SQL Tipps und Tricks

Ulrike Schwinn
Oracle Deutschland B.V. & Co.KG
München

Schlüsselworte
Row Limiting Klausel, Ausführungsplan, SQL Developer, SQLcl, Oracle Live SQL, Lateral Klausel, approx_count_distinct, JSON, 12.1, 12.2 Ausblick, Exadata Express Cloud Service

Einleitung

„Viel Funktionalität mit wenig Code“, einfache Verwendbarkeit und gute Lesbarkeit sind die Leitmotive für diesen Beitrag. Oracle SQL wie auch PL/SQL werden in jedem Release weiterentwickelt. Dabei wird nicht nur der Funktionsumfang vergrößert, sondern auch die Programmierung und Verwendung erleichtert. Anbindung an heterogene Daten wie zum Beispiel XML und ganz aktuell auch JSON stehen dabei ebenfalls im Fokus. Wichtig ist dabei immer, dass die Performance nicht unter der zunehmenden Funktionalität leiden darf, sondern ganz im Gegenteil die Abfragen performant erfolgen. Werkzeuge wie das altbekannte traditionelle SQL*Plus, der SQL Developer bzw. seine neueste Linemode Variante SQLcl, die seit September nun auch produktiv verfügbar ist, eignen sich gut zum Testen. Möchte man SQL Code schnell mal ausprobieren bzw. mit Kollegen teilen OHNE eine geeignete Oberfläche oder Installation zu besitzen, kann man auch auf **Oracle Live SQL** zurückgreifen. Die Oracle Live SQL Site ist eine kostenfreie browserbasierte Schnittstelle (siehe Link im Kapitel „Weitere Informationen“). Nur ein OTN Account ist erforderlich. Hier kann man SQL Statements ausprobieren oder eigene Skripte erstellen und auch teilen. Gespeicherte Tutorials helfen dann beim Einstieg in ein spezielles Thema.

Der folgende Beitrag stellt ein paar ausgewählte und interessante Features aus dem aktuellen Datenbankrelease 12*c* vor und zeigt wie einfach und effizient Daten innerhalb der Datenbank zu verarbeiten, zu bewerten oder abzufragen sind.

SQL Funktionalitäten zum Abfragen von relationalen Daten

Starten wir mit wichtigen SQL Funktionen aus dem Datenbankrelease 12.1, die jedem Entwickler bekannt sein sollten - Top N Abfragen und das Blättern in Applikationen. Bisher war das Arbeiten mit ROWNUM und die geschickte Nutzung einer Subquery dazu erforderlich. Im ersten Beispiel wird eine Implementierung vor 12*c* gezeigt. Die Aufgabe besteht darin, die 5 kostengünstigsten Produkte aufzulisten. Wir nutzen dabei das altbekannte Demoschema SH und die Tabelle PRODUCTS.

```
select prod_id, prod_list_price from
         (select * from sh.products order by prod_list_price)
          where rownum <= 5;
```

Im erweiterten Beispiel wird das Weiterblättern demonstriert.

```
select prod_id, prod_list_price
from (select prod_id, prod_list_price, rownum AS rnum
      from   (SELECT  prod_id, prod_List_price
                FROM  products
                ORDER BY prod_list_price)
      where rownum <= 10)
where rnum >= 5;
```

Diese Beispiele kann man viel übersichtlicher in 12*c* mit der sogenannten **ROW Limiting Klausel** programmieren. Die Syntax sieht dabei folgendermaßen aus:

```
[ OFFSET offset { ROW | ROWS } ] [ FETCH { FIRST | NEXT }
[ { rowcount | percent PERCENT } ] { ROW | ROWS } { ONLY | WITH TIES } ]
```

