nach Wertelisten und besonders geeignet für Daten die nach Gebieten bspw. Regionen- oder Bundeslandspalte zugeordnet werden. List Partitioning ist nicht geeignet für sich nach dem initialen Einfügen häufig ändernde Spalten - Paradebeispiel: eine Statusspalte. Hash Partitioning ist die Partitionierung nach Hash-Werten, es kommt also auf den konkreten Wert der Partitionierungsspalte nicht an und es sollen irgendwelche Partitionen geschaffen werden. Hash Partitioning wird häufig für Subpartitionen eingesetzt - Range-Hash, List-Hash. Am Beispiel einer Equi-Partitionierung lässt sich der Nutzen einer Hash-Partitionierung verdeutlichen. Zwei Tabellen werden also nach ihrer Join-Spalte Hash-partitioniert. Der Optimizer profitiert in diesem Falle von partitionsweisen Joins. Partition Pruning lässt sich hier aber nicht nutzen. Reference Partitioning beschreibt die Partitionierung nach einer übergeordneten Tabelle. Es kann nach Spalten partitioniert werden, die nur in einer per Fremdschlüssel übergeordneten Tabelle vorkommen. Beispielsweise lässt sich eine Auftragstabelle nach dem Bundesland des Kunden partitionieren. Sie ist die logische Fortsetzung und Erweiterung der Equi-Partitionierung. Vorteil ist eine erleichterte Administration und die Unterstützung partitonsweiser Joins der betreffenden Tabelle.

Wie soll also partitioniert werden? Diese Frage beantwortet sich durch die Anwort auf eine weitere Frage: Wie wird typischerweise auf die Daten zugegriffen? Die Partitionierungsspalte muss in der WHERE-Klausel vorkommen, nur in diesem Falle kann der Optimizer von Partition Pruning und partitionsweisen Joins profitieren. Unterm Strich werden statt der gesamten Tabelle nur diejenigen Partitionen, im besten Falle nur eine einzige Partition gelesen oder gejoint.

Grundsätzlich kann nur beim Erstellen einer Tabelle entschieden werden ob partitioniert werden soll. Das Einführen einer Partitionierung für eine nicht-partitionierte Tabelle bedeutet mit Ausnahme von Partition Exchange, den kompletten Neuaufbau der Tabelle genauso wie das ändern der Patitionierungsart. Nachträgliches Hinzufügen, Entfernen, Splitten oder Verschmelzen einer Partition einer bereits partitionierten Tabelle ist problemlos möglich. Schwer revidierbare Entscheidungen sollten aber vorher genau analysiert werden.

HINWEIS: Zu den folgenden Partitionierungsarten finden Sie konkrete Code-Beispiele in der Präsentation.

RANGE Partitioning

Große Datenmengen enthalten oftmals ein Datumsfeld, dieses ist für ein Range Partitioning geeignet sofern es überwiegend unveränderlich ist. Im Data Warehouse sind oftmals die Detaildaten (Faktentabelle) mit Datum versehen. Eine Range Partitionierung der Tabellen und zugehörigen Indizes nach dem Datum bietet sich also an. Die Partitionierung wird beim CREATE TABLE mit PARTITION BY RANGE () angegeben. Jede Partition erhält per VALUES-Klausel eine entsprechende Obergrenze. Damit ist bei der Range Partitionierung nachvollziehbar welche Daten in welcher Partition sind bzw. in welche Partition zukünftig laufen. Der Optimizer kann somit bestimmte Partitionen ausklammern und den Zugriff auf einzelne Partitionen beschränken. Für ein optimales Ausnutzen von Partition Pruning sollten die Daten annäherungsweise gleich auf die Partitionen verteilt sein.

Ein weiterer Vorteil, alte Daten können per DDL-Operation gelöscht oder archiviert werden. Besonders geeignet für Rolling-Window-Anwendungen, bei dem sich das Zeitfenster für die Betrachtung verschiebt. Das erste Quartal 2015 wird beispielsweise auf ein Read-Only-Medium ausgelagert und ein neues erstes Quartal 2016 wird angelegt.

Vorsicht bei Nutzung von SPLIT PARTITION, dies bewirkt dass alle Daten einer Partition physisch bewegt werden. Es wird beispielsweise das erste Quartal 2016 angelegt und erst am Ende des Quartals wird das zweite Quartal per SPLIT erzeugt. Quartal eins und zwei werden dabei aber komplett neu eingepflegt und es werden natürlich Undo- und Redolog-Informationen erzeugt. Besser ist es immer eine „Dummy“-Partition zu benutzen und dann auf einer leeren Partition den SPLIT durchzuführen.

