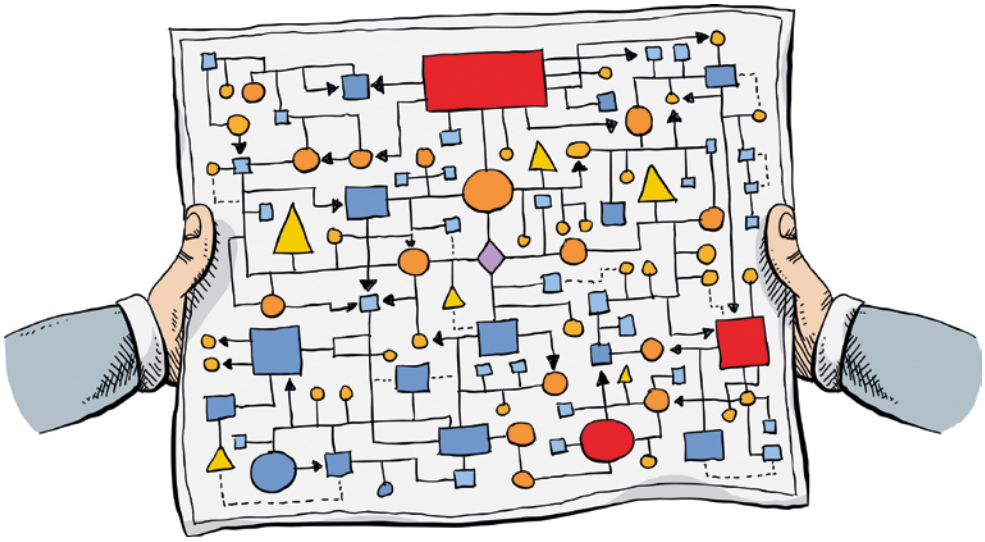

Warum kompliziert, wenn es auch einfach geht: (SQL-) Funktionen in der Oracle-Datenbank

Ulrike Schwinn, ORACLE Deutschland B.V. & Co. KG

Programmiersprachen kommen und gehen. Nur wenige Sprachen, die heute in Gebrauch sind, haben Wurzeln, die wie bei SQL über Jahrzehnte zurückreichen. Der SQL-Umfang wächst mit jedem Oracle-Datenbank-Release. Die Neuerungen erweitern dabei nicht nur die Funktionalität, sondern helfen auch, die Programmierung zu erleichtern und dabei sogar die Performance zu erhöhen.

Viele Anfragen – darunter auch komplexe – lassen sich mittlerweile mit einfachen SQL-Mitteln lösen. Man benötigt keine anderen Programmiersprachen oder gar zusätzlichen Code. Ein wichtiges Ziel muss dabei immer sein, dass die Performance nicht unter der zunehmenden Funktionalität leidet, sondern ganz im Gegenteil die Abfragen performant erfolgen. Grund genug, in den folgenden Abschnitten ein paar interessante und vielleicht auch noch unbekannte SQL-Funktionen aus unterschiedlichen Datenbank-Releases vorzustellen, die genau diese Ziele verfolgen.

SQL-Konstrukte, die jeder kennen sollte

Ein einfaches, häufig verwendetes Abfragekonstrukt, mit dem jeder Datenbank-Entwickler irgendwann konfrontiert wird, sind die Top-N-Abfragen und das Blättern in Applikationen. Lange Zeit waren das Arbeiten mit „ROWNUM“ und die geschickte Nutzung einer Subquery dazu erforderlich, wie das Beispiel in *Listing 1* mit dem altbekannten Demoschema „SH“ und der Tabelle „PRODUCTS“ zeigt. Die Aufgabe besteht darin, die fünf kostengünstigsten Produkte aufzulisten.