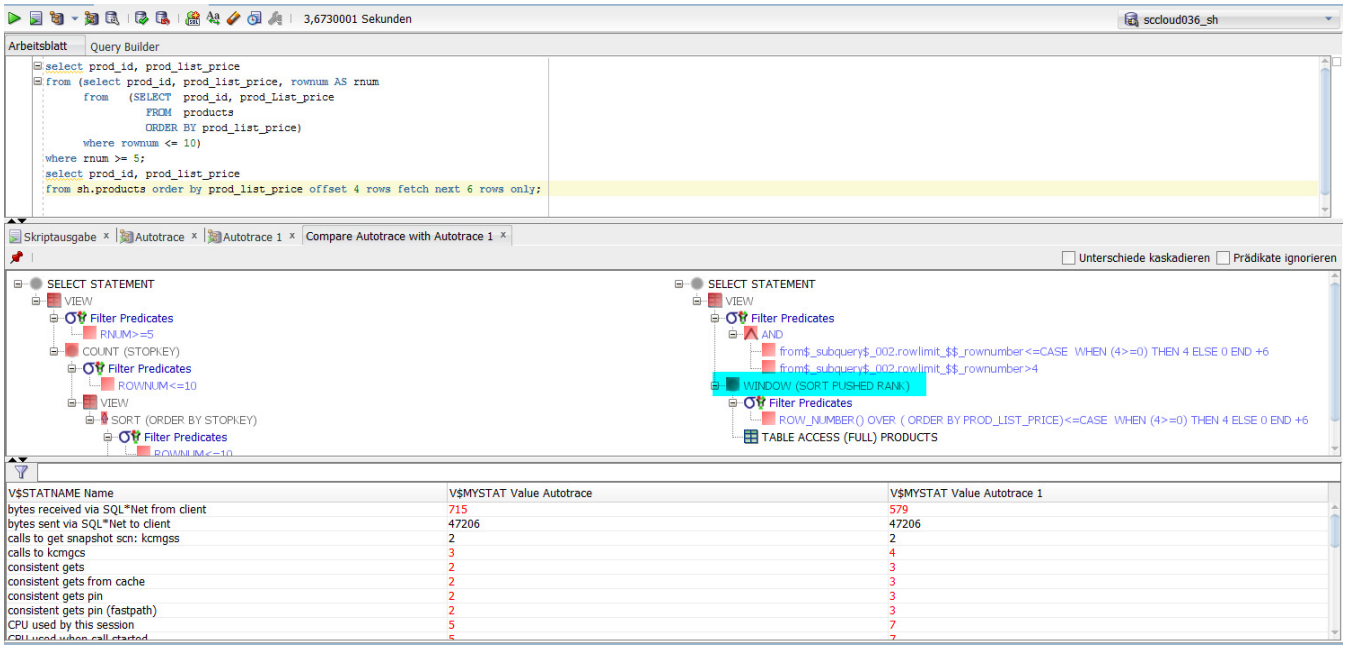
Unsere Beispiele von oben haben dann folgendes Aussehen: Für die 5 kostengünstigsten Produkte ergibt sich folgender SQL Code: es wird einfach „*fetch first <Anzahl der Zeilen> rows only*“ angefügt.

```
select prod_id, prod_list_price
from  sh.products order by prod_list_price fetch first 5 rows only;
```

Für das Blättern der nächsten 5 Zeilen ergibt sich die erweiterte Syntax mit *„offset“.*

```
select prod_id, prod_list_price
from sh.products order by prod_list_price offset 4 rows fetch next 6 rows only;
```

Die erste Frage, die man sich stellen sollte: Was hat sich geändert und wie wird diese Abfrage ausgeführt? Beim Einsatz von SQL Neuerungen ist es in der Regel sinnvoll, sich den Ausführungsplan anzusehen. Dazu bietet sich das Package DBMS_XPLAN oder noch viel einfacher die entsprechende Funktionalität im **SQL Developer Worksheet** an. SQL Developer bietet sogar die Möglichkeit nicht nur den Ausführungsplan selbst sondern sogar die **Unterschiede von Ausführungsplänen** anzuzeigen. Auch Informationen zu der AUTOTRACE Funktion, sofern die Privilegien diesen Zugriff gestatten, sind möglich. Dabei werden sogar sogenannte „Hotspots“ angezeigt, die man sich unter Umständen genauer anschauen sollte. In unserem Fall ist der neue Ausführungsplan nicht nur kürzer, sondern verwendet statt einer Sortierung (siehe Operation SORT) eine intelligente Window Funktion (blau markierte Operation WINDOW) um auf die Daten zuzugreifen.



