ORACLE

ORACLE DOJO NR. 13

Oracle Database 12.2 Spezial

Ausgewählte Artikel zur Oracle Database 12c Release 2

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

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

Im *Oracle Dojo Nr. 13* beschreiben Ralf Durben, Karin Patenge, Ulrike Schwinn, Norman Sibbing und Sebastian Solbach in verschiedenen Artikeln ausgewählte Themen aus Oracle Database 12c Release 2.

ORACLE®

Inhalt

ORACLE DOJO NR. 13

Oracle Database 12.2 Spezial

Ausgewählte Artikel zur Oracle Database 12c Release 2

VORWORT

Lange erwartet und endlich verfügbar: Oracle Database 12c Release 2 ist seit März 2017 für die unterschiedlichsten Plattformen verfügbar. Groß ist das Interesse, mehr über das neue Release zu erfahren. Viele Erweiterungen basieren natürlich auf dem Hintergrund der in Release 1 eingeführten Funktionen wie zum Beispiel im Bereich Multitenant und In-Memory. Aber dies sind nicht die einzigen Neuerungen: Interessante neue Aspekte finden sich auch in anderen Bereichen, angefangen von den Basisfunktionalitäten bis zu der Möglichkeit, eine brandneue Architektur, nämlich Oracle Sharding, zu verwenden.

Aus diesem Grund haben wir uns überlegt, die wichtigsten Themen rund um Oracle Database 12c Release 2 auch im beliebten Dojo-Format zu veröffentlichen. Jedoch handelt es sich nicht – wie in den anderen Dojos – um die Beschreibung eines einzigen speziellen Sachverhalts, sondern um mehrere unterschiedliche Artikel. Bei der Zusammenstellung haben wir uns auf Features konzentriert, die uns besonders beeindruckt haben.

Wir wollen dabei nicht nur einen bestimmten Nutzerkreis ansprechen, sondern decken mit unseren Artikeln ein breites Spektrum ab – angefangen bei Entwicklungsthemen, über Designfragen bis zur Administration.

Zu einigen dieser Themen finden Sie darüber hinaus weitere Informationen in unserem deutschsprachigen Technologie-Blog. Wir hoffen damit eine breite Leserschaft zu erreichen, die im Datenbank- und Cloud-Umfeld entwickelt, verwaltet, Sicherheit schafft, Hochverfügbarkeit herstellt und vieles mehr.

Wir wünschen Ihnen viel Spaß beim Lesen und beim Testen.

Ihre Ulrike Schwinn im Namen aller mitwirkenden Autorinnen und Autoren

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

EINFÜHRUNG

Die Themen in den folgenden Kapiteln sind unabhängig voneinander ausgewählt und verfasst worden. Die Artikel müssen somit nicht in einer bestimmten Reihenfolge oder gar in ihrer Gesamtheit gelesen werden. Weitere Links und Informationen stehen jeweils am Kapitelende sowie im Bereich Weiterführende Informationen (ab S. 89). Dort finden Sie die verwendeten Handbucheinträge, weitere Blogs, Informationen zur Lizenzierung oder auch Tipps und Tricks zum Upgrade.

Um Ihnen einen leichten Einstieg und einen guten Überblick über die Themen im Dojo zu geben, sind die Artikel hier noch einmal für Sie zusammengefasst:

Bereich	Titel	Einsatzmöglichkeiten
Grundlagen	Neue Funktionen in SQL und PL/SQL	Datenbankentwicklung, Tuning
Architektur	Einführung in Oracle Sharding	Management von skalierbaren und hochverfügbaren Daten
Performance	Kontrolle über die Indexverwendung	Tuning, Testing
Sicherheit	Oracle Database 12c Release 2: Die Version, die Datenbanksicherheit einfacher macht	Datenbankverschlüsselung und PDB-Isolierung
Anwendungen mit JSON	JSON-Daten in der Oracle-Datenbank	Anwendungsentwicklung, DevOps
Anwendungsarchitektur und -betrieb	Multitenant-Architektur mit Applikationscontainern	Zentrale Speicherung und Wartung von Anwendungen
Geodaten	Neue Möglichkeiten für das Management und die Nutzung von Geodaten	Anwendungsentwicklung mit Geodaten

ULRIKE SCHWINN

Neue Funktionen in SQL und PL/SQL

Das Programmieren von Datenbankanwendungen zu erleichtern, noch mehr Funktionalität zu bieten und dabei den Performanceansprüchen zu genügen, sind wichtige Ziele bei der Weiterentwicklung der Funktionen im Datenbankumfeld. Dies ist nicht nur in den Bereichen SQL und PL/SQL, sondern auch in anderen Bereichen wie Oracle Text, Spatial, XML oder auch bei JSON-Daten zu beobachten. Performance und die Verarbeitung großer Datenmengen dürfen dabei nicht außer Acht gelassen werden sowie das Tunen und Monitoren von neuem und existierendem SQL-und PL/SQL-Code.

Nimmt man die Features für Entwickler von Oracle Database 12c Release 2 in Augenschein, so stellt man fest, dass beispielsweise eine Weiterentwicklung der in Release 1 eingeführten JSON-Features ein wichtiges Thema ist. Neu sind die Erweiterungen JSON-Dokumente in materialisierten Views zu verwenden, zu partitionieren und/oder zusätzlich „In-Memory“ zu speichern. Das neue JSON-Data-Guide-Konzept, das die Struktur und den Inhalt der JSON-Dokumente vorhält, eröffnet zusätzlich ganz neue Möglichkeiten. Eine weitere interessante Funktionalität bietet das Feature Case Insensitivity by default. Die Verwendung ist bei der Spaltendefinition im CREATE TABLE möglich – ähnlich wie bei der Festlegung auf einen Datentyp. Auf diese Weise können Sie in Spalten einer Applikation die Groß- und Kleinschreibung vernachlässigen, ohne dass ein Datenbankentwickler explizit Funktionen wie UPPER oder LOWER hinzufügen muss.

Einige interessante ausgewählte Basisfeatures werden im Folgenden beschrieben und an Code-Beispielen illustriert.

1 NEUES IM TRADITIONELLEN SQL*PLUS-WERKZEUG

Das traditionelle Werkzeug SQL*Plus wird standardmäßig als Server, Client und/oder Instant Client Software mit ausgeliefert und mit jedem neuen Datenbankrelease ebenfalls erweitert. Ich möchte an dieser Stelle auch auf die neueren modernen Werkzeuge wie SQL*Developer und SQLcl hinweisen, die vom Oracle Technology Network (OTN) ladbar und einfach nach Entpacken einer Zip-Datei verwendbar sind. Sie besitzen einen hohen Funktionsumfang, sind sehr einfach zu bedienen und stellen somit häufig eine gute Alternative zur Verwendung von SQL*Plus dar. Probieren Sie diese unabhängig vom verwendeten Datenbankrelease unbedingt aus.

Bevor ich die neuen Features erkläre, noch ein wichtiger Hinweis für alle SQL*Plus-Anwender: Falls Sie die Datei login.sql für eigene User-spezifische Einstellungen im lokalen Verzeichnis verwenden, müssen Sie ab jetzt umdenken. Aus Sicherheitsgründen wird die login.sql im lokalen Verzeichnis nicht mehr berücksichtigt, sondern wird nur noch aus den Pfadverzeichnissen ORACLE_PATH (auf Linux-Systemen) beziehungsweise SQLPATH (auf Windows-Systemen) gelesen. Näheres dazu findet sich in der „My Oracle Support Note“ (Doc ID 2241021.1).

Ein Blick auf die speziellen SQL*Plus-Umgebungseinstellungen lässt erkennen, dass einige neue Parameter und Optionen eingeführt worden sind. So kennt SQL*Plus jetzt endlich eine History-Funktion. Damit lässt sich die Kommandohistorie anzeigen und

Kommandos aus der Historie können editiert und gelöscht werden. Wie immer kann man sich eine detaillierte Funktionsbeschreibung mit help anzeigen lassen.

Zur Performancesteigerung gibt es jetzt die neue Option -F, mit der SQL*Plus in einem speziellen Modus mit speziellen Set-up-Einstellungen gestartet wird. Damit werden beispielsweise verschiedene Caches aktiviert – für ein LOB Prefetching (LOBPREFETCH 16384), Zeilen Prefetching (ROWPREFETCH 2) und für ein Statement Caching (STATEMENTCACHE 20). Diese Werte können auch unabhängig von der eingestellten Option in der SQL*Plus-Session gesetzt beziehungsweise verändert werden.

Zudem ist eine weitere interessante und wichtige Formatierungsmöglichkeit implementiert worden – die CSV-Formatierung. Einfach SQL*Plus mit -M "CSV ON" starten und schon werden die Ausgaben in CSV mit Trennzeichen (engl. delimiter) – einem Komma – angezeigt. Die Einstellung kann auch hier innerhalb von SQL*Plus vorgenommen werden. Im folgenden Beispiel soll das Trennzeichen ein Semikolon sein:

```
SQL> set markup CSV on delimiter ";"
SQL> select * from dept;

"DEPTNO";"DNAME";"LOC"
10;"ACCOUNTING";"NEW YORK"
20;"RESEARCH";"DALLAS"
30;"SALES";"CHICAGO"
40;"OPERATIONS";"BOSTON"
```

Möchte man mehr über SQL*Plus-Einstellungen und -Optionen erfahren, eignet sich die Lektüre im Handbuch *SQL*Plus User's Guide and Reference*.

2 LÄNGERE BEZEICHNER, SQL-FUNKTIONSERWEITERUNGEN UND DIE „APPROXIMATE"-FUNKTIONALITÄT

Eine wichtige Neuigkeit für Entwickler und DBAs vorab: Die maximale Länge von Identifier ist von 30 auf 128 Bytes erhöht worden. Das bedeutet, dass nun Namen für Tabellen, Objekte, Prozeduren, Variablen und so weiter länger sein können. Das Limit ist natürlich additiv: Bei Verwendung von "schema"."table"."column" (inklusive Punkte und doppelte Anführungszeichen) ist somit eine maximale Länge von 392 Bytes möglich. Übrigens: Falls man die Länge überprüfen will, kann man die neue Funktion ORA_MAX_NAME_LEN_SUPPORTED verwenden:

```
SQL> exec dbms_output.put_line(dbms_standard.ora_max_name_len_supported);
128
```

Mehr Informationen zu weiteren Limitierungen in der PL/SQL-Programmierung kann man im *PL/SQL Language Reference Guide* im Abschnitt „PL/SQL Program Limits" finden.

Ein weiteres Ziel bei der Entwicklung von Features ist die Möglichkeit, mehr Kontrolle im Fehlerfall zu haben. Speziell bei den Konvertierungsfunktionen TO_NUMBER, TO_DATE, TO_TIMESTAMP und so weiter kann nun eine Fehlerausgabe vermieden

und stattdessen ein Defaultwert ausgegeben werden. Ein einfaches Beispiel liefert die TO_NUMBER-Funktion: Wird als Eingabeparameter eine DATE-Spalte mitgegeben, erhält man vor Release 12.2 den Fehler ORA-01722: invalid number. Ab 12.2 kann man diese Ausgabe mit einem Defaultwert umgehen wie folgendes Beispiel zeigt:

```
SQL> select to_number(hiredate default 42 on conversion error) ausgabe
     from emp where rownum = 1;
   AUSGABE
----------
        42
```

Mit der neuen Funktion VALIDATE_CONVERSION ist es darüber hinaus möglich zu überprüfen, ob ein Ausdruck in einen speziellen Datentyp konvertiert werden kann. Der Wert „1“ bedeutet, dass eine erfolgreiche Konvertierung möglich ist, der Wert „0“ hingegen, dass die Konvertierung fehlschlagen wird. Mögliche Datentypen sind dabei die numerischen und die Datums-Datentypen DATE, NUMBER, TIMESTAMP, BINARY_DOUBLE, INTERVAL YEAR TO MONTH und so weiter. Folgendes Beispiel demonstriert die einfache Verwendung. Im ersten Fall wird der Wert „1000“ überprüft. Die Ausgabe ist „0“, da „1000“ kein Datum darstellt:

```
SQL> select validate_conversion('1000' as date ) ergebnis from dual;
  ERGEBNIS
----------
         0
```

Unter der Verwendung einer Formatmaske ergibt die Abfrage auf DATE allerdings Sinn, sodass wir den Wert „1“ erhalten – eine Konvertierung ist möglich:

```
SQL> select validate_conversion('1996' as date ,'YYYY') ergebnis from dual;
  ERGEBNIS
----------
         1
```

Einige weitere interessante Funktionserweiterungen finden sich im Umfeld der analytischen Windowfunktionen. Die Funktion LISTAGG, die eine Stringaggregation vornimmt, bietet nun eine Ausgabemöglichkeit bei Überschreitung der maximalen Ausgabelänge an. Statt eines ORA-01489-Fehlers kann nun eine Konstante ausgegeben werden. Dies kann auch mit einem Zähler kombiniert werden, der die Anzahl der überzähligen Zeichen ausgibt. Im folgenden Beispiel ist das Trennzeichen ein Semikolon (;), die Zeichenkette, die die Überschreitung der Grenze angibt, die Konstante '...' und die Anzahl der überzähligen Zeichen „3631“. Leider ist es dabei nicht möglich, die überschüssigen Zeichen auszugeben oder zu speichern:

```
SQL> set pagesize 100000
SQL> select listagg(s, ';' on overflow truncate '...' with count)
     within group (order by r) ausgabe
     from (select rownum r , 'x' s from dual connect by level
<20000);
```

```
AUSGABE
------------------------------------------------------
x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;
x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;
x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;
x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;
x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;
x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;
x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;x;
x;x;x;x;x;x;x;x;...(3631)
```

Hinweis: Das Limit für die Grenzüberschreitung ist hier ungewohnt hoch, weil der Parameter MAX_STRING_SIZE in meiner Umgebung auf EXTENDED gesetzt wurde.