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

Listing 1

```
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;
```

Listing 2

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

Listing 3

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

Listing 4

```
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
```

Listing 5

```
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
...
```

Listing 6

```
SQL> select * from emp;
  EMPNO ENAME      JOB         MGR     HIREDATE      SAL     COMM    DEPTNO
------- ---------- ---------  ------ --------- -------- -------- --------
   7369 SMITH      CLERK      7902   17-DEC-80     800                20
   7499 ALLEN      SALESMAN   7698   20-FEB-81     1600     300        30

14 rows selected.
SQL> set linesize window
SQL> set feedback on sql_id
SQL> r
  1* select * from emp

  EMPNO ENAME      JOB         MGR     HIREDATE      SAL     COMM    DEPTNO
------- ---------- ---------  ------ --------- -------- -------- --------
   7369 SMITH      CLERK      7902   17-DEC-80     800                20
   7499 ALLEN      SALESMAN   7698   20-FEB-81     1600     300        30
…
14 rows selected.

SQL_ID: a2dk8bdn0ujx7
```

Listing 7

Nun soll um fünf Einträge weitergeblättert werden. Auch das lässt sich mit einer Erweiterung der Abfrage leicht bewerkstelligen. Setzt man zusätzlich noch den Result Cache für die sich wiederholenden Abfragen ein, kann man schnell einen Performance-Vorteil gewinnen *(siehe Listing 2)*.

Die Verwendung des Result Cache lässt sich mit dem Hint „RESULT_CACHE“ oder über (Session- beziehungsweise System-) Parameter einschalten und ist eine einfache und sehr effiziente Methode, das Ergebnis von komplexen Abfragen in einem speziellen Cache vorzuhalten. Sobald sich das Ergebnis ändert, wird der Cache automatisch invalidiert. Daher sollte man überprüfen, für welche Abfragen die Einstellung sinnvoll ist, und nicht unbedingt nach dem Gießkannen-Prinzip den Cache für alle Abfragen reservieren. Diese Art von Abfragen lässt sich ab der Version 12c viel übersichtlicher mit der sogenannten „Row Limiting“- Klausel programmieren. Für die fünf kostengünstigsten Produkte ergibt sich dann der SQL-Code in *Listing 3*. Es werden einfach die Schlüsselworte „fetch first <Anzahl Zeilen> rows only“ angefügt. Für das Blättern der nächsten fünf Zeilen ergibt sich die erweiterte Syntax mit „offset“ *(siehe Listing 4)*.

Eine weitere interessante Funktionalität, die das Programmieren mit SQL vereinfacht, ist die Unterstützung der „LATERAL“-Klausel ab 12c – übrigens eine Funktion aus dem „SQL:2016“-Standard. Es geht darum, Tabellen außerhalb einer Inline-View (Korrelation) zu referenzieren. Ein einfaches Beispiel in 11g R2 zeigt die Problematik: Es sollen die Angestellten, ihre Gehälter und die Durchschnittsgehälter der zugehörigen Abteilung ausgegeben werden. Ein erster Versuch scheitert mit einem Fehler *(siehe Listing 5)*. Diese Aufgabe lässt sich allerdings lösen. Mit 12c und der neuen „LATERAL“-Klausel ist dies ganz einfach mit einer einzigen SQL-Abfrage möglich *(siehe Listing 6)*.

Das Beispiel zeigt einen Join mit zwei Operanden. Der zweite Operand ist eine Lateral-Inline-View, gekennzeichnet durch das Schlüsselwort „LATERAL“, die die Tabelle „SCOTT.emp e“ in der „WHERE-Klausel“ nutzen kann, ohne einen Fehler zu erzeugen.

Monitoring von Ausführungsplänen

Es gibt in der Oracle-Datenbank viele Möglichkeiten (wie Views und Reports), die dabei helfen, die Ausführungspläne der Statements zu monitoren. Lästig ist

dabei häufig die Notwendigkeit, die „SQL_ID“ herauszufinden. Mit SQL Developer oder der neuesten „SQL*Plus“-Version ist dies nun einfach möglich. Mit „SET FEEDBACK ON SQL_ID“ beispielsweise lässt sich in SQL*Plus-18c ganz einfach die „SQL_ID“ des abgelaufenen Statements anzeigen. Kein langes Suchen im „V$SQL“ oder anderen Views ist erforderlich. Auch bei der Ausgabe-Formatierung in SQL*Plus gibt es weitere Erleichterungen: Die Ausgabe lässt sich nun wie bei der grafischen Variante an die Bildschirmgröße anpassen. Dazu muss nur der Wert für „SET LINESIZE“ auf den neuen Wert „WINDOW“ gesetzt werden *(siehe Listing 7)*.