Eine weitere interessante Neuigkeit, auf die ich durch Zufall gestossen bin, ist die neue Unterstützung der **LATERAL** Klausel in 12*c* – übrigens ein SQL;2003 Standard. Vor 12*c* war es nicht möglich, Tabellen außerhalb einer Inline View (Korrelation) zu referenzieren. Ein einfaches Beispiel in 11*g* Release 2 zeigt die Problematik.

```
select e.ename, e.sal, avg_sal from emp e, (select avg(sal) avg_sal from emp e1 where
e.deptno=e1.deptno);
*


ERROR at line 1:
ORA-00904: "E"."DEPTNO": invalid identifier
```

Mit 12*c* und der neuen **LATERAL** Syntax ist dies aber einfach möglich.

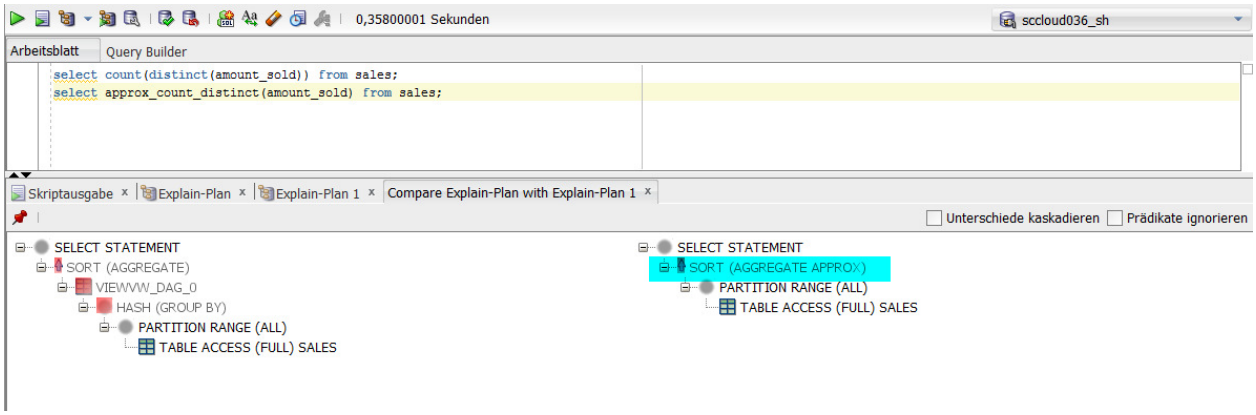
```
select e.ename, e.sal, avg_sal from scott.emp e,
lateral (select avg(sal) avg_sal from scott.emp e1 where e.deptno=e1.deptno);

ENAME                SAL    AVG_SAL

---------- ---------- ----------
JAMES                950 1566.66667
TURNER              1500 1566.66667
BLAKE               2850 1566.66667
MARTIN              1250 1566.66667
WARD                1250 1566.66667
ALLEN               1600 1566.66667
FORD                3000       2175
ADAMS               1100       2175
...
```

Bitte beachten Sie die Einschränkungen, die im SQL Reference Guide gelistet sind.

Eine weitere Neuigkeit in 12*c* ist eine neue Funktion um die (ungefähre) **Anzahl der unterschiedlichen Spalteneinträge** zu zählen. Statt die Verwendung von COUNT DISTINCT oder einer Abfrage auf die Spalte NUM_DISTINCT in USER_TAB_COLUMNS bietet sich die Nutzung der Funktion APPROX_COUNT_DISTINCT an. Wie im vorangegangenen Beispiel vergleichen wir die beiden Ausführungspläne. Man kann leicht erkennen, daß der zweite Ausführungsplan aus weniger Operationen bzw. einer speziellen Operation SORT (AGGREGATE APPROX), die extra für diese Art von Ausführung optimiert wurde, besteht.