Mit Oracle Database 12c Release 1 ist die Windowfunktion APPROX_COUNT_DISTINCT eingeführt worden. Die Idee dabei war, die Ausführung von COUNT (DISTINCT) zur Berechnung der Anzahl von verschiedenen Spalten-Ausprägungen zu beschleunigen. APPROX_COUNT_DISTINCT führt ein Sampling durch, um die ungefähre Anzahl der verschiedenen Einträge einer Spalte zu berechnen und ist in der Regel um ein Vielfaches schneller als COUNT (DISTINCT). Der Wermutstropfen dabei war, dass man dazu die Funktionen im eigenen Code austauschen musste. Mit Oracle Database 12c Release 2 sind daher zusätzliche Sessionparameter eingeführt worden. Beispielsweise führt das einfache Setzen des Sessionparameters APPROX_FOR_COUNT_DISTINCT auf TRUE zu der impliziten Anwendung von

APPROX_COUNT_DISTINCT ohne Änderung am Code. In dem Zusammenhang sind noch weitere „Approximate“-Funktionen eingeführt worden, zum Beispiel APPROX_COUNT_DISTINCT_AGG, APPROX_MEDIAN, APPROX_PERCENTILE.

3 SQL IN PL/SQL-PROGRAMMEN

Gerade beim Tunen und Analysieren von PL/SQL-Code stellt man sich auch häufig die Frage, ob und welche SQL-Statements im PL/SQL-Code verwendet wurden. Vor 12.2 war die Lösung dieser Fragestellung eher ein umständliches Unterfangen oder gar ein voneinander getrennter Vorgang. Jetzt in 12.2 ist es einfach und intuitiv möglich, SQL-Statements in den PL/SQL-Programmen aufzuspüren.

Eine Änderung dazu findet sich im PL/SQL Hierachical Profiler, der ein wichtiges Werkzeuge ist, um Ursachen für schlechte Performance (Bottlenecks) zu finden und Performance-Tuning für PL/SQL-Code durchzuführen. Das Set-up und die Handhabung haben sich nicht verändert. Es ist immer noch ein EXECUTE-Recht auf DBMS_HPROF, ein Schreibrecht auf ein Directory und das Starten und Stoppen über DBMS_HPROF erforderlich. Führt man die Analyse mit dem Linemode-Werkzeug plshprof in 12.2 durch, stellt man allerdings fest, dass DBMS_HPROF nun auch die entsprechenden SQL-ID und den SQL-Text mitliefert. Vor 12.2 wurden nur Ausführungsinformationen der Programme

und Unterprogramme einschließlich der betroffenen Zeilenzahl im Sourcecode ausgegeben. Jetzt ist nicht nur der SQL-Text, sondern auch die SQL-ID im Report aufgelistet. So ist es sehr einfach möglich, den zugehörigen SQL-Code zu analysieren und weitere Rückschlüsse, beispielsweise über den SQL-Monitor, zu ziehen.

Eine ähnliche Idee steckt auch hinter der Erweiterung von PL/Scope. Worum handelt es sich noch einmal bei PL/Scope? Mit PL/Scope können Informationen über die Verwendung von Identifier während der Compile-Zeit gesammelt und automatisch in Data-Dictionary-Tabellen gespeichert werden. Dazu benötigt PL/SCOPE Platz im Tablespace SYSAUX, den ein Administrator von Zeit zu Zeit überprüfen kann. Aufgezeichnet werden Typen, Namen, Nutzung und Vorkommen der Identifier im PL/SQL-Code. So ist eine einfache Analyse der verwendeten Identifier im Code möglich. Mit dem Parameter PLSCOPE_SETTINGS lässt sich die Nutzung aktivieren und einstellen. In 12.2 lassen sich damit nun auch SQL-Statements mitloggen. Dazu wurden neue Werte wie 'IDENTIFIERS:SQL, STATEMENTS:ALL', 'IDENTIFIERS:ALL, STATEMENTS:ALL','IDENTIFIERS:PLSQL, STATEMENTS:NONE' eingeführt. Um PL/Scope-Daten für alle SQL-Statements in den PL/SQL-Programmen aufzulisten, setzt man den PL/SQL-Compile-Parameter PLSCOPE_SETTINGS auf 'STATEMENTS:ALL':

```
alter session set PLSCOPE_SETTINGS ='IDENTIFIERS:SQL, STATEMENTS:ALL'
```

Nach der Kompilierung des Package STOCK_TRACK_PACK und der Prozedur P1, kann man jetzt nicht nur die Informationen über die

Identifier selbst selektieren, sondern auch die SQL-Statements (mit SQL_ID und SQL_TEXT) herausfiltern. Die neue View USER/ALL/DBA_STATEMENTS listet die SQL-Statements auf. Jede Zeile gibt dabei ein SQL-Vorkommen im PL/SQL-Code aus:

```
SQL> select object_name, object_type, line, col, sql_id, full_text
     from user_statements;

OBJECT_NAME           OBJECT_TYPE        LINE        COL SQL_ID
--------------------- ------------ ---------- ---------- -------------
FULL_TEXT
--------------------------------------------------------------------------
STOCK_TRACK_PACK      PACKAGE BODY         46          3

STOCK_TRACK_PACK      PACKAGE BODY         23         10 6nwmx5nabvmrc
INSERT INTO STOCK_HISTORY VALUES (:B1 ,:B2 , :B3 , :B4 , :B5 ,NULL, :B6 )

P1                    PROCEDURE             5          5 2fuxfptxnlOpk
SELECT ENAME FROM EMP WHERE EMPNO = :B1
```

Die neuen Features vereinfachen die Programmierung und helfen beim Tuning von PL/SQL und SQL. Zum Erlernen kann man dafür das Standardwerkzeug SQL*Plus in Verbindung mit den neuen Möglichkeiten verwenden. Die Werkzeuge wie SQL Developer oder die neue Linemode-Variante SQLcl bieten darüber hinaus gerade beim Testen und Programmieren weitere hilfreiche Funktionen, wie zum Beispiel zum Generieren und Vergleichen von Ausführungsplänen.

Generell ist dabei immer wichtig: Man sollte wissen, was man programmiert hat und welche Auswirkungen der Code haben kann. Dazu empfiehlt es sich, den eigenen Code zu überprüfen, nicht nur in Bezug auf das Ergebnis, sondern wenn möglich sich auch ein Bild von der Ausführungsperformance und vom Ausführungsplan zu machen. Ähnliches gilt auch für PL/SQL. Um langsam laufenden PL/SQL-Code zu monitoren beziehungsweise zu tunen, sollte man den PL/SQL Hierarchical Profiler verwenden. Dieser beinhaltet – wie auch PL/Scope ab 12.2 – den SQL_TEXT und die SQL_ID, sodass das Monitoring umfassend möglich ist. Übrigens: Ob die Funktion in der Cloud oder in Ihrer eigenen Umgebung (soweit verfügbar) getestet wird, spielt dabei keine Rolle: die Funktionen bleiben die gleichen.

SEBASTIAN SOLBACH

Einführung in Oracle Sharding

Sharding bezeichnet eine Funktionalität, große Datenmengen auf verschiedene Server zu verteilen. Hierzu werden die Daten einzelner Tabellen in unabhängige, benutzerdefinierte Teilmengen aufgeteilt, den sogenannten Shards. Das Besondere bei Sharding ist, dass sich die einzelnen Shards nicht auf einem Server und in einer Datenbank befinden, sondern in komplett unterschiedlichen Datenbanken. Die einzelnen Datenbanken haben dabei keine gemeinsamen Komponenten – weder verwenden sie dieselbe Software, noch die gleiche Hardware.

Letztendlich können die einzelnen Shards komplett unabhängig voneinander operieren, allerdings nur auf denen für sie zugewiesenen Daten. Diese Eigenschaft ist bei Sharding der größte Vorteil, aber gleichzeitig auch die größte Restriktion.

Einerseits erlaubt dies auch dann mit einzelnen Shards weiterzuarbeiten, wenn andere Shards beziehungsweise Datenbanken nicht verfügbar sind. Ein Ausfall einzelner Komponenten wird immer nur ein Shard betreffen, andere Shards sind davon nicht beeinträchtigt. Andererseits ist es möglich, eine Datenbank bei hohem Lastaufkommen einfach in mehrere Shards zu teilen und auf weitere Hardware auszulagern und somit eine lineare Skalierbarkeit zu gewährleisten. Mithilfe einer sharded Datenbank kann auch der Anforderung gesetzlicher Vorgaben Genüge getan werden, wenn diese zum Beispiel erfordern, dass Daten nur im jeweiligen Land gespeichert sein dürfen.

Soll eine Applikation aber eine Sharding-Architektur verwenden, muss im Vorfeld bekannt sein, auf welchem Shard gearbeitet wird, und die einzelnen Prozesse und Geschäftsabläufe sollten idealerweise nur innerhalb eines Shards ablaufen. Sobald einzelne Geschäftsprozesse mehrere Shards betreffen, ist ein erhöhter Koordinationsaufwand notwendig. Ebenso ist bei Abfragen, die mehrere Shards überspannen keine übergreifende Lesekonsistenz gewährleistet: Abfragen, die parallel in unterschiedlichen Shards ausgeführt werden, können zu unterschiedlichen Zeitpunkten starten und enden.

Dennoch bietet die Sharding-Architektur gegenüber klassischen Datenbanken gerade dann Vorteile, wenn nur auf sehr begrenzten „lokalen“ Daten gearbeitet wird.

1 ORACLE-SHARDING-ARCHITEKTUR: FUNKTIONALITÄTEN UND BEGRIFFLICHKEITEN

Die klassische relationale Datenbank von Oracle zeichnet sich durch eine auf atomic, consistent, isolated und durable (ACID) ausgelegten Architektur aus. Dank der seit Oracle 6 integrierten Lesekonsistenz gibt es keinen Dirty Read in einer Oracle-Datenbank. Dabei ist es auch gleich, ob man hier eine Single-Instanz-Datenbank oder einen Cluster-Verbund auf Basis des Real Applikation Cluster betrachtet. Allerdings bedeutet dies auch, dass alle Daten zentral liegen. Zwar gibt es auch Replikationsmechanismen wie

Golden Gate. Im Gegensatz zu Sharding existiert bei diesen Lösungen aber normalerweise mindestens ein Master, der alle Daten enthält.

Um von beiden Architekturen zu profitieren – der Sharding-Architektur und den Vorteilen der Oracle-Datenbank – haben einige große Kunden in der Vergangenheit eine Sharding-Architektur auf Basis mehrerer Oracle-Datenbanken selbst implementiert. Die Nutzung war allerdings nur mit stark erhöhtem Aufwand bei der Administration und Wartung möglich.

Dies war einer der Gründe, warum Oracle selbst diese neue Architektur in Oracle Database 12.2 zur Verfügung stellt, um auch den administrativen Aufwand so gering wie möglich zu halten. Dazu gehört das automatische Anlegen von Hunderten von Shards durch einfache SQL-Syntax, inklusive der Absicherung einzelner Shards durch Active Data Guard, Golden Gate und/oder RAC.

Dies funktioniert über eine zentrale Definition der einzelnen Sharded Tables und sogenannter duplizierter Tabellen. Eine automatische Verteilung der Daten ist ebenso integriert wie ein Lifecycle-Management, sollten Änderungen an den Tabellen notwendig sein.

Ebenso bietet Oracle Sharding eine zentrale Zugriffsebene auf die einzelnen Shards, um auch Abfragen zu erlauben, bei denen im Vorfeld nicht bekannt ist, in welchem Shard beziehungsweise in welchen Shards (mehrere Shards überspannend) die Daten liegen. Auch wenn letztere Abfragen nicht empfohlen sind.

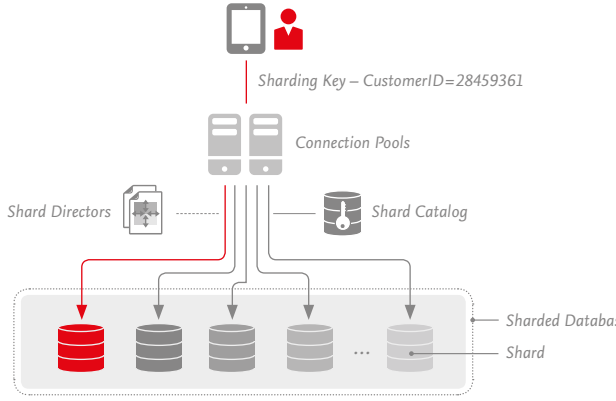


Abb.1: Architektur einer sharded Datenbank

Welche neuen Begrifflichkeiten und Architekturen gibt es nun im Umfeld einer sharded Datenbank, kurz SDB (nicht zu verwechseln mit Single Instanz DB)?

Zuerst einmal gibt es die einzelnen Shards. Dies sind die jeweiligen unabhängigen Datenbanken mit eigenem CPU, Memory und dazugehörigem Storage aus denen die sharded Datenbank besteht. Innerhalb eines Shards existieren abhängige Tabellen. Diese Tabellen können entweder verteilte Tabellen (Sharded Tables) oder duplizierte Tabellen (Duplicated Tables) sein, die in jeder einzelnen Datenbank repliziert vorliegen. Auf die Möglichkeiten und Eigenschaften der sharded und duplizierten Tabellen wird weiter unten noch eingegangen.

Damit auf eine sharded Datenbank zugegriffen werden kann, muss es einen oder mehrere zentrale Koordinatoren geben, welche die initiale Weiterleitung zum richtigen Shard übernehmen. Diese Aufgabe wird von den Shard Directors oder kurz GSM wahrgenommen (Global Service Managers, die übrigens in den mit Version 12.1 eingeführten Global Data Service integriert sind). GSM führen in diesem Zusammenhang auch den Globalen Service ein, der nichts anderes als eine Erweiterung der bestehenden Datenbankservices ist. So kennen Global Services auch Eigenschaften wie Netzwerk-Latenzen, Replikationslag und Regionsabhängigkeiten.

Diese und weitere Informationen über Services und deren Verteilung liegen in einer weiteren Datenbank, dem Shard-Katalog. Zusätzlich dient dieser auch als eine zentrale Verwaltungseinheit, die das Anlegen der einzelnen Shards überwacht und anstößt sowie die Tabellendefinitionen und die Desaster-Recovery-Mechanismen der einzelnen Shards enthält und das Lifecycle-Management übernimmt.