Für die Ausführungspläne der Abfragen gibt es das nützliche PL/SQL-Package „DBMS_XPLAN“, um Berichte über die Durchführung von Abfragen zu erstellen. Einfacher geht es mit dem SQL Developer, der nicht nur die „SQL_ID“ anzeigen kann, sondern auch den Ausführungsplan im eigenen Viewer ausgibt beziehungsweise die Unterschiede von zwei Ausführungsplänen hervorheben kann. Auch Informationen zur „AUTOTRACE“-Funktion sind möglich, sofern die Privilegien diesen Zugriff gestatten. Dabei werden sogar sogenannte „Hotspots“ angezeigt, die man genauer analysieren sollte. In der (aktuellen) Version 18.2 (Stand August 2018) des SQL Developer lassen sich die Ausführungspläne sogar in der „DBMS_XPLAN“-Sichtweise anzeigen *(siehe Abbildung 1)*.

Übrigens ist es in 12.2 auch einfacher, mögliche SQL-Statements in PL/SQL-Pro-

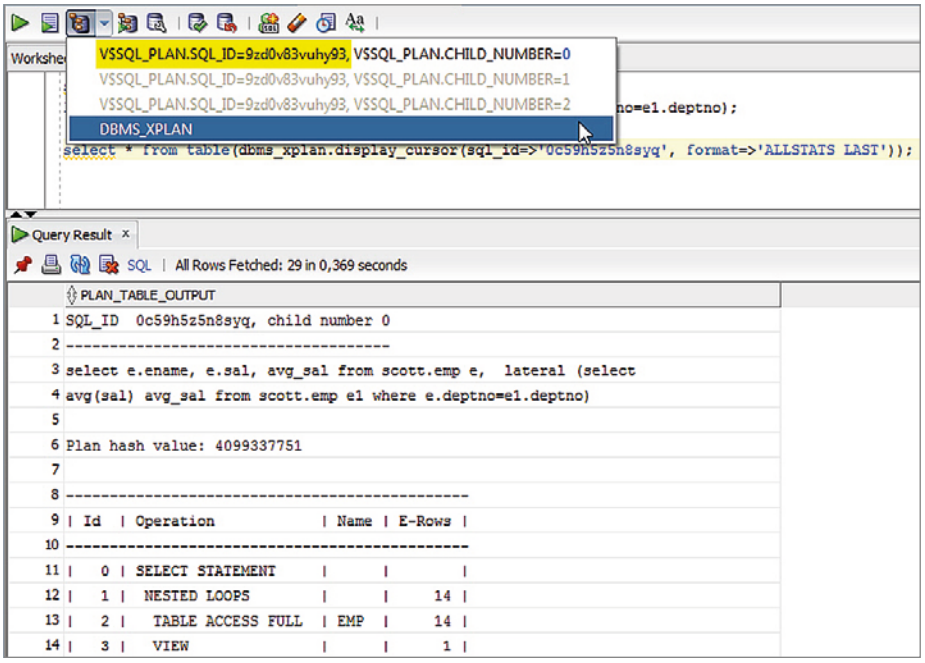

Abbildung 1: „DBMS_XPLAN“-Ausführungsplan im SQL Developer 18.2

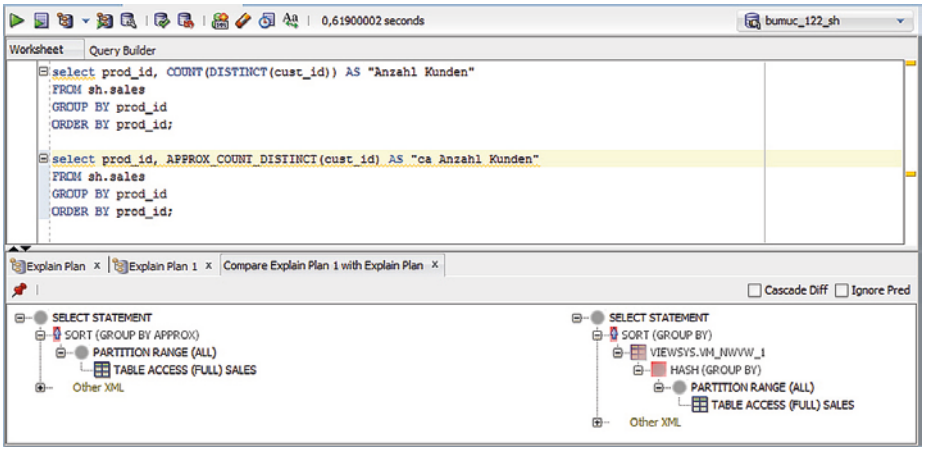

Abbildung 2: Vergleich zweier Ausführungspläne im SQL Developer

```
SQL> CREATE VIEW daily_prod_count AS
  2   SELECT t.calendar_year year,
  3       t.calendar_month_number month,
  4       t.day_number_in_month day,
  5       APPROX_COUNT_DISTINCT_DETAIL(s.prod_id) daily_detail
  6   FROM sh.times t, sh.sales s
  7   WHERE t.time_id = s.time_id
  8   GROUP BY t.calendar_year, t.calendar_month_number, t.day_number_in_month;

View created.

SQL> select * from daily_prod_count where rownum=1;

      YEAR      MONTH        DAY
---------- ---------- ----------
DAILY_DETAIL
--------------------------------------------------------------------------------
      2000          1          1
0D0C04CC0006000000000000000600000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000
```

Listing 8

grammen aufzuspüren. Der PL/SQL Hierachical Profiler, der immer noch eines der wichtigsten Werkzeuge darstellt, um Bottlenecks zu finden und Performance-Tuning in PL/SQL durchzuführen, zeigt jetzt automatisch die zugehörigen SQL-IDs und das Statements (also SQL-Text) an. So ist es einfach möglich, den zugehörigen SQL-Code zu analysieren.

