Inside Oracle11g Release 2:

Neues für den Anwendungsentwickler

Carsten Czarski
ORACLE Deutschland GmbH

Einleitung

Oracle11g bringt für nahezu alle Bereiche der Anwendungsentwicklung, unabhängig von der verwendeten Programmiersprache, neue Features mit. Dieser Beitrag kann daher nur eine kleine Auswahl der Neuerungen präsentieren, weiterführende Informationen finden sich im Oracle Technet oder in den Handbüchern. So vermittelt ein Blick in die SQL Language Reference einen ersten Eindruck über die große Anzahl der neuen SQL-Kommandos und Funktionen. Die neuen Möglichkeiten des Online Application Upgrade ("PL/SQL Editions") werden in einem separaten Artikel behandelt.

SQL-Funktionen: LISTAGG, NTH_VALUE, Recursive WITH-Klausel

Im Bereich der SQL- und PL/SQL-Funktionen bietet Oracle11*g* Release 2 vor allem Neuerungen im Detail. Eine Ausnahme ist **Online Application Upgrade** ("PL/SQL-Editions"), das eine grundlegend neue Technologie ist: Mit diesem Feature, für das es einen eigenen Artikel gibt, kann die ununterbrochene Verfügbarkeit von Applikationen gewährleistet werden, auch wenn neue Versionen eingespielt werden.
Die neue Funktion **LISTAGG**, eine Aggregatsfunktion für VARCHAR2-Spalten, dürfte für viele Entwickler eine Erleichterung sein. Das Zusammenfassen von VARCHAR2-Spalten einer Tabelle wurde bislang meist mit eigenem PL/SQL-Code realisiert. Listing 1 zeigt die Verwendung:

```
select
 deptno,
 listagg(ename, ':') within group (order by ename) ename_list
from emp
group by deptno

   DEPTNO ENAME_LIST
--------- ----------------------------------------
       10 CLARK:KING:MILLER
       20 ADAMS:FORD:JONES:SCOTT:SMITH
       30 ALLEN:BLAKE:JAMES:MARTIN:TURNER:WARD
```

Listing 1: LISTAGG-Funktion

Im Bereich der analytischen Funktionen erlaubt **NTH_VALUE** die Selektion des "n-ten" Wertes einer sortierten Reihe. So selektiert Listing 2 zu jeder Zeile der Tabelle **EMP** das zweithöchste Gehalt der jeweiligen Abteilung (**DEPTNO**)

```
select
  ename, deptno, sal,
  nth_value(sal, 2) over (
    partition by deptno
    order by sal desc
    range between unbounded preceding and unbounded following
  ) "2ND_BEST"
from emp

ENAME      DEPTNO       SAL   2ND_BEST
---------- ---------- ---------- ----------
KING           10       5000       2450
CLARK          10       2450       2450
MILLER         10       1300       2450
SCOTT          20       3000       3000
FORD           20       3000       3000
```

Listing 2: Die NTH_VALUE-Funktion selektiert den "n-ten" Wert einer sortierten Reihe

Auch in Oracle11*g* Release 2 setzt Oracle die Unterstützung der einschlägigen Industriestandards nahtlos fort. Das manifestiert sich unter anderem an der neuen "*Recursive WITH"*-Klausel für hierarchische Abfragen. Diese "neue" Syntax ist zwar länger, im Vergleich zum bekannten START WITH – CONNECT BY entspricht sie jedoch dem SQL99-Standard.
Listing 3a und Listing 3b zeigen den Unterschied. Das bedeutet natürlich nicht, dass man bestehende Anwendungen umschreiben sollte – die bekannte und erprobte Syntax bleibt weiterhin gültig. Für neue Anwendungen empfiehlt sich allerdings die Nutzung der Standard-Syntax.

```
select empno, ename, mgr
from emp
start with mgr is null
connect by prior empno = mgr
```

Listing 3a: Hierarchische Abfrage mit der START WITH – CONNECT BY-Syntax

```
with emp_rec (empno, ename, mgr) as (
  select empno, ename, mgr
  from emp where mgr is null
  union all (
    select c.empno, c.ename, c.mgr
    from emp_rec p, emp c
    where p.empno = c.mgr
  )
)
select * from emp_rec
```

Listing 3b: Hierarchische Abfrage mit der SQL99-konformen "recursive WITH clause"

Content API, Database Filesystem und Image Watermarking

Für unstrukturierte Daten wurden bereits mit Oracle11*g* Release 1 die *SecureFiles* eingeführt. Oracle SecureFiles ist die neue Technologie für *Large Objects*, also BLOB- und CLOB-Spalten – sie zeichnet sich durch bessere Performance und mehr Funktionen aus.
Da unstrukturierte Daten fast immer nach Dateisystem-Konzepten, also Dateien und Ordnern, organisiert werden, bringt Oracle11*g* Release 2 die **Content API** und das **Database Filesystem** mit. Abbildung 1 zeigt das Konzept.