2 ORACLE-SHARDING-DATENMODELL UND APPLIKATIONSANFORDERUNGEN

Innerhalb von Sharding unterscheidet man zwischen zwei neuen Tabellentypen: Sharded Tables und Duplicated Tables. Sharded Tables basieren auf einer Ursprungstabelle, die ihre Eigenschaften auf die einzelnen Tabellen innerhalb eines Shards vererben. Wichtig ist, dass jede dieser Sharded Tables einen Sharding-Schlüssel (Shard Key) enthalten muss –

dies muss die führende Spalte im Primärschlüssel sein. Dieser Sharding-Schlüssel ist allen Sharded Tables gemein. Anhand des Sharding-Schlüssels übernimmt Oracle auch das Anlegen der einzelnen Shards und verteilt die Daten dementsprechend, vergleichbar mit dem Partitionierungsschlüssel bei der Partitionierung. Alle abhängigen Partitionen (Tabellen mit gleichem Sharding-Schlüssel-Wert) werden hierbei als „Chunk" bezeichnet. Dies ist gleichzeitig die Einheit in der einzelne Partitionen im Falle eines Resharding neu verteilt werden können.

Auf Basis dieses Sharding-Schlüssels unterstützt Oracle 12.2 zwei Arten der Datenverteilung: system managed (konsistenter Hash) und einen zusammengesetzten Range-Hash oder List-Hash.

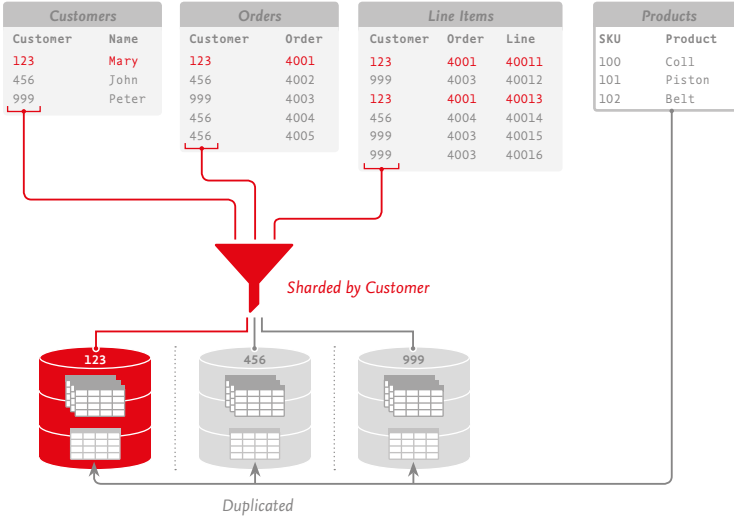
Überlässt man die Datenverteilung einfach einem konsistenten Hash, so werden die Daten uniform über alle Shards der sharded Datenbank verteilt. Die einzelnen Chunks werden dabei relativ klein und sind gleich groß. Allerdings hat der Administrator dadurch keinen Einfluss auf die Lokalität der Daten.

Aus diesem Grund gibt es das Composite Sharding, welches letztendlich eine Zusammensetzung von zwei Sharding-Schlüsseln ist: einem führenden Super Shard Key, der zum Beispiel die Lokation angibt (wie Amerika, Europa, Asien) und dem oben erwähnten Sharding-Schlüssel für den konsistenten Hash. Damit kann die Lokation der Daten gesteuert werden. Logischerweise sind dadurch die einzelnen Chunks aber nicht mehr zwingend gleich groß, da die Größe nun abhängig von den Daten ist.

Sharded Tables gleichen daher in ihrer Art und Weise durchaus dem in einer Oracle-Datenbank existierenden Partitioning. Nur dass die einzelnen Partitionen nicht innerhalb einer Datenbank existieren, sondern in unterschiedlichen Datenbanken gehalten werden.

Duplizierte Tabellen hingegen sind Tabellen, die in jedem Shard vorkommen und deren Daten letztendlich vom Shard-Katalog aus in jede Datenbank repliziert werden. Technisch gesehen sind duplizierte

Abb. 2: Sharded und duplizierte Tabellen



Tabellen Materialized Views in jedem Shard, welche regelmäßig (Default alle 60 Sekunden) vom Master aktualisiert werden.

Mehr Informationen zu sharded und duplizierten Tabellen findet sich im *12.2 Database Administrator's Guide* Kapitel 50.

Die größte Änderung ergibt sich beim Sharding aber hinsichtlich der Applikationsentwicklung. Denn Applikationen müssen mit der Absicht programmiert worden sein, dass diese mit einer sharded Datenbank arbeiten. An einem einfachen Beispiel wird dies sofort ersichtlich: Ein direktes Update auf den Sharding-Schlüssel ist nicht möglich, wenn dieses Update dazu führt, dass der Datensatz in einen anderen Chunk beziehungsweise Shard verschoben werden müsste, da es hierfür keine Automatismen gibt und geben kann. Bei sharded Datenbanken erfordert dies ein Löschen und eine Neuanlage des Datensatzes. Was sich im ersten Augenblick einfach und selbstverständlich anhört, wird komplex, wenn viele abhängige Datensätze vorhanden sind, bei denen sich dann ebenfalls der Wert des Sharding-Schlüssels ändert.

Ebenso profitieren Applikationen am meisten, wenn diese nur innerhalb eines Shards arbeiten und auch beim Zugriff mitteilen, mit welchem Shard gearbeitet werden soll. Hierzu sollten Applikationen bereits bei der Verbindung zur sharded Datenbank den Wert des Sharding-Schlüssels angeben. Oracle-integrierte Connections Pools (UCP, OCI, ODP.NET, JDBC) kennen diese Anforderungen und halten auch Verbindungen zu den jeweiligen

Shards vor. Hierbei ist nur bei der Initiierung des Connection Pools oder neuen Verbindungen eine Information vom Shard-Katalog notwendig. Danach werden die entsprechenden Session-Attribute im Connection Cache gehalten und direkt an die jeweilige Applikation ausgegeben. Nur wenn eine Neuverteilung der Shards ansteht, werden die Connections neu initialisiert.

Falls es der Applikation nicht möglich ist, den Sharding-Schlüssel zu spezifizieren oder falls Abfragen über mehrere Shards erforderlich sind, tritt das Proxy Routing auf den Plan. Dann werden über den Shard-Katalog die jeweiligen Shards ermittelt und die Abfragen an diese Shards weitergegeben. Dies führt jedoch zu einer schlechteren Performance, da die Sharding Directors am Ende auch die Datenaggregation übernehmen müssen.

3 VERFÜGBARKEIT DER SHARDS

Die Verfügbarkeit der einzelnen Shards kann über drei Mechanismen bewerkstelligt werden. Die jeweilige Konfiguration funktioniert automatisch und wird beim Anlegen der Shards aus dem Shard-Katalog heraus gesteuert.

Die einfachste Variante ist, jedem Shard eine Active-Data-Guard-Standby-Datenbank mit Fast Start Failover zur Seite zu stellen. Damit dies bei der Erstellung automatisch erzeugt werden kann, wurde übrigens der Database Configuration Assistant (DBCA) erweitert und

unterstützt mit 12.2 nun auch das Anlegen von Standby-Datenbanken. Allerdings ist diese Konfiguration rein aktiv/passiv, sodass es bei Ausfall eines Shards durchaus zu kleineren Verzögerungen kommen kann.

Die beste Lösung bietet Oracle Golden Gate: Dabei können sowohl ganze Shards als auch nur jeweils einzelne Chunks in ein anderes Shard repliziert werden. Hierbei kann die Redundanz angegeben werden, sodass ein Chunk auch in mehrere Shards repliziert wird. Die Golden-Gate-Sharding-Regeln sorgen für eine automatische Konflikterkennung und -lösung, sollten unterschiedliche Prozesse auf denselben Chunks in unterschiedlichen Shards arbeiten.

Kombinieren lassen sich beide Lösungen noch mit Oracle Real Application Cluster, der jeweils Hochverfügbarkeit für einzelne Datenbanken (Shards) gewährleistet.

4 DIE NEUE SHARDING-ARCHITEKTUR VS. ZENTRALE DATENBANK

Den Vorteilen disjunkter einzelner Shards, die unabhängig voneinander operieren und deshalb Flexibilität in Bezug auf Update und Verfügbarkeit bieten, steht eine Applikationsanpassung und die gestiegenen Anforderungen an die Shard Directors und des Shard-Katalogs gegenüber.

Eine funktionierende lokale, zentrale Zwei-Knoten-RAC-Cluster-Datenbank durch eine Sharding-Architektur abzulösen, macht sicherlich weniger Sinn, als eine große, global operierende OLTP-Applikation auf mehrere Shards zu verteilen, soweit es das Applikationsmodell zulässt. Auch sollte man die Anforderungen an einen verfügbaren Shard-Katalog nicht außer Acht lassen.

Nichts desto trotz bietet Oracle Sharding eine ideale Möglichkeit, verteilte Datenbanken bereitzustellen und automatisiert zu administrieren. Dennoch muss nicht auf die sonstigen Merkmale einer relationalen Datenbank verzichtet werden, wie Verfügbarkeit, funktionierende Desaster-Recovery-Konzepte, Performance-Tuning und Sicherheit. Lediglich die Multitenant-Architektur wird als Shard im Moment noch nicht unterstützt. Jedes einzelne Shard profitiert sonst aber von integrierten Oracle-Technologien, wie beispielsweise In-Memory-Datenbank, Verschlüsselung und Komprimierung.

ULRIKE SCHWINN

Kontrolle über die Indexverwendung

Immer wieder gibt es Anfragen zum Thema „Überprüfung der Indexnutzung in der Oracle-Datenbank“. Meist geht es darum herauszufinden, ob Indizes gänzlich ungenutzt sind oder selten verwendet werden. Wie wahrscheinlich jedem Oracle-Anwender bewusst ist, können Indizes vorteilhaft für den Zugriff sein, auf der anderen Seite kann die Ausführung von DML-Operationen dadurch allerdings auch verlangsamt werden. Indizes müssen nämlich bei DML-Operationen mitgepflegt werden.

Es gibt mehrere Möglichkeiten die Indexnutzung zu überwachen. Eine neue Variante ist in Oracle Database 12c Release 2 hinzugefügt worden.

1 MÖGLICHKEITEN ZUM INDEX TRACKING VOR 12.2

Seit jeher gibt es verschiedene Methoden herauszufinden, ob und wie ein Index verwendet wird (auch kurz: Index Tracking). Jede Methode hat dabei ihre Vor- und Nachteile und ist mit unterschiedlichem Aufwand verbunden. Eine erste Idee um Indizes zu monitoren, bietet sicherlich die manuelle Methode: das Generieren beziehungsweise Selektieren des Ausführungsplans. Hier kann man schnell erkennen, ob Indizes verwendet werden und welche Ausführungsstatistiken zugrunde liegen:

```
select distinct job_id from employees;
...
SQL> select * from table(dbms_xplan.display_cursor());
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
SQL_ID  636qd26k255mx, child number 0
-------------------------------------
select distinct Job_id from employees
Plan hash value: 1812349206
-------------------------------------------------------------------------------
| Id  | Operation          | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |            |       |       |     2 (100)|          |
|   1 |  SORT UNIQUE NOSORT|            |    19 |   171 |     2  (50)| 00:00:01|
|   2 |   INDEX FULL SCAN  | EMP_JOB_IX |   107 |   963 |     1   (0)| 00:00:01|
```

Möchte man die Ausführung in Views monitoren, stehen die Views V$SQL_PLAN und DBA_HIST_SQL_PLAN zur Verfügung. Diese Methode kann allerdings recht aufwendig und zeitintensiv bei der Überprüfung vieler Indizes werden.

Einfacher geht es mit dem Advisory Framework, dem SQL Access Advisor, der im Linemode über das Package DBMS_ADVISOR oder – intuitiv und einfach – grafisch im Enterprise Manager nutzbar ist. Der SQL Access Advisor steht wie (fast) alle Advisors ohne Installation in der Datenbank zur Verfügung, erfordert aber die Lizenzierung des Oracle Database Tuning Packs. Die Anwendung des SQL Access Advisors funktioniert ganz einfach: Man

benötigt nur einen Workload, den man über ein SQL Tuning Set (STS) zur Verfügung stellt. Das Ergebnis gibt dann beispielsweise an, welche Indizes erhalten bleiben sollen oder welche gelöscht werden können.

Darüber hinaus existiert seit jeher auch ein dediziertes Index Usage Tracking, das auch unter dem Namen Index Monitoring bekannt ist. Es steht in jeder Edition zur Verfügung, muss allerdings pro Index mit der speziellen Syntax MONITORING USAGE wie folgt aktiviert werden:

```
alter index emp_ix monitoring usage;
```

oder auch deaktiviert werden:

```
alter index emp_ix nomonitoring usage;
```

Mit V$OBJECT_USAGE lässt sich dann erschließen, ob ein Index überhaupt einmal verwendet wurde. Es ist wichtig zu wissen, dass Oracle während des Parsens diese Entscheidung fällt und einen Eintrag dazu tätigt. Folgendes Beispiel zeigt ein Ergebnis:

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

INDEX_NAME TABLE_NAME MON USE START_MONITORING    END_MONITORING
---------- ---------- --- --- ------------------- -------------------
EMP_IX     DEPT       YES YES 01/04/2017 21:50:43
```

Der Nachteil dieser Methode ist offensichtlich: Man kann nicht erkennen, wann oder gar wie häufig ein Index verwendet wurde. Nur ein

einziger Eintrag mit Flag und Uhrzeit erscheint. Es wird auch nicht ersichtlich wie die Indexstatistiken ausfallen. Zudem muss man alle Indizes manuell ein- beziehungsweise ausschalten, wenn man die erneute Nutzung evaluieren will. Eine Lösung dazu bietet jetzt Oracle Database 12c Release 2.

2 INDEX USAGE TRACKING IN ORACLE DATABASE 12.2

Im aktuellen Datenbankrelease (12.2) ist das Index Monitoring nicht nur stark vereinfacht worden, sondern liefert darüber hinaus auch detaillierte Ergebnisse. Neue Data-Dictionary-Strukturen speichern automatisch Informationen über den Gebrauch von Indizes in der Datenbank. So kann man leicht erkennen, welche Indizes überhaupt verwendet werden und wie häufig ein Index genutzt wird. Auf diese Weise lässt sich leicht herausfinden, ob die gewählte Indexstrategie die richtige ist oder ob es Indizes gibt, die wegfallen könnten. Kein Einschalten des Features ist erforderlich und es steht in allen Editionen zur Verfügung.

Die neue V$-View V$INDEX_USAGE_INFO gibt allgemeine Informationen zum Index Monitoring und zeigt an, wann der letzte Flush (Leerung) erfolgt ist. Beim Flush werden alle 15 Minuten die Informationen über die Indexstatistiken in die neue Data-Dictionary-Tabelle DBA_INDEX_USAGE übertragen. In der folgenden Umgebung, nach dem Öffnen einer PDB mit Namen PDB1, hat die V$View folgende aktuelle Werte:

```
SQL> select index_stats_enabled enabled, index_stats_collection_type type,
     active_elem_count active, alloc_elem_count alloc, max_elem_count max,
     flush_count, last_flush_time last_flush, con_id
     from v$index_usage_info;

ENABLED     TYPE ACTIVE ALLOC    MAX FLUSH_COUNT         LAST_FLUSH CON_ID
------- -------- ------ ----- ------ ----------- ------------------ ------
      1        1      0     0  30000           0                  0      3
```

Die ersten beiden Spalten weisen darauf hin, dass und wie das Feature eingeschaltet ist. Der Wert „1“ in der Spalte INDEX_STATS_ENABLED steht dabei für „Enabled“ (eingeschaltet). Bei INDEX_STATS_COLLECTION_TYPE gibt der Wert „1“ an, dass für die Statistiken ein Sampling bei den Ausführungen verwendet wird. Dies bedeutet, dass nicht bei jeder Ausführung eines Statements die Indexnutzung aufgezeichnet wird. Ändern lassen sich diese Werte nur über entsprechende Underscore-Parameter, was nicht empfohlen ist beziehungsweise nur mit Support oder Consulting-Unterstützung von Oracle geschehen sollte. Die Spalte LAST_FLUSH_TIME gibt dabei an, wann der letzte Flush erfolgt ist.

Um ein wenig Last zu erzeugen, wird in SQLcl mit dem REPEAT-Kommando eine Query gestartet, die einen Index verwendet und 1000 Mal ausgeführt wird:

```
SQL> select distinct job_id from employees;
JOB_ID
----------
AC_ACCOUNT
...
```

```
ST_MAN
19 rows selected.
SQL> repeat 1000 1
```

Selektiert man erneut V$INDEX_USAGE_INFO, hat sich der Wert der Spalte ACTIVE_ELEM_COUNT auf „1“ geändert. Zum letzten Mal erfolgte ein Flush dabei um 10:12 Uhr. Mittlerweile sind seit Öffnen der PDB auch schon 5 Flushes erfolgt:

```
SQL> select active_elem_count active,
     alloc_elem_count alloc, flush_count, last_flush_time last_flush,
     con_id from v$index_usage_info;

ACTIVE       ALLOC FLUSH_COUNT LAST_FLUSH           CON_ID
------- ---------- ----------- -------------------- ------
      1          1           5 05.01.2017 12:37          3
```

Nach ca. 15 Minuten wird das Ergebnis in der neuen Data Dictionary View DBA_INDEX_USAGE selektiert. Interessant sind im Moment nur der Owner HR und der Index EMP_JOB_IX:

```
SQL> select TOTAL_ACCESS_COUNT total_access, …
     from dba_index_usage where owner='HR' and index_name= 'EMP_JOB_IX';

TOTAL_ACCESS TOTAL_EXEC TOTAL_ROWS_RETURNED
------------ ---------- -------------------
         388        388               41516
```

Wie man erkennen kann, ist der Index EMP_JOB_IX mittlerweile 388 Mal verwendet worden, das bedeutet 41516 Zeilen sind durch den Index im Zugriff – also 107 pro Ausführung. Weiter vorne

im Ausführungsplan sieht man das gleiche Ergebnis. Zusätzliche Spalten, die aus Gründen der Übersichtlichkeit hier nicht mitausgegeben werden, starten mit dem Präfix BUCKET_. Sie lauten BUCKET_101_1000_ACCESS_COUNT oder BUCKET_1000_PLUS_ACCESS_COUNT beziehungsweise BUCKET_101_1000_ROWS_RETURNED und BUCKET_1000_PLUS_ROWS_RETURNED und sollen helfen, zusätzlich eine Klassifizierung der Verwendung vornehmen zu können.

Wird das Experiment mit weiteren Abfragen fortgeführt, werden weitere Einträge folgen. Zusätzlich wird angezeigt, wann ein Index zum letzten Mal und wie häufig dieser verwendet wurde. Dies sind sicherlich wichtige Kriterien bei der Entscheidung für eine gute Indexstrategie:

```
SQL> select object_id, name, total_access_count, total_rows_returned, last_used
     from dba_index_usage …;

OBJECT_ID  NAME         TOTAL_ACCESS_COUNT TOTAL_ROWS_RETURNED LAST_USED
---------- ------------ ------------------ -------------------- ----------------
     88289 EMP_JOB_IX                  454                48578 05.01.2017 10:42
     88296 LOC_CITY_IX                 143                 3289 05.01.2017 13:57
```

Da die Funktion ausführungsbasierend arbeitet, kann man leicht erkennen, wann ein Index zum letzten Mal im Zugriff war, wie viele Zeilen der Index nutzte (wichtig zur Bestimmung der Güte eines Index) und wie häufig dieser verwendet wurde. Beachten sollte man, dass das Feature aus Performancegründen mit einem Sampling-

Algorithmus konfiguriert ist. Das heißt, nicht jede Ausführung wird automatisch berücksichtigt.

Da die Verwendung sehr einfach ist und keinen zusätzlichen Overhead beinhaltet, sollte man diese Art des Index Trackings bei jeder Tuningmaßnahme einplanen. Vorausgesetzt ist dabei allerdings, dass Zugriffe auf die Data Dictionary Views DBA_INDEX_USAGE und die V$- View V$INDEX_USAGE_INFO möglich sind.

NORMAN SIBBING

Oracle Datenbank 12c Release 2: Die Version, die Datenbanksicherheit einfacher macht

Oracle führt die Weiterentwicklung der Sicherheitsfeatures für die Datenbank, auch mit der neuen Oracle Datenbank 12c Release 2, kontinuierlich fort. Hierbei geht es nicht um große, neue Optionen, sondern vielmehr um viele kleine Sicherheitsfunktionalitäten, welche es dem Benutzer einfacher machen, Sicherheitsmaßnahmen wirkungsvoll umzusetzen. Im Wesentlichen handelt es sich bei den Neuerungen um Funktionalitäten aus dem Bereich Zugriffsschutz.

1 EIN NEUER EVOLUTIONSSCHRITT IN DER DATENBANKVERSCHLÜSSELUNG

Das weltweit am häufigsten eingesetzte Sicherheitsfeature der Oracle-Datenbank ist definitiv die Transparent Data Encryption (TDE). Eingeführt wurde diese Verschlüsselungstechnologie bereits 2005 mit der Datenbankversion 10.2. Seitdem wurde sie von Version zu Version stetig erweitert und der Funktionsumfang vergrößert.

Mit der Version 12.2 lässt sich nun auch die Datenbank komplett verschlüsseln (Fully Encrypted Database). Bisher war es lediglich möglich benutzerdefinierte Tablespaces verschlüsselt anzulegen. Oracle-definierte Tablespaces, wie SYSTEM, SYSAUX, TEMP oder das UNDO, konnten nicht verschlüsselt werden. Die Konsequenz war, dass hochprivilegierte Benutzer wie ein Betriebssystem- oder ein Storage-Administrator durch eine einfache Linux-Strings-Suche nach Data-Dictionary-Informationen im SYSTEM-Tablespace

beziehungsweise in dessen Datenbankdateien suchen konnten – ohne sich an der Datenbank anmelden zu müssen. Die Möglichkeit jetzt alle Tablespaces inklusive der Oracle-definierten Tablespaces zu verschlüsseln, ist eine hervorragende Neuerung, wenn bedacht wird, dass im Data Dictionary neben allen Benutzernamen auch die Hashwerte der Kennwörter zu finden sind.

Es stellt sich nun noch die Frage, wie eine bestehende Datenbank nachträglich einfacher verschlüsselt werden kann. Auch hier wartet die Datenbank mit neuen Funktionen auf: Zwei neue Möglichkeiten – nämlich Online Conversion und Offline Conversion – zum nachträglichen Verschlüsseln der Datenbank werden zur Verfügung gestellt.

Mit TDE Live Conversion lassen sich auch bereits bestehende Datenbanken einfacher „nachverschlüsseln". Bisher war dies nur durch das Umkopieren von Daten aus einem nicht verschlüsselten in einen verschlüsselten Tablespace möglich – entweder durch Online Redefinition oder Data Pump Import/Export. Für beide Varianten musste gegebenenfalls neben einer Applikations-Downtime der doppelte Plattenplatz vorgehalten werden.

Um einen Tablespace, beziehungsweise dessen Datenbankdatei(en), während des Betriebs nachträglich zu verschlüsseln (Online Conversion), reicht jetzt ein einziger Datenbankbefehl aus. Bei der Online Conversion wird intern zu jeder dem Tablespace zugeordneten Datenbankdatei eine korrespondierende neue, verschlüsselte Datenbankdatei angelegt. Danach werden jeweils die Inhalte der originalen

Datenbankdateien – im laufenden Betrieb – in die verschlüsselten Datenbankdateien übertragen. Wenn alles erfolgreich war, werden die originalen Datenbankdateien automatisch gelöscht und somit der Platz wieder freigegeben:

```
ALTER TABLESPACE users ENCRYPTION ONLINE USING 'AES192'
       ENCRYPT FILE_NAME_CONVERT = ('users.dbf', 'users_enc.dbf');
```

Das Ganze funktioniert zwar ohne Datenbank-Downtime, aber der doppelte Plattenplatz der Datenbankdateien wird trotzdem benötigt.

Alternativ dazu gibt es eine platzsparende Variante mit der Offline Conversion. Sie ermöglicht das Konvertieren eines nicht verschlüsselten Tablespaces in einen verschlüsselten Tablespace, ohne zusätzlichen Plattenplatz zu benötigen. Eine partielle Downtime der Applikation muss hierbei eingeplant werden, da die zu konvertierenden Tablespaces OFFLINE sein müssen:

```
ALTER TABLESPACE users OFFLINE NORMAL;
ALTER DATABASE DATAFILE '…/users.dbf' ENCRYPT;
ALTER TABLESPACE users ONLINE;
```

Momentan kann man die beiden Methoden nicht miteinander kombinieren. Dafür wurde aber für die Offline-Conversion-Variante ein Backport für die Datenbankversionen 11.2.0.4 und 12.1.0.2 zur Verfügung gestellt. Dieser Backport steht seit dem Juli 2016 Bundle-Patch (12.1.0.2.160719 beziehungsweise 11.2.0.4.160719) zur Verfügung. Die Offline Conversion arbeitet

momentan ausschließlich mit dem Oracle-Standard-Encryption-Algorithmus AES 128.

Damit beim Anlegen eines neuen Tablespaces nicht vergessen wird die Verschlüsselung zu nutzen, gibt es einen neuen Datenbankparameter: ENCRYPT_NEW_TABLESPACES sorgt nun dafür, dass bei entsprechender Einstellung alle neuen Tablespaces ohne zusätzliche Storageklausel automatisch verschlüsselt werden.

Es existieren dabei drei Einstellungsmöglichkeiten:

```
ENCRYPT_NEW_TABLESPACES = CLOUD_ONLY | ALWAYS | DDL
```

Bei CLOUD_ONLY erkennt die Datenbank automatisch, dass sie sich in der Oracle Public Cloud befindet und verschlüsselt neue Tablespaces mit AES 128, ohne expliziter Encryption-Storageklausel.

ALWAYS macht da keine Unterschiede: Hier ist es egal, ob sich die Datenbank in der Oracle Public Cloud befindet oder nicht. Neue Tablespaces werden immer mit AES 128 verschlüsselt.

DDL stellt das bisherige Verhalten sicher. Die explizite Angabe einer Encryption-Storageklausel ist notwendig, wenn verschlüsselt werden soll.

Eine weitere Erleichterung gibt es für das Management und Klonen verschlüsselter Pluggable und Non-Pluggable Databases.

2 VEREINFACHUNG DER SCHLÜSSELVERWALTUNG (KEY MANAGEMENT)

Bisher musste für das Klonen einer Pluggable Database der entsprechende Master Key, aus dem Keystore der Ursprung-Container-Datenbank, in eine PKCS12-Datei exportiert werden, um diesen dann wieder vorab in den Keystore der Ziel-Container-Datenbank zu importieren. Dieser doch eher umständliche Weg ist bei der Version 12.2 nicht mehr notwendig. Eine Erweiterung des Befehlssatzes macht es möglich.

Das Kommando CREATE PLUGGABLE DATABASE wurde durch die Klausel KEYSTORE IDENTIFIED BY "kennwort" erweitert. Diese Klausel ermöglicht das automatische Exportieren und Importieren des Master Keys aus dem Quell-Keystore in den Ziel-Keystore ohne weitere Zwischenschritte.

Allerdings wird hierbei noch ein Klartext-Kennwort bei IDENTIFIED BY "kennwort" verwendet. Bisher musste der Datenbankadministrator somit das Kennwort des Keystores für alle Key-Operationen kennen. Dies ist zumindest aus Sicherheitsgründen ungünstig, aber auch manchmal einfach nur lästig. Abhilfe schafft eine neue Funktionalität namens EXTERNAL STORE. Der neue External Store ist ein separater PKCS12-Keystore, in dem das Kennwort der Datenbank-Keystores gespeichert wird – ähnlich wie bei den seit langer Zeit bekannten und bei Kunden häufig eingesetzten Oracle Secure External Password Stores (SEPS).

Aus KEYSTORE IDENTIFIED BY "kennwort" wird KEYSTORE IDENTIFIED BY EXTERNAL STORE. Ermöglicht wird dies durch Verwendung eines neuen Datenbankparameters: EXTERNAL_KEYSTORE_CREDENTIAL_LOCATION.

Um diese Funktionalität nutzen zu können, sind nur folgende drei Schritte erforderlich.

1. Verzeichnis für den External Store anlegen.
2. Datenbankparameter setzen und Datenbank durchstarten:

```
ALTER SYSTEM SET external_keystore_credential_location='external store file-location' SCOPE=SPFILE;
```

3. External Store mit dem Schlüsselwort "TDE_WALLET" und dem Keystore-Kennwort anlegen:

```
ADMINISTER KEY MANAGEMENT ADD SECRET “kennwort” FOR CLIENT 'TDE_WALLET' TO LOCAL AUTO_LOGIN KEYSTORE 'external store file-location';
```

Fertig.

Ab jetzt braucht der Datenbankadministrator kein Kennwort zur Verwaltung des Master Keys mehr. Somit sind die folgenden vereinfachten Kommandos möglich:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY EXTERNAL STORE;
ADMINISTER KEY MANAGEMENT SET KEY USING TAG 'cdb1:root:v.02' IDENTIFIED BY EXTERNAL STORE WITH BACKUP USING 'backup:v.01' CONTAINER = CURRENT;
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE IDENTIFIED BY EXTERNAL STORE;
```

Aber was kann man machen, wenn ein Auto Login Wallet verwendet wird? Bisher konnten Operationen, wie zum Beispiel das Erneuern eines Master Keys, nur durchgeführt werden, wenn kein Auto Login Wallet verwendet wurde. Die bisherige Vorgehensweise in diesem Fall war doch eher umständlich, wie folgendes Beispiel zeigt:

1.1.1 Löschen des Auto Login Wallets "cwallet.sso".

1.1.2 Schließen des Auto Login Wallets:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE CLOSE;
```

1.1.3 Öffnen des Standard Wallets:

```
ADMINISTER KEY MANAGEMENT SET KEYSTORE OPEN IDENTIFIED BY "kennwort";
```

1.1.4 Nach den Key-Management-Aktivitäten musste das Auto Login Wallet wieder erstellt werden.

In der Version 12.2 wurde auch hieran gedacht. Die Klausel FORCE KEYSTORE ermöglicht das Key Management bei gleichzeitiger Nutzung des Auto Login Wallets:

```
ADMINISTER KEY MANAGEMENT SET KEY USING TAG 'cdb1:root:v.03' FORCE
KEYSTORE IDENTIFIED BY EXTERNAL STORE WITH BACKUP CONTAINER = CURRENT;
```

Dieses Beispiel zeigt ein Rekeying ohne Angabe eines Kennworts und bei gleichzeitiger Nutzung eines Auto Login Wallets. Und das Ganze funktioniert mit nur einem einzigen Befehl.

Durch all diese Neuerungen aus dem Bereich Datenbankverschlüsselung mittels Oracle Transparent Data Security (TDE), wird die Verwaltung einer verschlüsselten Oracle-Datenbank nicht nur einfacher, sondern auch wesentlich sicherer.

3 PLUGGABLE-DATABASE-ISOLIERUNG

Ein ganz anderes neues Sicherheitsfeature heißt PDB LOCKDOWN PROFILE. Die Frage danach, in wie weit einzelne Pluggable Databases voneinander isoliert betrieben werden können, konnte bisher nur teilweise befriedigend beantwortet werden. Hier spielt sowohl die physische als auch die logische Trennung zwischen den Pluggable Databases eine entscheidende Rolle. Keine Pluggable Database darf eine andere „gefährden". Dazu ist es zwingend notwendig, entsprechend potenziell gefährliche Datenbankbefehle und -funktionalitäten, welche die Isolierung beeinträchtigen können, einzuschränken. Gemeint sind hier Datenbankbefehle wie ALTER SYSTEM, ALTER PLUGGABLE DATABASE, ALTER SESSION und ALTER DATABASE sowie Datenbank-Features wie die XMLDB. Letzteres ermöglicht zum Beispiel Zugänge über HTTP/S und FTP auf dem Datenbankserver und könnte somit ein potenzielles Sicherheitsrisiko aller dort betriebenen Pluggable Databases darstellen. Diese doch sehr mächtigen Befehle und Features konnten in der Vergangenheit nicht weiter individuell eingeschränkt werden. Genau das erledigen jetzt die sogenannten PDB Lockdown Profiles.

Es besteht nun die Möglichkeit, den Umfang bestimmter Datenbankbefehle und Features innerhalb einer Pluggable Database besser zu kontrollieren und zu steuern. Soll zum Beispiel verhindert werden, dass der Administrator einer Pluggable Database den CPU_COUNT seiner Pluggable Database über einen Maximalwert vergrößert, formuliert man folgende Regel:

```
ALTER LOCKDOWN PROFILE profile_name DISABLE STATEMENT = ('ALTER SYSTEM')
CLAUSE = ('SET') OPTION = ('CPU_COUNT') MAXVALUE = '8';
```

Soll eine Datenbankfunktionalität innerhalb einer Pluggable Database komplett ausgeschaltet werden, um zum Beispiel den Zugang über HTTP/S oder FTP erst gar nicht zu ermöglichen, kann dies durch diesen Befehl geschehen:

```
ALTER LOCKDOWN PROFILE profile_name DISABLE FEATURE = ('XDB_PROTOCOLS');
```

Diese Beispiele sind nur ein kleiner Teil der Möglichkeiten, die das Feature PDB Lockdown Profile bietet. Mit PDB Lockdown Profile können sehr feingranulare Sicherheitsregeln erstellt werden, welche eine weitgehende Isolierung einzelner Pluggable Databases ermöglichen.

Übrigens setzt Oracle das Lockdown Profile bereits seit Längerem selbst für den sicheren Betrieb des Oracle Exadata Express Cloud Service ein. Nur so kann gewährleistet werden, dass sich Pluggable Databases (Mandanten) nicht untereinander beeinflussen können.

Eine weitere neue Isolierungsmöglichkeit für Pluggable Databases kann über den Datenbankparameter PDB_OS_CREDENTIAL erreicht werden.

Auch in einer Multitenant-Database-Umgebung muss es möglich sein, Betriebssystemfunktionalitäten wie zum Beispiel Betriebssystembefehle oder Bibliotheken verwenden zu können. Dies geschieht im Allgemeinen über die Oracle-Listener-Funktionalität External Procedure Calls. Die Folge davon ist, dass alle Aktivitäten mit der Berechtigung des Listener-Prozess-Besitzers ausgeführt werden. In den meisten Installationen ist das der Oracle-Softwarebesitzer (Oracle). Das ist ein sehr großes Sicherheitsrisiko!

Um dieses neue Sicherheitsfeature nutzen zu können, muss zuerst für die entsprechenden Pluggable Databases innerhalb der Container Database ein sogenanntes „Credential-Objekt“ erstellt werden. Dieses Objekt beinhaltet neben einem eindeutigen Namen (zum Beispiel der PDB-Name), den für die External Procedure Calls zu verwendenden Betriebssystembenutzer, inklusive dessen Kennwort:

```
DBMS_CREDENTIAL.CREATE_CREDENTIAL (
 credential_name => 'PDB1_OS_USER',
 username        => 'os_admin',
 password        => 'password');
```

Danach kann diese Credential-Information in der Pluggable Database als Einstellung über Parameter PDB_OS_CREDENTIAL mithilfe eines ALTER SYSTEM-Befehls gesetzt werden:

```
ALTER SYSTEM SET PDB_OS_CREDENTIAL = PDB1_OS_USER SCOPE = SPFILE;
```

Betrachtet man nur die Sicherheitsaspekte der Oracle-Datenbank Version 12.2 kann man wichtige Neuerungen finden. Ich habe mich hier nur auf einige wenige Teilbereiche konzentriert. Einen Gesamtüberblick ist in der Dokumentation zu Oracle Database 12.2 im Bereich „Oracle Database 12c Release 2 (12.2) New Features“ zu finden.

ULRIKE SCHWINN

JSON-Daten in der Oracle-Datenbank

„All your Data“ in der Oracle-Datenbank gilt immer noch, auch und gerade in der aktuellen Datenbankversion Oracle Database 12c Release 2: sei es im Bereich XML, bei Texten mit linguistischen Suchanforderungen oder auch für Geoinformationen (Spatial) und im semantischen Netzwerkumfeld, um nur einige Beispiele zu nennen.

JSON-Daten werden allerdings häufig in NoSQL oder anderen speziellen Datenbanken gespeichert beziehungsweise damit in Verbindung gebracht. Diese erlauben zwar die Speicherung und den Zugriff der Daten, weisen aber kein vergleichbares Konsistenzmodell, Transaktionsmodell und andere Standardfunktionalitäten von relationalen Datenbanken auf.

Ab 12c Release 1 ist es möglich, performant und einfach auf JSON-Daten in der Oracle-Datenbank zuzugreifen. Die Verwendung ist dabei ganz einfach: Man definiert eine Datenbankspalte mit einem beliebigen Datentyp für Textstrings (wie zum Beispiel VARCHAR2 oder CLOB). Mit der Bedingung IS JSON kann dann zusätzlich der Inhalt validiert werden – auf Wohlgeformtheit oder auf die Art der Syntaxverwendung (STRICT oder LAX). Die Zugriffe erfolgen dann mit Standardmitteln und neu eingeführten SQL/JSON-Funktionen. Liegen die JSON-Daten in einem Dump-Format (DMP) außerhalb der Datenbank vor, kann sogar über die Funktion EXTERNAL TABLE zugegriffen werden. Es ist keine zusätzliche Installation von Software erforderlich; eine JSON-

Unterstützung ist in allen Softwareinstallation und Cloud-Editionen der Datenbank sofort verfügbar. Mehr über die Basisfunktionen in der Oracle-Datenbank können Sie im Community-Beitrag „JSON in der Datenbank mit Basisfunktionen“ nachlesen.

In Oracle 12.2 gibt es jetzt auch Lösungen zu folgenden Fragestellungen: Wie kann man beispielsweise JSON mit Mitteln der Datenbank generieren? Wie lassen sich JSON-Inhalte in der Datenbank mit PL/SQL-Mitteln manipulieren? Wie kann man schnell und einfach relationale Views erzeugen? Wie lässt sich umfassend in JSON suchen? Für hohe Performance-Anforderungen ist zusätzlich zu den bestehenden In-Memory-Zugriffen ein spezielles optimiertes JSON-Dokument-Handling mit In-Memory in 12.2 implementiert worden.

1 JSON-DATEN GENERIEREN MIT JSON/SQL-OPERATOREN

Kennt man die entsprechenden Werkzeuge, lassen sich auch schon vor Oracle 12.2 JSON-Daten aus der Datenbank generieren. Arbeitet man beispielsweise mit Oracle Application Express (APEX) kann man JSON-Dokumente mit der Funktion APEX_JSON erzeugen. Eine andere Möglichkeit bietet das Linemode-Werkzeug SQLcl. Mit dem Format-Kommando SET SQLFORMAT JSON wird beispielsweise automatisch eine Ausgabe im JSON-Format angezeigt. In 12.2 gibt es nun eine dritte Möglichkeit unter Verwendung reiner SQL-Mittel und den neuen SQL/JSON-Operatoren wie JSON_ARRAY, JSON_ARRAYAGG, JSON_OBJECT und JSON_OBJECTAGG. Mit JSON_ARRAY wird beispielsweise ein zugehöriges JSON-Feld ausgegeben:

```
SQL> select json_array(empno, ename, mgr, deptno) emp
     from scott.emp where deptno=10;

EMP
----------------------
[7782,"CLARK",7839,10]
[7839,"KING",10]
[7934,"MILLER",7782,10]
```

JSON_OBJECT erzeugt hingegen zeilenweise die entsprechenden Key-Value-Paare aus jeder referenzierten Spalte:

```
SQL> select json_object('empid' is empno,'empname' is ename, 'Managerid'
is mgr,
    'Deptid' is deptno) emp
     from scott.emp where deptno=10;

EMP
--------------------------------------------------------------------------
-----
{"empid":7782,"empname":"CLARK","Managerid":7839,"Deptid":10}
{"empid":7839,"empname":"KING","Managerid":null,"Deptid":10}
{"empid":7934,"empname":"MILLER","Managerid":7782,"Deptid":10}
```

Interessant ist auch die Möglichkeit ein Key-Value-Paar bezüglich einer bestimmten Spalte zu aggregieren. Um die reine Funktionsweise aufzuzeigen, wird im nächsten Beispiel weiterhin die EMP-Tabelle verwendet. Es gibt 14 Angestellte, mit 6 verschiedenen Managern. Pro Manager sollen nun die entsprechenden Angestellten mit Berufsbezeichnung als Key-Value-Paar ausgegeben werden:

```
SQL> select json_objectagg(ename, job) employees_per_manager from emp
     group by mgr;

EMPLOYEES
--------------------------------------------------------------------------------
{"SCOTT":"ANALYST","FORD":"ANALYST"}
{"ALLEN":"SALESMAN","JAMES":"CLERK","TURNER":"SALESMAN","MARTIN":"SALESMAN","WARD
":"SALESMAN"}
{"MILLER":"CLERK"}
{"ADAMS":"CLERK"}
{"JONES":"MANAGER","CLARK":"MANAGER","BLAKE":"MANAGER"}
{"SMITH":"CLERK"}
{"KING":"PRESIDENT"}
```

Natürlich lassen sich diese Operatoren miteinander kombinieren und ergänzen und somit wesentlich komplexere Ausgaben erzeugen.

2 PL/SQL-INTEGRATION MIT JSON-DATENSTRUKTUREN

Weitere Funktionen über PL/SQL bietet eine neue PL/SQL-API in Oracle 12.2. Damit ist es möglich Key-Value-Paare hinzuzufügen, zu verändern oder zu löschen. Man kann durch ein JSON-Dokument navigieren, vergleichbar mit dem XML DOM (Document Object Model) Handling. Neue Objekttypen wie JSON_ELEMENT_T, JSON_OBJECT_T und JSON_ARRAY_T mit ihren zugehörigen Funktionen können bei der Umsetzung hilfreich sein. Der Basistyp JSON_ELEMENT_T liefert beispielsweise eine PARSE-Funktion um die interne Repräsentation für die weitere Verarbeitung zur Verfügung zu stellen. Darüber hinaus gibt es Funktionen und Methoden, um ein einfaches Error-Handling zu

gewährleisten oder einfach nur um Put /Get, Clone und Rename/ Remove Funktionalität zur Verfügung zu stellen. Da die Erläuterung der einzelnen Methoden und Konstruktoren zu weit führen würde, soll ein kurzes, einfaches Beispiel die generelle Vorgehensweise demonstrieren. Verwendet wird dazu die Tabelle JSON_TAB mit einer Spalte JSON_DOCUMENT, in der die JSON-Daten mit Bestellinformationen gespeichert sind. Unter anderem existiert ein Key "User", der von 'CJOHNSON' in den Wert 'USCHWINN' geändert werden soll:

```
declare
cursor getDocuments is
     select JSON_DOCUMENT
     from JSON_TAB j
     where json_value(json_document,'$.User') = 'CJOHNSON' for UP-
DATE;
     V_RESULT            JSON_ELEMENT_T;
     V_DOCUMENT_OBJECT   JSON_OBJECT_T;
     V_NEW_DOCUMENT      VARCHAR2(4000);
begin
 --Dokumente einlesen und interne Repräsentation aufsetzen
    for doc in getdocuments loop
    V_RESULT := JSON_ELEMENT_T.parse(doc.json_DOCUMENT);
    V_DOCUMENT_OBJECT := treat(V_RESULT as JSON_OBJECT_T);

    if (V_Document_OBJECT.get_String('User') = 'CJOHNSON')
       then   V_document_object.put('User','USCHWINN');
    end if;
```

```
-- Umwandlung in eine Zeichenkette
   V_NEW_DOCUMENT := V_DOCUMENT_OBJECT.to_string();

-- tatsächliche Änderung der Spalte
   update JSON_TAB set JSON_DOCUMENT = V_NEW_DOCUMENT
   where current of getDocuments;
   commit;
end loop;
end;
```

Eine genaue Erläuterung findet sich im *Database PL/SQL Packages and Types Reference* im Kapitel „JSON Data Structures".

3 EINE NEUE STRUKTUR FÜR DIE EINFACHE ERZEUGUNG VON VIEWS UND SCHNELLER TEXTSUCHE

Hat man mit großen und unübersichtlichen JSON-Daten zu tun, stellt sich schnell die Frage, wie man darin performant nach einem Wert oder Eintrag suchen kann. Auch die Möglichkeit, eine relationale Sichtweise auf JSON-Dokumente zur Verfügung zu stellen, ist eine übliche Fragestellung. Die Basisfunktionen in Oracle 12.1 liefern schon erste Möglichkeiten diese Anforderungen zu lösen. Eine Vereinfachung und weitere Funktionen verspricht das neue Konzept in 12.2. Die Metadaten eines JSON-Dokuments können nun im Data Dictionary gespeichert werden. Optional wird ein zugehöriger Index zur Volltextsuche zur Verfügung gestellt. Die neue Struktur hat den treffenden Namen Data Guide. Damit ist es nun möglich schnell und einfach mit einem Befehl relationale Views zu erzeugen, eine performante Suche im Text

zu gestalten und sogar automatisch virtuelle Spalten zu einer View hinzuzufügen, sobald sich der Inhalt des JSON-Dokuments erweitert.

Im ersten Beispiel wird ein Data Guide erzeugt. Auch hier dient die Tabelle JSON_TAB als Grundlage. Je nach Verwendung der Syntax gibt es unterschiedliche Varianten um einen Data-Guide-Konstrukt zu erzeugen – beispielsweise mit oder ohne einen speziellen Oracle-Text-Index. Die einfachste Form der Syntax, die alle Defaulteinstellungen verwendet, führt zu einer automatischen Erzeugung eines Oracle-Text-Index. Die Funktion Oracle Text ist eine in die Datenbank integrierte Volltextrecherche, die in allen Datenbankeditionen enthalten ist und normalerweise ohne weitere Installation direkt zur Verfügung steht:

```
create search index JSON_TAB_GUIDE on JSON_TAB (JSON_DOCUMENT) for
json;
```

Kontrolliert man nach der Erstellung der Data-Guide-Struktur die View USER_OBJECTS, stellt man fest, dass eine ganze Reihe neuer zusätzlicher Objekte wie Indizes, Tabellen und LOBs erzeugt wurden. Genaue Informationen über die Syntax findet man übrigens im *Oracle Text Reference Guide*. So kann man beispielsweise auf das Erzeugen eines Textindex verzichten oder die Synchronisierung beeinflussen. Kennt man sich zusätzlich etwas mit Oracle-Text-Funktionen aus, kann man einige Parallelen in der Syntax und der weiteren Verwendung erkennen.

Wo liegen nun die Strukturen und was genau wird gespeichert? Die Strukturen sind beispielsweise in USER_JSON_DATAGUIDES zu finden oder lassen sich über die neue Funktion DBMS_JSON.GET_INDEX_DATAGUIDE ausgeben:

```
SQL> select dbms_json.get_index_dataguide (
        OWNER     => 'SCOTT',
        TABLENAME => 'JSON_TAB',
        JCOLNAME  => 'JSON_DOCUMENT',
        FORMAT    => dbms_json.format_hierarchical,
        PRETTY    => dbms_json.pretty) data_guide
 from dual;

DATA_GUIDE
--------------------------------------------------------------------------------
{
  "type" : "object",
  "properties" :
  {
    "User" :
    {
      "type" : "string",
      "o:length" : 8,
      "o:preferred_column_name" : "JSON_DOCUMENT$User"
    }, ....
```

Das Ganze sieht schon sehr vielversprechend aus: Es werden einzelne Elemente, Datentypen und mögliche Spaltennamen (hier JSON_DOCUMENT$User) ausgegeben. Das Package DBMS_JSON kann

dann im nächsten Schritt zur Erzeugung einer Standard View (hier: JSON_PO_VIEW) dienen:

```
begin
   dbms_json.create_view_on_path (
     viewName   => 'JSON_PO_VIEW',
     tableName  => 'JSON_TAB',
     jcolname   => 'JSON_DOCUMENT',
     path       => '$');
end;
```

```
SQL> desc JSON_PO_VIEW
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------
----
 ID                                        NOT NULL NUMBER
 JSON_DOCUMENT$User                                 VARCHAR2(8)
 JSON_DOCUMENT$PONumber                             NUMBER
 JSON_DOCUMENT$Reference                            VARCHAR2(32)
 …
```

Die Spaltennamen lassen sich anschließend mit der Prozedur DBMS_JSON.RENAME_COLUMN umbenennen.

Zum Abschluss soll noch die Verwendung des erzeugten Textindex demonstriert werden. Ein typisches Beispiel liefert eine Ad-hoc-Abfrage mit JSON_EXISTS, in der alle JSON-Dokumente mit einem Ländereintrag in der Auslieferungsadresse gesucht werden. Der Ausführungsplan bestätigt dabei die Verwendung des Textindex (siehe Operation DOMAIN INDEX):

```
SQL> select count(*) from JSON_TAB
     where json_exists(json_document,'$ShippingInstructions.Address.country');
```

Das letzte Beispiel zeigt eine Volltextsuche mit einem Oracle-Text-Operator. Die Aufgabe besteht darin, die Dokumente zu finden, bei denen der Wert von Requestor auf „Sarah“, „Sarath“ oder einem ähnlichen Namen lautet. Hier bietet sich der Einsatz des Oracle-Text-Operators FUZZY oder SOUNDEX an. Das Ergebnis sieht dann folgendermaßen aus:

```
SQL> select json_document from json_tab
     where json_textcontains(json_document, '$.Requestor','fuzzy(Sarat) ');
```

Möchte man mehr über die Möglichkeiten der Oracle-Text-Abfragen wissen, eignet sich die Lektüre des Oracle-Text-Handbuchs.

Wichtig zu wissen ist, dass diese implementierten JSON-Funktionalitäten nicht nur in allen Editionen und Cloud-Angeboten der Datenbank ab Oracle Database 12c verfügbar sind, sondern darüber hinaus beliebig mit anderen Technologien wie zum Beispiel Virtual Private Database (VPD) kombiniert werden können. Die Zugriffe sind dabei performant und über die üblichen Mechanismen der Ausführungspläne zu monitoren.

RALF DURBEN

Multitenant-Architektur mit Applikationscontainern

Oracle Multitenant in der Oracle-Datenbank 12c ist nicht nur ein neues Feature, sondern eine für Oracle völlig neue Architektur zum Betrieb von Datenbanken, die auf vielen Gebieten Veränderungen und große Vorteile bewirkt. Dabei trennt Oracle die strikte Eins-zu-eins-Beziehung von Datenbank und Instanz und erlaubt den Betrieb einer globalen Instanz für mehrere Datenbanken, auch Pluggable Database oder kurz PDB genannt. Die PDB wird in eine sogenannte Container-Datenbank (oder kurz CDB) „eingeklinkt" und mit der globalen Instanz betrieben. Je nach Bedarf und Anwendung kann eine Datenbank von einer Container-Datenbank zu einer anderen umziehen. Weitere Details zu Oracle Multitenant können Sie im Oracle *Dojo Nr. 7* nachlesen, welches Sie unter http://tinyurl.com/dojoonline beziehen können.

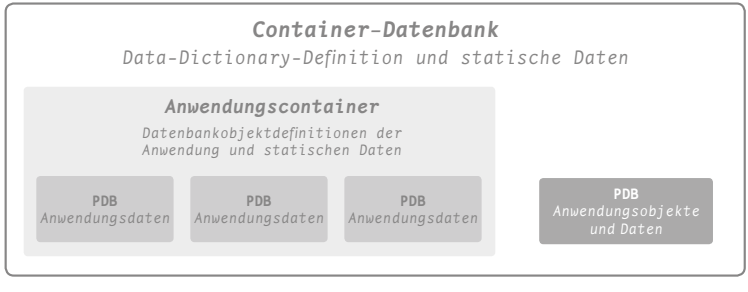
Oftmals werden für eine Applikation mehrere Datenbanken benötigt, sei es um neben der Produktions-Datenbank auch Test- und Integrations-Datenbanken zu betreiben oder auch, wenn eine Anwendung mehrfach, zum Beispiel getrennt für verschiedene Filialen, Regionen oder Mandanten, betrieben wird. In allen diesen Fällen benötigen diese Datenbanken das gleiche Datenmodell, und der traditionelle Ansatz zur Erstellung einer solchen Datenbank besteht im Ausführen vorgegebener Skripte in jeder einzelnen Datenbank. Die Multitenant-Architektur beinhaltet eine Trennung der Speicherung von Data-Dictionary-Definitionen und -Inhalten, wobei ein statischer Teil auf CDB-Ebene und der dynamische Teil auf PDB-Ebene gespeichert wird. Das Data Dictionary ist das Datenmodell der Oracle-Datenbank;

man kann es mit einem Datenmodell von Anwendungen gut vergleichen. Also kann man auch auf der Ebene der Anwendung diese Trennung durchführen. Es gibt für jede Anwendung:

- das Datenmodell, also die Definition von Datenbankobjekten
- statische Daten, wie zum Beispiel ein Katalog von Postleitzahlen
- dynamische Daten, wie zum Beispiel Buchungsdaten

Wenn für eine Anwendung mehrere Datenbanken betrieben werden, stellt sich schnell die Frage, ob man nicht – wie beim Data Dictionary auch – eine einmalige, zentrale Speicherung all dessen durchführen kann, was in allen PDBs gleich genutzt wird. Genau für dieses Ziel gibt es den Applikationscontainer. Das folgende Schaubild zeigt diesen Zusammenhang.

Abb.1: Applikationscontainer und die Multitenant-Architektur



Da eine CDB (ab Oracle Database 12.2) bis zu 4096 PDBs betreiben kann, werden in aller Regel auch mehrere verschiedene Anwendungen in mehreren PDBs betrieben. Daher können Sie mehrere Applikationscontainer in einer CDB betreiben. Jeder Applikationscontainer ist sowohl eine PDB (in der Haupt-CDB), als auch CDB (für die Anwendungs-PDBs). Auch für den Applikationscontainer kann eine Seed-Datenbank vorbereitet werden, um die Erstellung neuer PDBs, die automatisch für die gegebene Anwendung vorbereitet sind, zu beschleunigen. Die folgenden Abschnitte zeigen den Umgang mit den neuen Applikationscontainern.

1 ERSTELLEN EINES APPLIKATIONSCONTAINERS

Um einen Applikationscontainer zu erstellen, melden Sie sich an die CDB an. Sie benötigen dazu das Privileg CREATE PLUGGABLE DATABASE. Dann verwenden Sie das Kommando CREATE PLUGGABLE DATABASE ergänzt um die neue Klausel AS APPLICATION CONTAINER. Hier ein Beispiel:

```
CREATE PLUGGABLE DATABASE myapp_ac AS APPLICATION CONTAINER
ADMIN USER myapp_ac_adm IDENTIFIED BY manager
FILE_NAME_CONVERT=('pdbseed','myapp');
```

Sie öffnen dann den neuen Applikationscontainer wie üblich mit einem ALTER PLUGGABLE DATABASE-Kommando:

```
ALTER PLUGGABLE DATABASE myapp_ac OPEN;
```

Sie können sich nun an den neuen Applikationscontainer anmelden, um weitere Aktionen durchzuführen, die in den nächsten Abschnitten beschrieben werden.

2 ERSTELLEN EINER SEED-DATENBANK IN EINEM APPLIKATIONSCONTAINER

Mit einer Seed-Datenbank erstellen Sie durch schnelles Cloning die später angeforderten Anwendungs-PDBs. Sie ist also eine Art Template für weitere PDBs. Um eine Seed-Datenbank zu erstellen, melden Sie sich an den Applikationscontainer an. Dann führen Sie das Kommando CREATE PLUGGABLE DATABASE AS SEED aus. Beachten Sie dabei, dass in einem Applikationscontainer nur eine Seed-Datenbank erstellt werden kann:

```
CREATE PLUGGABLE DATABASE AS SEED
ADMIN USER pdbadmin IDENTIFIED BY manager;
```

Auch diese PDB müssen Sie zunächst öffnen mit:

```
ALTER PLUGGABLE DATABASE myapp_ac$SEED OPEN;
```

In der Seed-Datenbank können Sie Datenbankobjekte erstellen, die später beim Cloning auch in der PDB lokal gespeichert sind. Dabei wird aber ein Kopieren vorgenommen. Spätere Änderungen in der Seed-Datenbank haben keinen Effekt auf bereits existierende Anwendungs-PDBs.

Des Weiteren stellen Sie die PDB-spezifischen Instanzparameter ein. An dieser Stelle verweise ich auf eine weitere Funktionalität: PDB Lockdown Profiles. Damit können Sie zum Beispiel Einfluss nehmen auf die Datenbankoptionen und -features, die in der PDB nutzbar sind.

3 ERSTELLEN VON ANWENDUNGS-PDBS

Wenn Sie den Applikationscontainer inklusive Seed-Datenbank vorbereitet haben, können Sie nun die Anwendungs-PDBs erstellen. Dazu melden Sie sich an den Applikationscontainer an und erstellen die PDB zum Beispiel mit:

```
CREATE PLUGGABLE DATABASE pdb_app2
ADMIN USER pdbadmin IDENTIFIED BY manager DEFAULT TABLESPACE data;
```

Dabei wird die Seed-Datenbank geklont. Beachten Sie, dass nachträgliche Änderungen an der Seed-Datenbank nicht für die bereits erstellten Anwendungs-PDBs gelten.

Die Verflechtungen im Bereich Applikationscontainer, also ob eine PDB in einem Applikationscontainer liegt und wenn ja, in welchem, können Sie der View V$CONTAINERS entnehmen:

```
col name format a25
col app_root format a8
col app_pdb format a7
col app_seed format a8
col app_r_id format 99999999
set pagesize 100
```

```
SELECT con_id,name,
       application_root as app_root,
       application_pdb as app_pdb,
       application_seed as app_seed,
       application_root_con_id as app_r_id
FROM v$containers;

   CON_ID NAME                      APP_ROOT APP_PDB APP_SEED  APP_R_ID
--------- ------------------------- -------- ------- -------- ---------
        1 CDB$ROOT                       NO      NO       NO
        2 PDB$SEED                       NO      NO       NO
        3 PDB1                           NO      NO       NO
        4 DEMOS                          NO      NO       NO
        5 myapp_ac$SEED                  NO     YES      YES         7
        6 PDB_APP1                       NO     YES       NO         7
        7 myapp_ac                      YES      NO       NO
        8 PDB_APP2                       NO     YES       NO         7
```

4 ERSTELLEN DES ZENTRAL GESPEICHERTEN DATENMODELLS

Bislang wurde gezeigt, wie der Applikationscontainer und die PDBs erstellt werden. Jetzt müssen diese mit Inhalt gefüllt werden. Sie können im Applikationscontainer Datenbankobjekte speichern, die inhaltlich oder auf der Ebene der Metadaten von den Anwendungs-PDBs referenziert werden. Aus der Sicht der PDB sind dies lokale Datenbankobjekte, die aber nur einmalig gespeichert sind.

Applikationscontainer unterstützen dabei auch eine Versionierung von Anwendungen. Dazu müssen spezielle Rahmenbedingungen geschaffen werden, die unten in Beispielen gezeigt werden. Ein

einfaches Erstellen einer Tabelle in einem Applikationscontainer reicht also nicht, um diese allen Anwendungs-PDBs zugänglich zu machen. Vielmehr muss im Applikationscontainer in einer speziellen Syntax eine Anwendung mit einer angegebenen Version installiert werden. Entsprechend können Sie später diese Anwendung patchen und upgraden. Am folgenden kleinen Beispiel erkennen Sie das Prinzip:

```
ALTER PLUGGABLE DATABASE APPLICATION mywdg_app BEGIN INSTALL '1.0';
CREATE USER wdg IDENTIFIED BY wdg CONTAINER=ALL;
GRANT CREATE SESSION, DBA TO wdg;
CONNECT wdg/wdg@PDB_RALF_APPCON
CREATE TABLE wdg.plz_verzeichnis SHARING=DATA (PLZ number,ort varchar2(100));
INSERT INTO wdg.plz_verzeichnis VALUES (59556 , 'Lippstadt');
COMMIT;
ALTER PLUGGABLE DATABASE APPLICATION mywdg_app END INSTALL '1.0';
```

Sie beginnen also eine Installation einer Anwendung und erstellen einen Datenbankbenutzer mit der Option CONTAINER=ALL. Unter diesem Benutzer – oder auch mehreren – erstellen Sie dann die Datenbankobjekte. Es gibt dabei drei Varianten der gemeinsamen Nutzung von Objekten:

- METADATA
  Die PDBs sehen nur die Definition, also die Metadaten von Datenbankobjekten, also zum Beispiel die Definition einer Tabelle mit Spalten und Datentypen. Die Daten selbst werden allein in der Anwendungs-PDB gespeichert.

- DATA
  Die Daten werden im Applikationscontainer gespeichert und stehen in der Anwendungs-PDB nur lesend zur Verfügung, wie zum Beispiel eine Tabelle mit den Postleitzahlen von Deutschland.
- EXTENDED DATA
  Vorgegebene Daten sind im Applikationscontainer nur lesend gespeichert, aber die Anwendungs-PDB kann weitere Daten lokal speichern. Das könnte eine Tabelle mit global gültigen Daten sein, die in der Anwendungs-PDB mit lokalen Daten erweitert wird.

Mit dem Initialisierungsparameter DEFAULT_SHARING können Sie auf der Ebene des Applikationscontainers einen Default-Modus setzen. Standardmäßig ist dieser Parameter auf METADATA gesetzt.

Sie können aber auch auf Objektebene eine individuelle Einstellung vornehmen, wie im obigen Beispiel. Die Tabelle PLZ_VERZEICHNIS wird inklusive der Daten im Applikationscontainer gespeichert und soll in den Applikations-PDBs nur lesend zur Verfügung stehen.

Nachdem die Anwendung im Applikationscontainer installiert ist, müssen Sie in den Anwendungscontainern einen SYNC durchführen mit:

```
ALTER PLUGGABLE DATABASE APPLICATION mywdg_app SYNC;
```

Erst dann sind die neuesten Definitionen verfügbar. Damit können also auch neue Versionen von Anwendungen im Applikationscontainer vorbereitet werden, ohne dass die Anwendungs-PDBs davon betroffen sind. Erst nachdem die neue Version getestet und freigegeben ist, schalten Sie sie mit dem SYNC-Kommando „scharf". Im folgenden Beispiel wird eine neue Version der Anwendung mit einer neuen PL/SQL-Funktion erstellt:

Im Applikationscontainer:

```
ALTER PLUGGABLE DATABASE APPLICATION mywdg_app BEGIN UPGRADE '1.0' to '2.0';

CREATE OR REPLACE FUNCTION wdg.plz_check (p_plz number,p_ort varchar2)
RETURN number
AS
 valid number:=0;
 n     number;
BEGIN
 select count(*) into n
  from  wdg.plz_verzeichnis
 where  plz=p_plz and ort=p_ort;
 if n>0 then valid:=1; end if;
 return valid;
END;
/

ALTER PLUGGABLE DATABASE APPLICATION mywdg_app END UPGRADE to '2.0';
```

Das Update wird in der Anwendungs-PDB einfach aktiviert mit:

```
ALTER PLUGGABLE DATABASE APPLICATION mywdg_app SYNC;
```

Mit den neuen Applikationscontainern ist es also möglich, Anwendungsobjekte und Anwendungslogik, die mit Datenbankobjekten wie zum Beispiel PL/SQL Stored Procedures implementiert sind, einmalig und zentral zu speichern. Damit verringert sich der Speicher- und Wartungsaufwand, wenn die Anwendung in mehreren Datenbanken betrieben wird. Gerade für den mandantenorientierten Einsatz von Oracle Multitenant ist dieses Feature interessant, wenn das gleiche Datenmodell für jeden Mandanten in einer eigenen Datenbank verwendet wird.

KARIN PATENGE

Neue Möglichkeiten für das Management und die Nutzung von Geodaten

Dass die Oracle-Datenbank schon seit vielen Versionen auch den Orts- beziehungsweise Raumbezug von Daten versteht und auswerten kann, ist bekannt. Die dafür notwendige Funktionalität wird in jeder Edition über das Feature Oracle Locator bereitgestellt. Der nachfolgende Artikel beschreibt eine Auswahl an interessanten Neuerungen, die helfen, aktuelle Anforderungen an Anwendungen mit Fokus auf Geodaten umzusetzen. So bezieht sich der erste Teil auf das Thema „Location Data Enrichment", also das Anreichern von Unternehmensdaten um öffentlich verfügbare, teilweise hierarchisch organisierte Datensätze, zum Beispiel aus Open-Data-Initiativen wie GeoNames.org. Der Abschnitt „Räumliche Indexierung und Partitionierung" liefert ein Beispiel für Performanzoptimierung insbesondere bei großen Punktdatenbeständen, wie sie unter anderem geokodierte Adressen darstellen. Im letzten Abschnitt zu „GeoJSON" zeigen wir eine weitere praktische Anwendung der mit 12c eingeführten JSON-Unterstützung für Geodatenmanagement und -analyse.

Wer mehr Informationen und nützliche Hinweise zum Umgang mit Geodaten in der Oracle-Datenbank sucht, findet diese unter anderem in unserem deutschsprachigen Blog *Geodaten für Alle: Oracle Spatial. Tipps, Tricks, Best Practices und Aktuelles zu Oracle Locator und Oracle Spatial and Graph* auf http://oracle-spatial.blogspot.com sowie ganz ausführlich im *Oracle Spatial Developer's Guide* (Version 12c Release 2).

1 LOCATION DATA ENRICHMENT FÜR MEHR WERTSCHÖPFUNG AUS VORHANDENEN DATEN

Stellen Sie sich vor, Sie haben als Informationsschnipsel in einem Text oder Datensatz das Wort „Märchenbrunnen“. Aus dem Kontext schließen Sie, dass es sich hierbei um einen Ort handeln kann. Sie wissen aber nicht, wo sich dieser Ort befindet.

Die in 12.2 neue Funktionalität Location Data Enrichment beantwortet nicht nur diese Frage – das kann je nach Vorhandensein von Referenzdaten zu sogenannten Points of Interest auch eine Geokodierung (Umrechnung von Adressdaten in Geokoordinaten) –, sondern auch, wie dieser Ort in eine administrative Hierarchie eingebettet ist und ob es zusätzliche Informationen dafür gibt.

Technische Grundlage für Location Data Enrichment sind ein Referenzdatenbestand sowie die neue Funktion GEO_SEARCH im PL/SQL Package SDO_UTIL des Standard-Nutzers MDSYS der Oracle-DB. Der Referenzdatenbestand kann dabei selbst aufgebaut, über einen Datenprovider (zum Beispiel HERE) bezogen werden oder eine Kombination von beidem sein. Die erwartete Struktur des Referenzdatenbestandes ist über die Spalten der Tabelle ELOC_ADMIN_AREA_SEARCH definiert:

```
SQL> describe eloc_admin_area_search
```

```
Name                                      Null?    Type
----------------------------------------- -------- ------------------
FULL_NAME                                          VARCHAR2(4000)
AREA_NAME                                          VARCHAR2(100)
KEY                                                VARCHAR2(4000)
LANG_CODE                                          VARCHAR2(3)
CENTER_LONG                                        NUMBER
CENTER_LAT                                         NUMBER
POPULATION                                         NUMBER
PART_ID                                            NUMBER
```

Hinweis: Sofern die Nutzungsrechte für die Datenbank auch die Partitioning-Option einschließen, wird empfohlen, den Referenzdatenbestand zu partitionieren. Ausdruck dessen ist die Spalte PART_ID. So enthält der beispielhafte Referenzdatenbestand von HERE ca. 2,3 Mio. Einträge. Dieser wird mit der Installation der Database Examples im Ordner $ORACLE_HOME/md/demo/GeoSearch bereitgestellt.

Die Spalte FULL_NAME ist dafür vorgesehen, diese mit allen Namensvarianten eines Ortes sowie mit den Namensvarianten der unterschiedlichen hierarchischen Ebenen als durch Kommas separierte Liste zu befüllen. Folgendes Beispiel zeigt einen Eintrag:

VOLKSPARK FRIEDRICHSHAIN,FRIEDRICHSHAIN, BERLIN-FRIEDRICHSHAIN, BLN-FRIEDRICHSHAIN,BERLIN,BLN,DEUTSCHLAND,DE,DEU

Diese Spalte wird dann mit den Mitteln von Oracle Text volltextindexiert:

```
SQL> exec ctx_ddl.create_preference('eloclex', 'WORLD_LEXER');

SQL> exec ctx_ddl.create_section_group('elocsec', 'BASIC_SECTION_GROUP');

SQL> exec ctx_ddl.create_preference('elocds', 'MULTI_COLUMN_DATASTORE');

SQL> exec ctx_ddl.set_attribute('elocds', 'COLUMNS', '''XSTARTX ''||full_
name||'' XENDX''');

SQL> exec ctx_ddl.set_attribute('elocds', 'DELIMITER', 'NEWLINE');

SQL> /* Lokalen Volltextindex anlegen für part. ELOC_ADMIN_AREA_SEARCH Ta-
belle */

SQL> create index geo_search_idx on ELOC_ADMIN_AREA_SEARCH(FULL_NAME)

     indextype is ctxsys.context

     order by population parameters ('sync (on commit) lexer eloclex
     datastore elocds section group elocsec memory 400M')

     parallel 4 local;
```

Die zuvor erwähnte Funktion SDO_UTIL. GEO_SEARCH erwartet zwei Eingangswerte. Der erste Parameter ist ein Text mit einem oder mehreren durch Kommas separierten Suchbegriffen. Der zweite Parameter ist optional und steuert, ob die Oracle-Text-Fuzzy-Suche genutzt wird oder nicht. Letzteres ist der Default. In den folgenden Beispielen nutzt nur die zweite Abfrage die Oracle-Text-Fuzzy-Suche:

```
select * from table(sdo_util.geo_search('Volkspark Friedrichshain'));
select * from table(sdo_util.geo_search('Volkspark Friedrichshein',1));
```

Rückgabewert der Funktion ist eine Liste von auf Basis der Volltextsuche gefundenen Einträgen aus dem Referenzdatenbestand. Die Liste enthält auch alle hierarchischen Ebenen (Spalte KEY), die Längen- und Breitengrad-Informationen (Spalten CENTER_LONG und CENTER_LAT) sowie ein Ranking (Spalte RANK) der gefundenen Einträge:

```
NAME                      KEY
------------------------- ------------------------------------------------
----
LAN                CENTER_LONG CENTER_LAT       RANK
------------------ ----------- ---------- ----------
VOLKSPARK FRIEDRICHSHAIN FRIEDRICHSHAIN|BERLIN|BERLIN|BERLIN|DEUTSCHLAND
GER                     52.5267    13.4339         20

MÄRCHENBRUNNEN            MÄRCHENBRUNNEN|VOLKSPARK FRIEDRICHSHAIN|FRIEDRIC
HSHAIN|BERLIN|BERLIN|BERLIN|DEUTSCHLAND
GER                     52.5281     13.4269        19
```

Zusätzliche Einträge, um Daten anzureichern, können sehr leicht ergänzt werden. Dies verdeutlichen die zwei nachfolgenden INSERT-Statements, welche auch schon das Ergebnis für die beiden vorherigen SDO_UTIL.GEO_SEARCH-Abfragen liefern. Bitte schließen Sie dabei immer die Abfragen mit einer COMMIT-Operation ab:

```
insert into eloc_admin_area_search
(full_name, area_name, key, lang_code, center_long, center_lat, part_id)
values
```

```
('MÄRCHENBRUNNEN,VOLKSPARK FRIEDRICHSHAIN,FRIEDRICHSHAIN,BLN FRIEDRICHSHAIN,
BERLIN,BLN,DEUTSCHLAND,DE,DEU','MÄRCHENBRUNNEN', 'FRIEDRICHSHAIN|BERLIN|BERL
IN|BERLIN|DEUTSCHLAND', 'GER', 52.5281, 13.4269, 10);

insert into eloc_admin_area_search
(full_name, area_name, key, lang_code, center_long, center_lat, part_id)
values
('VOLKSPARK FRIEDRICHSHAIN,FRIEDRICHSHAIN,BLN FRIEDRICHSHAIN,BERLIN,BLN,DEUT
SCHLAND,DE,DEU','VOLKSPARK FRIEDRICHSHAIN', 'FRIEDRICHSHAIN|BERLIN|BERLIN|BE
RLIN|DEUTSCHLAND', 'GER', 52.5267,13.4339, 10);
```

Grundlegendes Wissen sowie viele praktische Tipps zum Thema Oracle Text finden sich im deutschsprachigen Blog http://oracle-text-de.blogspot.com. Der Blog-Post „Wie ähnlich soll es sein ...? Ein paar Worte zum FUZZY-Operator“ unter http://oracle-text-de.blogspot.com/2009/06/wie-ahnlich-soll-es-sein-ein-paar-worte.html behandelt speziell Ähnlichkeitssuchen mittels des Fuzzy-Operators.

2 RÄUMLICHE INDIZIERUNG UND PARTITIONIERUNG

Die sehr weitgehenden Möglichkeiten, Tabellen und Indizes in der Oracle-Datenbank zu partitionieren, galten bisher nur eingeschränkt für das Management raumbezogener Daten (Geodaten oder auch Location Data). Das hat sich geändert. Mit Version 12.2 sind nun weitere Partitioning-Methoden wie RANGE, LIST, HASH, INTERVAL, VIRTUAL COLUMN oder auch RANGE-RANGE, RANGE-LIST, LIST-HASH, LIST-LIST und mehr unterstützt. Damit kommen die Vorteile von Partitionierung sowohl aus Anwendungsentwicklungssicht, wie

bessere Performanz bei Abfragen durch Parallelisierung und Partition Pruning, als auch aus Datenbankadministrationssicht, wie das Einpassen von Admin-Tätigkeiten in immer kürzeren Wartungsfenstern ebenfalls durch Parallelisierung der Arbeiten auf Partitionen, viel besser zum Tragen.

Stellen Sie sich die effektive Nutzung und Analyse von immer größeren Datenmengen vor, die zum Beispiel über Sensoren, Smart Devices oder im Social-Media-Umfeld generiert werden. Ganz häufig sind Orts- und Zeitinformationen wesentliche Bestandteile eines Datensatzes, die in Echtzeit oder nach dem Persistieren der Daten als Analysekriterien genutzt werden. Was liegt da näher, als die Daten zu partitionieren, also zum Beispiel in Zeitscheiben „aufzuteilen“:

```
create table gps_tracker (
  id number,
  position sdo_geometry,                                    -- GPS-Koordinate
  datetime timestamp,                                       -- Zeitstempel
  constraint gps_tracker _pk primary key (id) enable)
partition by range (datetime) interval(numtoyminterval(1, 'MONTH'))
(
  partition pos_data_p1 values less than (to_date('01.01.2017', 'DD.MM.YYYY')),
  partition pos_data_p2 values less than (to_date('01.02.2017', 'DD.MM.YYYY')),
  partition pos_data_p3 values less than (to_date('01.03.2017', 'DD.MM.YYYY'))
);
```

Ermöglicht wurden die zusätzlichen Partitioning-Methoden auch durch Optimierungen am räumlichen Index. Diese werden über

den neuen Domain-Index SPATIAL_INDEX_V2 (bisher: SPATIAL_INDEX) bereitgestellt. Die nächsten zwei Beispiele demonstrieren die Verwendung. Das zweite Beispiel nutzt einen zusätzlichen Parameter für einen möglichst optimal aufgebauten Index. In diesem Fall wird der Parameter LAYER_GTYPE mit dem Wert POINT belegt. Das hat den Hintergrund, dass GPS-Daten typischerweise als Punkt (mit Raumbezug) abgebildet werden:

```
create index gps_tracker_six on gps_tracker(position)
       indextype is mdsys.spatial_index_v2 local;

create index gps_tracker_six on gps_tracker(position)
       indextype is mdsys.spatial_index_v2 parameters ('layer_gtype=POINT')
       local;
```

Warum braucht es für Geodaten spezielle Indizes? Diese werden benötigt, um effizient räumliche Operatoren auch auf großen Datenmengen anwenden zu können. Eine Vielzahl solcher Operatoren gibt es vorimplementiert in der Datenbank. Sie werden zum Beispiel für Umgebungssuchen eingesetzt (Abfragen wie „Nearest Neighbor“ oder „Within Distance“) oder für die Feststellung der Art, wie zwei räumliche Objekte miteinander in Beziehung stehen (zum Beispiel INSIDE, TOUCH). Als Beispiel sei hier eine räumliche Abfrage mit dem räumlichen Operator SDO_INSIDE angeführt. Hierüber werden alle in der Tabelle GPS_TRACKER erfassten Positionsdaten im Umkreis von 10 km um das Olympiastadion (Eintrag in Tabelle REGIONS_OF_INTEREST) gesucht. Die gleiche räumliche Abfrage

kann alternativ auch mittels SDO_WITHIN_DISTANCE formuliert werden:

```
select p.id, p.datetime
from gps_tracker p, regions_of_interest r
where sdo_inside(p.position,

      sdo_geom.sdo_buffer(r.geometry, 1000, 0.5,
'unit=km')))='TRUE'

and r.name = 'Olympiastadion';
```

Es wird generell empfohlen, mit dem Upgrade der Datenbank auf Version 12.2 auch die schon vorhandenen räumlichen Indizes auf den neuen Indextyp umzustellen.

3 GEOJSON – DER ORTS- UND RAUMBEZUG IN JSON

JSON (JavaScript Object Notation) findet als kompaktes und einfach zu lesendes Format immer größere Verwendung für den Datenaustausch. GeoJSON ist – wie der Name schon treffend beschreibt – das Format, um standardbasierte, geografische, raumbezogene Daten auszutauschen. Alle wichtigen geografischen Informationen und Typen müssen über das Format abgedeckt werden können. Ein Beispiel für eine Formatvorlage findet sich auf http://geojson.org.

Nicht nur Sensoren, Smart Devices oder die Kommunikation zwischen Geräten im Internet der Dinge setzen auf ein solches Austauschformat. Auch zum Beispiel im Bereich Open-Data

verfügbare Dokumente beziehungsweise Datensätze nutzen GeoJSON für die Beschreibung von geografischen und geometrischen Objekten. Ein sehr gutes Beispiel dafür ist das Open-Data-Portal der Deutschen Bahn (http://data.deutschebahn.com/dataset). Ein Ziel solcher Datenportale ist es, die Entwicklung neuer, datengetriebener Produkte in der Developer Community voranzutreiben.

Die mit Version 12.1.0.2 eingeführte Unterstützung für JSON hat die Möglichkeit eröffnet, die Oracle-Datenbank auch im Sinne eines (JSON) Document Store zu verwenden, wie man es typischerweise aus dem Umfeld von NoSQL-Datenbanken kennt. Die mit 12.2 eingeführte Unterstützung für GeoJSON ermöglicht das Management und die Nutzung von Geodaten, die als JSON-Objekte beschrieben sind. Somit können in Abfragen mittels SQL beliebig relationale Attribute mit JSON/GeoJSON-Objekten, XML und weiteren Datentypen kombiniert werden. Das entspricht ganz dem multimodalen Ansatz der Oracle-Datenbank.

Doch schauen wir zunächst einmal zwei Objekte im GeoJSON-Format an: Ortsinformationen (hier „Feature“) beziehungsweise Kollektionen von Orten (hier: „FeatureCollection“) werden über ihre Koordinaten und zusätzlichen Attribute in Form von Key-Value-Paaren beschrieben. Eine weitere wichtige Information ist im Wert für den zweiten Key "type" (hier: "type": "Point") angegeben. Hierüber erfahren wir, ob im GeoJSON-Objekt ein Punkt, eine Linie, ein Polygon oder Multi-Versionen davon repräsentiert sind:

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [13.06156, 52.40102]
  },
  "properties": {
    "name": "Oracle Geschäftsstelle Potsdam"
  }
}
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates": [13.06156,
52.40102]},
      "properties": {"name": " Oracle Geschäftsstelle Potsdam"}
    },
    {
      "type": "Feature",
      "geometry": {"type": "Point", "coordinates":
[11.51659,48.17573]},
      "properties": {"name": " Oracle Geschäftsstelle München"}
    }
  ]
}
```

GeoJSON-Objekte können in der gleichen Weise in Tabellen abgelegt werden wie JSON-Objekte. Die Validierung, ob gültiges JSON

übergeben wird, kann bei der Tabellendefinition über das bekannte CHECK CONSTRAINT IS JSON mitgegeben werden:

```
create table cities_geojson (
  id number generated always as identity,
  location varchar2(4000)
  constraint ensure_json check (location is json));
```

Ein gültiger Eintrag in die Tabelle CITIES_GEOJSON sieht dabei wie folgt aus:

```
insert into cities_geojson (1, '
{"a":{"type":"Point","coordinates":[+50.0,+10.0]}}');
```

Speziell für das Format GeoJSON, bietet ab Version 12.2 die Funktion JSON_VALUE die Möglichkeit, GeoJSON-Objekte direkt als Objekt vom Typ SDO_GEOMETRY zurückzugeben:

```
SQL> select json_value(location,'$.a' returning sdo_geometry) geom
     from cities_geojson;

GEOM(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------
SDO_GEOMETRY(2001, 4326, SDO_POINT_TYPE(50, 10, NULL), NULL, NULL)
```

Warum ist das sinnvoll? SDO_GEOMETRY ist das native Format der Oracle-Datenbank, um raumbezogene Daten, vereinfacht auch Geodaten, zu speichern und abzufragen. Dafür gibt es bereits eine Vielzahl von spezifischen Methoden, Operatoren und Funktionen. Im Abschnitt „Räumliche Indexierung und Partitionierung“ wurde schon

beschrieben, wie ein räumlicher Index auf SDO_GEOMETRY-Objekten angelegt und wofür dieser verwendet wird. Die Möglichkeit, GeoJSON direkt als SDO_GEOMETRY zurückzugeben, kann auch genutzt werden, um einen function-based räumlichen Index auf den GeoJSON-Objekten anzulegen:

```
create index cities_geojson_six on geojson_tab (
  json_value (location, '$.a' returning sdo_geometry)
  indextype is mdsys.spatial_index_v2 parameters ('layer_gtype=POINT');
```

Eine räumliche Abfragen wie beispielsweise eine Nearest-Neighbor-Suche mit räumlich indexierten GeoJSON-Objekten, sieht dann so aus:

```
select * from cities_geojson
  where sdo_nn (
    json_value (location, '$.a' returning sdo_geometry),
    json_value ('{"type":"Point","coordinates":[+50.0,+10.0]},
'$.a'
                  returning sdo_geometry)) = 'TRUE';
```

Auch die umgekehrte Anwendung funktioniert: SDO_GEOMETRY-Objekte können direkt als GeoJSON zurückgegeben werden. Auf diese Weise können in der Datenbank liegende Geodaten als GeoJSON zum Beispiel über einen REST-Service bereitgestellt werden. Funktional erledigt das die für den Datentyp SDO_GEOMETRY neu implementierte Methode GET_GEOJSON. Deren Anwendung skizziert das nachfolgende Statement:

```
SQL> select c.geometry.get_geojson() from world_capital_cities c
     where c.name = 'Berlin';

G.GEOMETRY.GET_GEOJSON()
--------------------------------------------------------------------------------
{ "type": "Point", "coordinates": [13.412956, 52.506394] }
```

Die Spalte GEOMETRY vom Datentyp SDO_GEOMETRY wird durch die Methode GET_GEOJSON in ein GeoJSON-Objekt umgewandelt und zurückgegeben.

Dies soll nur ein Exkurs in die Neuerungen der Version 12.2 zu Funktionalität rund um die Nutzung und Analyse raumbezogener Daten darstellen. Alle Handbücher dazu mit detaillierten Informationen finden sich online auf http://docs.oracle.com > Database > Spatial and Graph.

Der englischsprachige Blog *Oracle Spatial and Graph* ist über https://blogs.oracle.com/oraclespatial/ zugänglich. Fragen und Hinweise zu den beschriebenen Neuerungen können über das Oracle-Technology-Forum für Oracle Spatial and Graph, https://community.oracle.com/community/database/oracle-database-options/spatial, direkt an das Development-Team adressiert werden.

Weiterführende Informationen

Interessante Links

- **Oracle Live mit 12.2 Tutorials**
  https://livesql.oracle.com/

- **Software OTN Download**
  http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html

- **Sharding FAQ**
  http://www.oracle.com/technetwork/database/availability/sharding-faq-3610620.pdf

- **Database Cloud Link**
  https://cloud.oracle.com/database

- **Download SQL Developer und SQLcl**
  http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html

Blogs

- **Deutschsprachiger Datenbank & Cloud Technologie Blog**
  https://blogs.oracle.com/coretec/

- **Upgrade Blog**
  https://blogs.oracle.com/UPGRADE/

- **Oracle Spatial Blog**
  https://blogs.oracle.com/oraclespatial/

- **Oracle Text Blog**
  http://oracle-text-de.blogspot.com

Handbücher

- **Alle Handbücher zu Oracle Database 12.2**
  http://docs.oracle.com/database/122/index.htm

- **Licensing Information User Manual**
  http://docs.oracle.com/database/122/DBLIC/toc.htm

Dojos

Alle Dojos finden Sie unter http://tinyurl.com/dojoonline

FÜR IHRE NOTIZEN

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

ORACLE®

Herausgeber: Roland Aussermeier, Oracle Deutschland B.V.

Design: volkerstegmaier.de, Lektorat: Oliver Lauber
Druck: Stober GmbH, Eggenstein