Eine ähnliche Idee steckt auch hinter der PL/Scope-Erweiterung. Diese sammelt während der Kompilierung Informationen über die Verwendung der Identifier und speichert diese automatisch in Data-Dictionary-Tabellen. Aufgezeichnet werden Typen, Namen, Nutzung und Vorkommen der Identifier im PL/SQL-Code. So ist deren einfache Analyse im Code möglich. Der Parameter „PLSCOPE_SETTINGS" aktiviert die Nutzung und stellt sie ein. In 12.2 lassen sich damit auch SQL-Statements mitloggen. Dazu werden neue Werte wie „IDENTIFIERS:SQL, STATEMENTS:ALL", „IDENTIFIERS:ALL, STATEMENTS:ALL" und „IDENTIFIERS:PLSQL, STATEMENTS:NONE" bereitgestellt. Die neue View „USER_STATEMENTS" listet dabei die SQL-Statements auf.

SQL für analytische Anfragen

Standard-SQL kann sicherlich viele oder auch die meisten Datenbank-Anfragen beantworten. Wenn es allerdings um Fragen wie zum Beispiel „Wie hoch ist die laufende Summe der Mitarbeitergehälter?" geht, ist dies mit Basis-SQL-Syntax nicht einfach zu lösen, da Berechnungen Zeile für Zeile durchgeführt werden müssen. Analytische Funktionen (auch „Window-Funktionen" genannt) fügen SQL-Erweiterungen hinzu, die solche Operationen nicht nur schneller ablaufen lassen, sondern die auch einfacher zu programmieren sind. Im Internet stehen viele Beispiele dazu, etwa im Oracle Magazine oder in den Live-SQL-Tutorials.

In diesem Zusammenhang ist eine neue Kategorie von Funktionen eingeführt worden – die sogenannten „Approximate-Funktionen", die besonders dann sinnvoll sind, wenn keine exakten Werte benötigt werden und Näherungen akzeptabel sind. Schon seit Release 12.1 gibt es die Funktion „APRROX_COUNT_DISTINCT", die eine Alternative zu „COUNT(DISTINCT)" darstellt. Sie verarbeitet große Datenmengen deutlich schneller, wie man bei der Analyse der beiden Ausführungspläne in *Abbildung 2* erkennen kann – mit vernachlässigbarer Abweichung vom exakten Ergebnis.