Neue Funktionen für JSON Daten

XML Daten können schon seit Langem in der Oracle Datenbank gespeichert und abgefragt werden. JSON Daten werden hingegen häufig in NoSQL oder anderen speziellen Datenbanken gespeichert. Diese erlauben zwar die Speicherung und den Zugriff der Daten, aber weisen kein vergleichbares Konsistenzmodell, Transaktionsmodell und andere Standardfunktionalitäten von relationalen Datenbanken auf. Neu in Oracle Database 12*c* (12.1.0.2) ist die Möglichkeit auf JSON Daten mit Standard Datenbankmitteln zuzugreifen. Die Idee dahinter ist, nicht nur einen einfachen Textstring zu speichern und auf diesen zuzugreifen, was schon immer in jedem Release möglich war, sondern auch spezielle JSON Pfad Zugriffe oder JSON Validierungen zu ermöglichen, um nur einige Features zu nennen.

Bevor man über den Zugriff von JSON Daten in der Datenbank nachdenkt, stellt man sich allerdings folgende Fragen: Wie kann man überhaupt JSON in der Datenbank zur Verfügung stellen? Wie werden die Daten gespeichert? Eines vorweg: Es gibt keinen speziellen JSON Datentyp - ganz im Unterschied zu XML in der Datenbank (auch XMLDB). JSON kann also einfach in Spalten vom Datentyp VARCHAR2 oder LOB gespeichert werden. Mit der Bedingung IS JSON kann man die Daten dann zusätzlich validieren - auf Wohlgeformtheit oder auf die Art der Syntax Verwendung (STRICT oder LAX).

Dabei können JSON Daten sogar in einem DMP File über eine External Table zugegriffen werden. Hinweis: Diese Beispieldaten stehen übrigens in $ORACLE_HOME/demo/schema/order_entry zur Verfügung.

```
SQL> create table json_dump_file_contents (json_document CLOB)
     ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY order_dir
                            ACCESS PARAMETERS (RECORDS DELIMITED BY 0x'0A'
                            BADFILE 'JSONDumpFile.bad'
                            LOGFILE 'JSONDumpFile.log'
                            FIELDS (json_document CHAR(5000)))
                            LOCATION ('PurchaseOrders.dmp'))
                            REJECT LIMIT UNLIMITED;
```

```
SQL> set long 10000 pagesize 1000
SQL> select * from json_dump_file_contents where rownum=1;

JSON_DOCUMENT
--------------------------------------------------------------------------------
{"PONumber":1,"Reference":"MSULLIVA-20141102","Requestor":"Martha
Sullivan","User":"MSULLIVA","CostCenter":"A50","ShippingInstructions":{"name":"Martha
Sullivan","Address":{"street":"200 Sporting Green","city":"South San
Francisco","state" "CA", "zipCode":99236, "country":"United States of
America"},"Phone":[{"type": "Office", "number":"979-555-6598"}]},"Special
Instructions":"Surface Mail", "LineItems":[{"ItemNumber":1,"Part":{"Description":"Run
Lola Run", "UnitPrice":19.95,"UPCCode":43396040144},"Quantity":7.0},{"ItemNumber":
2,"Part":{"Description":"Felicia's Journey", "UnitPrice": 19.95, "UPCCode":
12236101345},"Quantity":1.0},{"ItemNumber":3,"Part":{"Description":"Lost and
Found","UnitPrice":19.95,"UPCCode":85391756323},"Quantity":8.0},{"ItemNumber":4,"Part"
:{"Description":"Karaoke: Rock & RollHits of 80's & 90's 8", "UnitPrice": 19.95,
"UPCCode":13023009592},"Quantity":8.0{"ItemNumber":5,"Part":{"Description":"Theremin:
An Electronic Odyssey","UnitPrice":19.95,"UPCCode":27616864451},"Quantity":8.0}]}
```

Ähnlich wie bei der Verwendung von XML Dokumenten stehen neue Funktionen und Operatoren zur speziellen Nutzung für JSON Dokumente zur Verfügung. Im folgenden sind ein paar Beispiele gelistet:

- IS JSON/IS NOT JSON: Testet auf Wohlgeformtheit bzw. auf die Art der Nutzung
- JSON_EXISTS: Testet in einer Bedingung auf Existenz eines speziellen Wertes
- JSON_VALUE: Selektiert einen skalaren Wert
- JSON_QUERY: Selektiert ein JSON Fragment - normalerweise ein Objekt oder Feld
- JSON_TABLE: Projiziert JSON Daten in ein relationales Format über eine virtuelle Tabelle, ähnlich einer relationalen Inline View

Die nächsten Beispiele demonstrieren die einfache Verwendung: zuerst wird die Ausgabe von skalaren Werten demonstriert. Verwendet man die Bedingung IS JSON als Check Constraint ist sogar eine Abfrage mit Punktnotation möglich.

```
select min(json_value (json_document, '$."PONumber"' returning number)),
       max(json_value (json_document, '$."PONumber"' returning number))
from json_tab;
```

Will man Fragmente selektieren - wie im Fall von Arrays, kann man auf JSON_QUERY zurückgreifen.

```
SQL> set long 1000
SQL> select json_query(json_document,'$.ShippingInstructions')
     from json_tab
     where json_value(json_document,'$.PONumber' returning number)=1000;

JSON_QUERY(JSON_DOCUMENT,'$.SHIPPINGINSTRUCTIONS')
--------------------------------------------------------------------------------
{"name":"Charles Johnson","Address":{"street":"Magdalen Centre, The Isis Science
Park","city":"Oxford","county":"Oxon.","postcode":"OX9 9ZB","country":"United
Kingdom"},"Phone":[{"type":"Office","number":"66-555-3120"}]}
```

Wie kann man nun JSON Daten in eine relationale Form projizieren? Die SQL Funktion JSON_TABLE (vergleichbar mit XMLTABLE bei XMLDB) überführt die JSON Daten in relationale Zeilen und Spalten einer virtuellen Tabelle. Folgendes Beispiel zeigt die Funktionsweise. Verwendet wird die Tabelle JSON_TAB mit der Spalte JSON_DOCUMENT

von oben. Als Ergebnis sollen 3 relationale Spalten zur Verfügung stehen - nämlich REQUESTOR (mit VARCHAR2 (32)), ADRESSE (im JSON Format) und die Spalte SPECIAL (mit VARCHAR2(10)). In der FROM Klausel wird dazu die Tabelle JSON_TAB und die SQL Funktion JSON_TABLE verwendet. Die Funktion JSON_TABLE benötigt als erstes Argument die Spalte mit den JSON Daten, einen JSON Ausdruck (hier $) und das Schlüsselwort COLUMNS, das das Mapping auf die relationalen Spalten und die entsprechenden Datentypen vornimmt.

```
create or replace view json_view as
           select jt. ponumber, jt.requestor, jt.adresse, jt.special
           from json_tab,
           json_table (json_document, '$'
                              COLUMNS (
          ponumber number PATH '$.PONumber',
          requestor varchar2(32 CHAR) PATH '$.Requestor',
          special varchar2(10) PATH '$."Special Instructions"',
          adresse varchar2(400) FORMAT JSON PATH '$.ShippingInstructions.Address')) jt
  where json_value(json_document,'$.PONumber' returning number error on error) < 5000;

SQL> desc json_view

Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
PONUMBER                                           NUMBER
REQUESTOR                                          VARCHAR2(128)
ADRESSE                                            VARCHAR2(400)
SPECIAL                                            VARCHAR2(10)
```

Die Funktion JSON_TABLE generiert für jeden JSON Wert, auf den das JSON Pattern zutrifft, eine Zeile. Die relationale View JSON_VIEW projiziert die gewünschte Sichtweise. Die Filterbedingung reduziert dabei die Zeilenzahl auf Dokumente mit "PONumber" kleiner als 5000.

Wie man sich vorstellen kann, eignen sich die Funktionen JSON_VALUE und JSON_EXISTS gut dazu, einen Function Based Index auf die entsprechenden JSON Pfade mit skalaren Werten zu erzeugen. Dabei können auch Bitmap Indizes erzeugt werden.