Auf eine MAXVALUE Partition kann verzichtet werden, wenn sichergestellt ist, dass keine Daten hierein laufen würden. Ein ADD PARTITION an der oberen Partitionsgrenze kann nur genutzt werden, wenn keine MAXVALUE Partition existiert.

INTERVAL Partitioning

Im Rolling-Window-Verfahren kann seit Version 11g der Oracle Datenbank die Erstellung neuer Range Partitionen automatisiert werden. Dies ist sowohl für neue als auch existierende Range-partitionierte Tabellen möglich. Vorrausgesetzt es wird nach einer einzigen Spalte vom Typ DATE oder NUMBER partitioniert und es existiert keine MAXVALUE Partition als Obergrenze.

Als sogenannte Start-Partition dient die Partition mit der höchsten vorhandenen Partitionsgrenze, sie ist der Ausgangspunkt für automatisch erzeugte Partitionen. Die INTERVAL-Klausel bestimmt die Partitionsbreite. Die optionale STORE IN-Klausel gibt an in welchen Tablespaces Partitionen erstellt werden. Ein solcher Tablespace muss spätestens beim Einfügen der Daten existieren. Per ALTER TABLE SET lassen sich Partitionsbreiten und zu verwendende Tablespaces anpassen. Dies wirkt sich allerdings nur auf zukünftige Partitionen aus.

Über die user_tab_partitions View lassen sich die Partitionen anzeigen. Die Spalte INTERVAL gibt mit YES oder NO Auskunft über manuell oder automatisch erzeugte Partitionen. Neue Partitionen werden mit systemgenerierten Namen angelegt und lassen sich zwar nicht vorgeben aber zumindest nachträglich ändern. Bei einem Rollback bleibt eine einmal automatisch erzeugt Partition bestehen. Bei einem Datensatz jenseits der nächsten anzulegenden Partition, werden nicht alle dazwischen liegenden Partitionen angelegt, sondern nur eine einzige. Bei Bedarf wird diese automatisch, ohne physisches Bewegen der Zeilen und ohne Invalidierung globaler Indizes, gesplittet.

Interval Partitioning automatisiert lediglich das Erstellen neuer Partitionen. Alte Partitionen obliegen weiterhin dem Administrator und können beispielsweise nicht automatisiert gelöscht werden. Ein manuelles Erstellen von Partitionen ist nicht mehr möglich. Deaktivieren der automatischen Partitionserstellung ist mit einem einfachen SET INTERVAL() möglich.

LIST Partitioning

Beim List Partitioning sind die Informationen nach Gebieten oder sonstigen ungeordneten Wertelisten eingeteilt. Diese sollten relativ unveränderlich sein und nach diesem Kriterium wird in Abfragen häufig eingeschränkt. List Partitioning lässt sich gut mit virtuellen Spalten kombinieren. Es können jedoch keine Spaltenkombinationen verwendet werden, sondern nur eine einzige Spalte. Die letzte Partition - VALUES (DEFAULT) - bildet die Rest-Partition und nimmt alle nicht explizit aufgeführten Daten auf. Jedoch sollte jeder Wert der vorkommen kann auch zugewiesen werden. Ohne diese Zuweisung oder eine DEFAULT-Partition erhält man einen Fehler. Per MODIFY PARTITION ADD VALUES () kann man nachträglich Werte hinzufügen. Partition Pruning setzt voraus, dass der Definitionsausdruck bspw. einer virtuellen Spalte in der Abfrage vorkommt, sonst kann der Optimizer die Partitionierung nicht berücksichtigen. Die typische Abfrageform entscheidet also über die Partitionierung.

HASH Partitioning

Sollten sich keine sinnvollen Wertelisten oder Wertebereiche finden, bleibt die Hash-Partitionierung. Hierbei sollten möglichst immer zwei oder mehr Tabellen auf ihren jeweiligen Join-Spalten gleich partitioniert werden. Hash Partitionierung sollte vor allem auf Fremdschlüsselspalten und den jeweils übergeordneten Primärschlüssel- oder Unique-Spalten vorgenommen werden. Es lassen sodann partitionsweise Joins nutzen, welche einen erheblichen Vorteil für den Optimizer bei großen Tabellen darstellt. Partition Pruning lässt sich mit Hash Partitioning nicht nutzen.

Eine interne Hash-Funktion übernimmt die Zuordnung. Die Anzahl der Hash-Partitionen kann bestimmt werden und sollte ein Vielfaches von 2 sein, es ist jedoch nicht absehbar welcher Spaltenwert in welcher Partition gespeichert wird. Dies macht ein Löschen oder Archivieren von Hash-Partitionen wertlos. Spalten mit einer möglichst hohen Selektivität, im Optimalfall eindeutige Werte, gewährleisten eine homogene Vereteilung.