Weitere Funktionen wie zum Beispiel „APPROX_COUNT_DISTINCT_DETAIL", „APPROX_COUNT_DISTINCT_AGG" und „TO_APPROX_COUNT_DISTINCT" helfen seit 12.2 dabei, einmalig Ressourcen-intensive Näherungszahlberechnungen durchzuführen, die resultierenden Details zu speichern und anschließend effiziente Aggregationen und Abfragen zu diesen Details durchzuführen. „APPROX_COUNT_DISTINCT_DETAIL(expr)" ist dabei eine Art Helper-Funktion, die Näherungswerte für „expr" berechnet und einen „BLOB"-Wert, das sogenannte „Detail", in einem speziellen Format zurückliefert.

Dies können hochgranulare Details sein, wie etwa demografische Zählungen der Städte oder tägliche Verkaufszahlen, die in einer resultierenden Detail-Tabelle oder (materialisierten) View gespeichert werden. Das Beispiel in *Listing 8* berechnet die Produkt-Verkaufszahlen pro Tag mit „APPROX_COUNT_DISTINCT_DETAIL“.

Die von „APPROX_COUNT_DISTINCT_DETAIL“ zurückgegebenen Details können dann als Input für die Funktion „APPROX_COUNT_DISTINCT_AGG“ dienen, um Aggregationen der Details durchzuführen. Dies können Details geringerer Granularität sein, wie etwa demografische Zählungen oder monatliche Verkaufszahlen. Die Funktion „TO_APPROX_COUNT_DISTINCT“ wandelt dann die Ergebnisse aus „APPROX_COUNT_DISTINCT_DETAIL“ oder „APPROX_COUNT_DISTNCT_AGG“ in ein lesbares Format um *(siehe Listing 9)*.

Korrespondierende Funktionen stehen übrigens auch im Zusammenhang mit „PERCENTILE“ und „MEDIAN“ zur Verfügung, also „APPROX_PERCENTILE_DETAIL“, „APPROX_PERCENTILE_DETAIL” und „TO_APPROX_PERCENTILE”. Um den Einsatz auch ohne Änderung des bestehenden Codes zu ermöglichen, sind mit 12.2 zusätzliche Initialisierungsparameter wie „APPROX_FOR_COUNT_DISTINCT“ und „APPROX_FOR_PERCENTILE“ eingeführt worden. Sind diese aktiviert, wird die exakte Funktion einfach durch die korrespondierende „Approximate“-Funktion ersetzt.

Die Idee, weitere Näherungsfunktionen zur Verfügung zu stellen, um die Performance bei Berechnungen speziell im Massendaten-Umfeld zu erhöhen, wurde auch in 18c weiterverfolgt. So gibt es die neuen „Window“-Funktionen „APPROX_RANK“, „APPROX_COUNT“ und „APPROX_SUM“. „APPROX_RANK“ gibt dabei den Rang in einer Gruppe von Werten zurück. Hier ist allerdings eine bestimmte Syntax-Vorgabe erforderlich. Um die Funktionsweise kurz zu demonstrieren, nehmen wir die Tabelle „EMP“ als Beispiel. Es werden jeweils pro Abteilung die Top-10-Jobs ausgegeben. Für jeden Job sind das Gesamtgehalt und das Ranking angegeben *(siehe Listing 10)*.

```
SQL> SELECT year,
  TO_APPROX_COUNT_DISTINCT(APPROX_COUNT_DISTINCT_AGG(daily_detail))
yearly_detail
  FROM daily_prod_count
  GROUP BY year order by year;

      YEAR YEARLY_DETAIL
---------- -------------
      1998            60
      1999            72
      2000            72
      2001            71
```