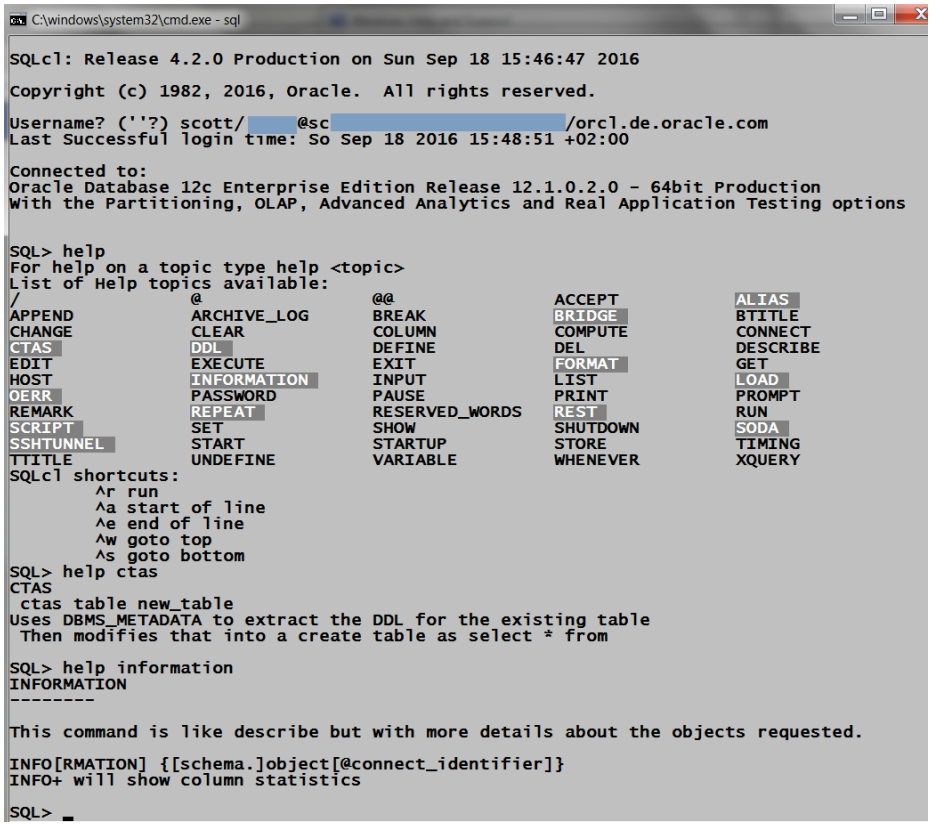
```
create index PONUMBER_IDX on json_tab(JSON_QUERY(JSON_DOCUMENT, '$."PONumber"'
returning number error on error));
```

Auch hier sollte man die Verwendung des Index immer mit der Ausgabe des Ausführungsplan überprüfen. Mehr zu diesem Thema gibt es in unserem Dojo oder in unserem Community Tipp zu lesen (siehe „Weitere Informationen“ unten).

SQLcl – das moderne Linemode Werkzeug

Noch nie etwas von SQLcl gehört? SQLcl ist eine Art Linemode Version von SQL Developer und kann auch als moderne SQL*Plus Variante bezeichnet werden. Nach langer Early Adopter Verfügbarkeit kann man sie seit September diesen Jahres auch produktiv laden. Gute Zusammenfassung mit interessanten Fakten darüber findet man viele im Internet. Daher möchte ich an dieser Stellen nur einen Einstieg in das Thema geben und zwei meiner Lieblingsfunktionen kurz hervorheben. Wie der „große Bruder“ SQL Developer wird einfach nur eine zip Datei (ca. 60 MB) von OTN (Link siehe unten) geladen, ausgepackt und schon kann es losgehen. Die Verbindung über die Datenbank geht dabei ganz einfach über den sogenannten Easy Connect, keine weiteren Informationen sind erforderlich. Möchte man einen Überblick über die verfügbaren SQLcl Kommandos erhalten, sollte man zuerst das Kommando „help“

eingeben. Die dunkelgrau unterlegten Kommandos zeigen die neuen speziellen SQLcl Kommandos an, die es sich lohnt näher zu betrachten.