Beim Join eines einzelnen Kundendatensatzes an die zugehörigen Auftragsdaten muss nun nicht auf die komplette Tabelle zugegriffen werden, sondern nur auf die passende Auftragspartition bzw. den zugehörigen Index.

Reference Partitioning

Reference Partitioning ist seit 11g verfügbar und baut auf eine per Fremdschlüssel übergeordnete Tabelle auf, welche bereits partitioniert ist. Die betreffenden Fremdschlüsselspalten müssen mit NOT-NULL-Constraints versehen sein, ansonsten erhält man bei der Erstellung einen Fehler. Häufige Joins zwischen zwei Tabellen machen eine deckungsgleiche Partitionierung sinnvoll. Bei der Definition muss der Name des Fremdschlüssel-Constraints angegeben werden PARTITION BY REFERENCE ().

Jede Umpartitionierung, also Hinzufügen, Löschen, Splitten oder Verschmelzen, einer übergeordneten Tablle vererbt sich automatisch auf die untergeordnete Tabelle. Der Administrator muss also nicht mehr auf eine konsistente Partitionierung achten. Partitionen einer untergeordneten Tabelle werden standardmäßig in den selben Tablespace wie die zugehörigen Partitonen der übergordneten Tabelle abgelegt. Eigenständiges umpartitionieren einer untergeordneten Tabelle ist jedoch nicht mehr möglich.

Reference Partitioning kann sich beliebig breit und tief verzweigen. Es lassen sich weitere Tabellen per Fremdschlüssel an eine untergeordnete Tabelle anhängen, ebenso wie an eine übergeordnete Tabelle. Umpartitionierungen der obersten Tabelle kaskadieren dabei auf alle direkt oder indirekt abhängigen Tabellen durch.

Bis 11gR2 war es nicht möglich übergeordnete Intervall-partitoinierte Tabellen mit Referenz-Partitionierung anzulegen. Mit 12c gibt es nun Interval-Reference-Partitioning, dies minimiert den Verwaltungsaufwand auf ein Minimum.

Subpartitioning

Die Verknüpfung zweier Partitionierungsarten kann bei sehr großen Tabellen sinnvoll sein. In erster Instanz setzt man dabei eine Range- oder List-Partiton ein. In zweiter Instanz wird nach Range-, List oder Hash-Methode innerhalb jeder Partition subpartitioniert. Jede Subpartition ist dabei speichertechnisch unabhängig, kann also mit eigenen Speicherparametern ausgestattet sein. Administrative Operationen also Hinzufügen, Löschen, Splitten, Verschmelzen oder physisches Verschieben sind für jede Subpartition verfügbar. Optimierungsmöglichkeiten, also Partition Pruning und partitionsweise Joins sind ebenso für jede Subpartition verfügbar.

Zwei Tabellen, welche generell inkompatibel sind, können durch Subpartitionierung über einen Fremdschlüssel, der bspw. per Hash Subpartitioniert wird, kompatibel gemacht werden – SUBPARTITION BY HASH (). Dadurch lassen sich nun partitionsweise Joins nutzen. Der Optimizer erkennt nun bei einer entsprechenden Abfrage, dass er nur die passenden Hash-Partitionen joinen muss.

INDEX Partitioning

Partitionierte Indizes lassen sich für partitionierte und nicht-partitionierte Tabellen anlegen. Partitionierte Tabellen lasssen sich auch mit nicht-partitionierten Indizes versehen. Sowohl B*-Tree als auch Bitmap-Indizes können verwendet werden. Mit der Einschränkung, dass nicht partitionierte Tabellen und partitionierte Bitmap-Indizes nicht möglich sind und Bitmap-Indizes nach den gleichen Kriterien partitioniert sein müssen wie die zugeordnete partitionierte Tabelle.

Index Partitioning differenziert nach verschiedenen Typen, welche jeweils durch zwei Eigenschaftspaare gekennzeichnet werden. Lokale und globale Indizes als ein Paar und Prefidxed und Non-Prefixed Indizes als weiteres Paar. Dies ergibt Local prefixed, Local non-prefixed und Global prefixed Indizes. Global non-prefixed Indizes werden von der Oracle Datenbank nicht unterstützt.