Listing 9

```
SQL> SELECT deptno, job, APPROX_SUM(sal),
  APPROX_RANK(PARTITION BY deptno ORDER BY APPROX_SUM(sal) DESC) rk
  FROM emp
  GROUP BY deptno, job
  HAVING APPROX_RANK(PARTITION BY deptno ORDER BY APPROX_SUM(sal)
DESC) <= 10;

    DEPTNO       JOB APPROX_SUM(SAL)          RK
---------- --------- --------------- ----------
        10     CLERK            1300          3
        10   MANAGER            2450          2
        10 PRESIDENT            5000          1
        20     CLERK            1900          3
        20   MANAGER            2975          2
        20   ANALYST            6000          1
        30     CLERK             950          3
        30   MANAGER            2850          2
        30  SALESMAN            5600          1
```

Listing 10

```
CREATE PRIVATE TEMPORARY TABLE ORA$PTT_sales_ptt_session
  (time_id       DATE,
  amount_sold    NUMBER(10,2))
  ON COMMIT PRESERVE DEFINITION;
```

Listing 11

Weitere Neuigkeiten in Oracle Database 18c

Lange ersehnt und endlich realisiert ist nun die Möglichkeit, mit privaten temporären Tabellen zu arbeiten. Um eine solche Tabelle (PTT) zu erzeugen, ist eine gewisse Namenskonvention einzuhalten: Der Name beginnt immer mit einem Präfix (Default „ORA$PTT“), der über den Initialisierungsparameter „PRIVATE_TEMP_TABLE_PREFIX“ festgelegt ist und geändert werden kann. Wie bei der globalen temporären Tabelle gibt es auch hier eine Transaktions- und eine Session-spezifische Variante – allerdings mit einigen Unterschieden. Die Transaktions-spezifische Variante, die den Default darstellt oder mit den Schlüsselworten „DROP DEFINITION“ erzeugt wird, speichert Daten nur bis zum Ende einer Transaktion. Die Session-spezifische Variante *(siehe Listing 11)* wird mit „PRESERVE DEFINITION“ angelegt. In beiden Fällen erfolgt dabei kein implizites Commit.

Externe Tabellen (auch External Tables) sind aus der Oracle-Datenbank nicht mehr wegzudenken. Sie sind seit jeher ein geeignetes Mittel, um auf Daten einfach zuzugreifen, die in Flat-Files außerhalb der Datenbank gespeichert sind. Erforderlich sind dazu ein logisches Directory zur Lokalisierung der Daten und ein Access-Treiber wie „ORACLE_LOADER“, „ORACLE_DATAPUMP“, „ORACLE_HIVE“ oder „ORACLE_HDFS“.

In der Datenbank 18c gibt es einige interessante Neuerungen dazu; beispielsweise besteht die Möglichkeit, Inline External Tables zu nutzen. Es wird dabei keine externe Tabelle als persistentes Objekt im Data Dictionary angelegt, sondern einfach das „SELECT“-Kommando um eine neue „inline_external_table“-Syntax in der „FROM“-Klausel erweitert. Dies vereinfacht den Zu-

```
SQL> select * from sales_tab where sales_id=961;

  SALES_ID    PROD_ID    CUST_ID TIME_ID   QUANTITY_SOLD AMOUNT_SOLD
---------- ---------- ---------- --------- ------------- -----------
       961      11160      17450 01-JUN-18            20         800
Execution Plan
----------------------------------------------------------
Plan hash value: 1909132751
-------------------------------------------------------------------------------------------------------
| Id  | Operation                              | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |              |     1 |    28 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID READ OPTIM| SALES_TAB    |     1 |    28 |     2   (0)| 00:00:01 |
|*  2 |   INDEX UNIQUE SCAN READ OPTIM         | SYS_C0022714 |     1 |       |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("SALES_ID"=961)
```

Listing 12

griff auf externe Daten und ermöglicht die Entwicklung einfacher und effizienter Datenbank-Anwendungen.