SET SQLFORMAT zum Beispiel ermöglicht eine Ausgabe in unterschiedlichen Formaten. Kombiniert man dieses Kommando mit einem SELECT lassen sich verschiedene Ausgaben erzielen wie zum Beispiel CSV, XML, JSON oder auch INSERT. Im folgenden Beispiel generieren wir JSON Daten.

Kombiniert man die SET SQLFORMAT, DDL und DBMS_METADATA miteinander hat man auch schnell ein Testskript zum Erzeugen von Testdaten einer Tabelle produziert.

```
SQL> execute dbms_metadata.set_transform_param(dbms_metadata.session_transform,'SEGMENT_ATTRIBUTES',
false);

PL/SQL procedure successfully completed.

SQL>
SQL> ddl scott.emp
  CREATE TABLE "SCOTT"."EMP"
   (    "EMPNO" NUMBER(4,0),
        "ENAME" VARCHAR2(10),
        "JOB" VARCHAR2(9),
        "MGR" NUMBER(4,0),
        "HIREDATE" DATE,
        "SAL" NUMBER(7,2),
        "COMM" NUMBER(7,2),
        "DEPTNO" NUMBER(2,0),
        "T" NUMBER,
        "D" NUMBER,
         CONSTRAINT "PK_EMP" PRIMARY KEY ("EMPNO")
  USING INDEX  ENABLE,
         CONSTRAINT "FK_DEPTNO" FOREIGN KEY ("DEPTNO")
          REFERENCES "SCOTT"."DEPT" ("DEPTNO") ENABLE
   )
  INMEMORY PRIORITY NONE MEMCOMPRESS FOR QUERY LOW
  DISTRIBUTE AUTO NO DUPLICATE ;
SQL> set sqlformat INSERT
SQL> select * from scott."EMP" offset 5 rows fetch next 2 rows only;
REM INSERTING into SCOTT."EMP"
SET DEFINE OFF;
Insert into SCOTT."EMP" (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO,T,D) values ('7698','BLAKE','M
ANAGER','7839',to_timestamp('01.05.81','DD.MM.RR HH24:MI:SSXFF'),'2850',null,'30',null,null);
Insert into SCOTT."EMP" (EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COMM,DEPTNO,T,D) values ('7782','CLARK','M
ANAGER','7839',to_timestamp('09.06.81','DD.MM.RR HH24:MI:SSXFF'),'2450',null,'10',null,null);
```

Neues mit 12*c* Release 2

Codierung 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.

Wo steht die neueste Oracle Version zur Verfügung? Die Funktionen werden nach und nach in den entsprechenden Cloud Service Editionen bzw. in den downloadbaren Software Releases verfügbar gemacht. Stand heute (September 2016) ist 12*c* Release 2 in der sogenannten **Exadata Express Cloud Edition** (siehe „Weitere Informationen“) verfügbar. Dieser Cloud Service ist speziell für Entwickler geeignet und konzipiert.

Nimmt man die Features für Entwickler von Oracle Database 12*c* Release 2 in Augenschein, so stellt man fest, dass auch hier die Weiterentwicklung rund um **JSON** in der Datenbank ein wichtiges Thema bleibt. Neu sind beispielsweise die Möglichkeiten JSON Dokumente in materialisierten Views zu verwenden, zu partitionieren und/oder zusätzlich In-Memory zu speichern. Die Neuerung einen JSON "Dataguide" zu definieren, der die Struktur und den Inhalt der JSON Dokumente erfasst, eröffnet dabei ganz neue Möglichkeiten: So können diese Informationen beispielsweise persistent als Teil einer JSON Search Index Infrastruktur gespeichert werden und damit schnelle Ad-hoc Abfragen bzw. Volltextsuchen ermöglichen. Zusätzlich besteht darüberhinaus die Möglichkeit Views und Projektionen auf spezielle JSON Felder zu definieren.