Ein Index ist lokal, wenn er dieselben Partitionierungskriterien wie die zugrunde liegende Tabelle aufweist – Equi-Partitionierung. Dies ergibt eine 1:1 Zuordnung zwischen Tabellen- und Indexpartition. Partitionsänderungen der Tabelle werden also 1:1 auf den entsprechden Index übertragen. Ein lokaler Index wird bei allen Operationen auf partitionierten Tabellen automatisch mitgeführt. Er ist besonders geeignet für Rolling-Window-Anwendungen. Die Erstellung erfolgt mit LOCAL.

Ein Index ist global, wenn er eigene von der Tabelle unabhängige Partitionierungskriterien aufweist oder im Extremfall überhaupt nicht partitioniert ist. Operationen auf der zugeordneten partitionierten Tabelle werden hier nicht übertragen. Zugriffe auf eine Tabellenpartition können mehrere Indexpartitionen umfassen und umgekehrt. Es sind ausschließlich Range- oder Hash-Index-Partitionierungen möglich. List- oder Composite-Index-Partitionierung ist nicht möglich. Die Erstellung erfolgt mit GLOBAL.

Ein Index ist prefixed, wenn die für die Partitionierung benutzten Spalten selbst auch indiziert sind und außerdem in der Liste der Indexspalten ganz vorne stehen.

Ein Index ist non-prefixed, wenn die Spalte nicht ganz vorne in der Liste der Indexspalten steht oder gar nicht unter den Indexspalten wäre.

Ein local prefixed Index lässt sich durch die Equi-Partitionierung einfach handhaben. Nach einem SPLIT, MOVE oder DROP ist kein Neuaufbau des Index notwendig. Der Optimizer profitiert von der 1:1 Zuordnung von Tabllen- und Indexpartition. Sie können als Unique-Index definiert werden. Die Erstellung erfolgt über CREATE [UNIQUE] INDEX LOCAL. Ein Local non-prefixed Index lässt sich nur unter bestimmten Umständen als Unique-Index verwenden. Da Schlüsselwerte in mehreren Tabellenpartitionen vorkommen können gibt es weniger Möglichkeiten für den Optimizer.

Ein global prefixed Index lässt sich als Unique-Index definieren. Er wird häufig für Archivdaten benötigt, wenn die Indizierung nach einem anderen Kriterium als dem Datum, welches meist Partitionierungsspalte ist, erfolgen soll. Es ist ausschließlich eine Range- oder Hash-Partitionierung möglich. Ein Neuaufbau der Index-Tabelle ist bei Partitionsänderungen notwendig und mit UPDATE GLOBAL INDEXES durchführbar. Auch hier gibt es weniger Möglichkeiten für den Optimizer. Erstellung über CREATE [UNIQUE] INDEX GLOBAL.

Abschluss

Mit 12c gibt es ein paar wichtige Neuerungen im Bereich Partitioning. Bespielsweise ist jetzt ein Online Move der Partition möglich, d.h. DMLs auf die Tabelle während dem MOVE sind erlaubt – MOVE PARTITION ONLINE. Die Kombination Interval-Reference-Partitioning ist endlich verfügbar. Im Bereich Reference-Partitioning kaskadieren durch Vererbung TRUNCATE und EXCHANGE PARTITION Operationen auf Child-Tabellen. Das SPLIT und MERGE Kommando wurde erweitert und man kann jetzt mit einem einzigen Befehl mehrere (Sub-)Partitionen verarbeiten. Als letzten Punkt bietet die Indexerzeugung nun mehr Flexibilität, denn man kann nun durch ein INDEXING ON / OFF pro Tabelle und / oder pro Partition lokale und globale Indizes auf eine Untermenge von Partitionen der Tabelle erzeugen.

Als Fazit bleibt die Partitionierung als mächtiges Werkzeug zur Optimierung von Zugriffen auf (sehr) große Datenbanken und deren Verwaltung hervorzuheben. Weiterführende Themen wie Information Lifecycle Management und Advanced Compression konnten nicht besprochen werden, erweitern das Spektrum aber noch um ein Vielfaches.

Mit der Oracle Enterprise Edition kann Partitionierung jeder nutzen, da sie durch Transparenz für jede Applikation direkt einsetzbar ist. Einziger Nachteil: Zusätzliche Kosten für eine Partitioning Option. Übrigens verwenden Flashback Data Archive und Unified Auditing automatisch Partitioning auch in der Standard Edition, dafür ist allerdings keine Lizenz notwendig.

Kontaktadresse:

Sebastian Winkler

CarajanDB GmbH

Siemensstraße 25

D-50374 Erftstadt

Telefon:	+49 (0) 12-345 6789
Fax:	+49 (0) 12-345 6788
E-Mail	Ihre@adresse.de
Internet:	www.adresse.de