Darüber hinaus ist es mit 18c jetzt auch möglich, External-Table-Informationen in Verbindung mit dem In-Memory Column Store zu verwenden. So lassen sich Daten, die in externen Dateien vorliegen, in diesen Column Store im Hauptspeicher laden, um damit alle Vorteile der In-Memory-Funktionen (wie zum Beispiel Performance-Steigerung beim Zugriff) nutzen zu können. Verwendet man den neuen Autonomous Database Cloud Service, gibt es darüber hinaus eine interessante Erweiterung: Externe Daten, die im Objekt-Storage gespeichert sind, können einfach über eine External Table zur Verfügung gestellt werden.

Bei Oracle Database 18c gibt es einen neuen Memory-Bereich, den Memory-optimierten Rowstore (auch „MemOptimized Rowstore" genannt), nicht zu verwechseln mit dem In-Memory Column Store. Konzipiert für hohe Abfrage-Leistungen (wie Internet-of-Things-Workloads), sorgt er für sehr schnelle und latenzarme Abfragen. Key-Value-Lookups auf Basis von Primärschlüsselwerten (mit Abfragefilter „Spalte = Wert") nutzen direkt einen Memory-Hash-Index bei der Ausführung. Das neue Attribut auf Tabellen-Ebene („MEMOPTIMIZE FOR READ") gibt an, welche der Tabellen mit dem neuen Memory-Hash-Index in den Buffer-Cache gepinnt werden sollen. Die Datenbank erzeugt einen Hash-Index und pflegt ihn automatisch. Das Beispiel in *Listing 12* zeigt eine Ausführung mit dem Memory-Hash-Index.

Das Feature „MemOptimized Rowstore" ist im Moment auf Exadata und dem Oracle Database Cloud Service „Enterprise Edition Extreme Performance" verfügbar. Weitere Informationen dazu stehen im Handbuch „Licensing Information User Manual".

Neue, aussagekräftige Optimizer-Hints wie „OPTIMIZER_IGNORE_HINTS" und „OPTIMIZER_IGNORE_PARALLEL_HINTS", die speziell für das Arbeiten mit der Autonomous Database eingeführt wurden, erleichtern ab 18c das Testen von Queries mit Hints. Der Default ist in den On-Premises-Installationen natürlich „FALSE". Möchte man wissen, wie eine Abfrage ohne Hints ausgeführt wird, können diese Parameter in der Session verändert werden. Mehr Details dazu und zu weiteren 18c-Features stehen auf „blogs.oracle.com/coretec" in der Kategorie „18c".

Fazit

Sich mit SQL-Funktionen auseinanderzusetzen und neue Funktionen zu erlernen kann viele Vorteile bringen. Zum Erlernen eignet sich beispielsweise die Code Library in „Oracle Live SQL:", die kleinen Tutorials erklären häufig neue Features. Auch die Werkzeuge wie SQL Developer oder die neue Linemode-Variante „SQLcl" bieten gerade beim Testen hilfreiche Funktionen.

Ob die Funktion in der Cloud oder in ihrer eigenen Umgebung (soweit verfügbar) getestet wird, spielt dabei übrigens keine Rolle; die Funktionen bleiben die gleichen. Dabei bieten SQL Developer, SQLcl und auch andere Werkzeuge in der Regel einfache Möglichkeiten, sich mit den Oracle-Cloud-Umgebungen zu verbinden.

Man sollte sich allerdings im Klaren darüber sein, was man programmiert hat beziehungsweise welche Performance-Auswirkungen die Änderung am eigenen Code haben kann. Daher sollte vor jedem produktiven Einsatz nicht nur die Überprüfung des Ergebnisses stehen, sondern man sollte, wenn möglich, sich auch ein Bild von der Ausführungs-Performance beziehungsweise vom Ausführungsplan machen.

Handbücher und interessante Links

- SQL Language Reference
- Development Guide
- Licensing Information User Manual
- Oracle Magazine: A Window into the World of Analytic Functions: *https://blogs.oracle.com/oraclemagazine/a-window-into-the-world-of-analytic-functions*
- Oracle Dojos: *tinyurl.com/dojoonline*
- Oracle-Datenbank und Cloud Community Blog: *blogs.oracle.com/coretec*
- Oracle Live SQL: *http://livesql.oracle.com*

Ulrike Schwinn
ulrike.schwinn@oracle.com