Auch der gesamte Bereich des **Approximate Query Processing** wurde stark erweitert. So gibt es jetzt nicht nur die schon bekannte Funktion APPROX_COUNT_DISTINCT, sondern darüber hinaus viele weitere Funktionen wie beispielsweise APPROX_COUNT_DISTINCT_AGG,

APPROX_MEDIAN, APPROX_PERCENTILE usw. Ein wichtiges Ziel ist dabei, den Einsatz ohne Änderung des existierenden Codes zu ermöglichen. So gibt es neue Session- bzw. System- Initialisierungsparameter, die die exakte Funktion einfach durch die korrespondierende "approx" Funktion ersetzt.

Auch im Bereich **Spaltensortierung bzw. Casesensitivität** gibt es einige Vereinfachung für die Programmierung. So wird es möglich sein, bei der Spaltendefinition schon zu Beginn Caseinsensitivität zu definieren oder eine binäre bzw. linguistische Spaltenreihenfolge mitzugeben. Interessant könnte dies dann für die Verwendung von Function based Indizes sein. Bisher war es immer wichtig in der Applikation sicherzustellen, dass die Sessioneinstellungen und somit die Zugriffe auf den Index gewährleistet werden; dies wird nun mit diesem Feature vereinfacht.

Eine weitere Vereinfachung liefern die Erweiterungen zu den Funktionen LISTAGG bzw. CAST oder die neue Funktion VALIDATE_CONVERSION. Das Ziel ist beispielsweise Fehler bei der Konvertierung mit der CAST Funktion über eine spezielle Ausgabe zu steuern oder den Überlauf bei der sehr nützlichen Funktion LISTAGG zu kontrollieren.

Dies sind nur einige ausgesuchte Beispiele von Neuerungen in 12.2. Möchte man einen Überblick über alle Neuigkeiten bekommen, eignen sich wie immer die einzelnen **What's New** Abschnitte in den entsprechenden Handbüchern.

Fazit

Performantes Programmieren mit weniger Code ist nur ein Vorteil, den man aus der Verwendung von neuen Features gewinnen kann. Zum leichteren Verständnis/Erlernen kann man dabei zum Beispiel die Code Library in Oracle Live SQL verwenden: die kleinen Tutorials erklären häufig auch neue Features. Auch die Werkzeuge wie SQL Developer oder die neue Linemode Variante SQLcl bieten gerade beim Testen hilfreiche Funktionen. 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. SQL Developer und SQLcl und natürlich auch andere Werkzeuge bieten in der Regel einfache Möglichkeiten sich mit den Oracle Cloud Umgebungen zu verbinden.

Generell ist aber wichtig: man sollte wissen, was man programmiert hat bzw. welche Auswirkungen der Code haben kann. Daher empfehle ich immer den eigenen Code zu überprüfen, nicht nur das Ergebnis sondern, wenn möglich, sich auch ein Bild von der Ausführungsperformance bzw. vom Ausführungsplan zu machen.

Weitere Informationen

Handbücher und interessante Links

- SQL Language Reference
- Development Guide
- Database PL/SQL Language Reference
- XML DB Developer's Guide (12.1.0.2) Kapitel 13
- Exadata Express Cloud Service:
  https://docs.oracle.com/cloud/latest/exadataexpress-cloud/index.html
- Oracle Dojos: tinyurl.com/dojoonline (speziell Dojo #11 und #12)
- Oracle Datenbank und Cloud Community Blog: blogs.oracle.com/dbacommunity_deutsch

- Oracle Live SQL: http://livesql.oracle.com
- SQLcl Download von OTN:
  http://www.oracle.com/technetwork/developer-tools/sqlcl/overview/index.html

Kontaktadresse

Ulrike Schwinn
Oracle BU Database & Cloud Technologies

ORACLE Deutschland B.V. & Co. KG
Riesstr 25, 80992 München
Telefon: +49 89 1430 1865
E-Mail Ulrike.Schwinn@oracle.com
Internet: http://blogs.oracle.com/dbacommunity_deutsch
https://twitter.com/uschwinn