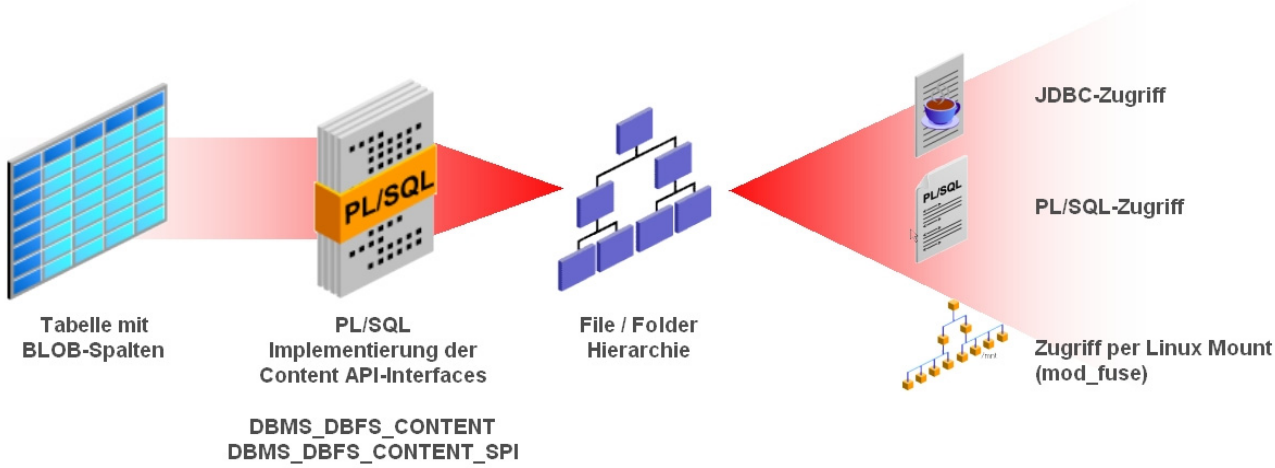

Abbildung 1: Die Content API stellt eine Dateisystem-Sicht auf eine Tabelle mit Dokumenten bereit

Die PL/SQL-Pakete **DBMS_DBFS_CONTENT** und **DBMS_DBFS_CONTENT_SPI** beschreiben die Schnittstelle (*Interface*) für das Database Filesystem. Eine Implementierung, die sich an diese Schnittstelle hält, kann für beliebige Tabellen eine abstrakte Dateisystem-Sicht erzeugen.
Mit dem *Securefile Store* bringt die Datenbank eine eigene Implementierung mit, die sofort genutzt werden kann. Ein solcher Securefile Store wird per SQL-Skript erzeugt: **$ORACLE_HOME/rdbms/admin/dbfs_create_filesystem.sql** nimmt einen Tablespace-Namen und einen frei wählbaren Namen für das "Dateisystem" entgegen. Im angegebenen Tablespace werden dann die Datentabellen für den Securefile Store erstellt. Das Skript **dbfs_create_filesystem_advanced.sql** erlaubt zusätzlich die Konfiguration von Tabelleneigenschaften wie Partitionierung, Verschlüsselung oder Komprimierung.

```
SQL> start ?/rdbms/admin/dbfs_create_filesystem.sql USERS MYFS
```

Nach Ausführung des Skripts befinden sich zwei Tabellen im Datenbankschema. Die Tabelle für die Dateien bekommt den Namen **T_*{Dateisystemname}*** (hier: **T_MYFS**). Die BLOB-Spalte **FILEDATA** enthält die konkreten Daten, die anderen Spalten nehmen Dateisystem-typische Metadaten auf.
Auf Linux-Systemen kann das Dateisystem nun mit Hilfe des Executables **dbfs_client** eingebunden werden (Listing 4). Voraussetzung ist, dass **fuse** (*Filesystem in Userspace)* auf dem System installiert und das Kernelmodul **mod_fuse** geladen ist. Das Handbuch "SecureFiles and Large Objects Developer's Guide" enthält nähere Informationen hierzu.

```
$ dbfs_client {username}@{dbhost}:{port}/{service} \
  -o allow_other
  -o rw
  /mnt
```

Listing 4: Einbinden (mount) des Database Filesystems in den Verzeichnisbaum eines Linux-Servers

Nach dem Einbinden können die Dateien mit normalen Betriebssystem-Mitteln bearbeitet und gelesen werden. In der Datentabelle **T_MYFS** finden sich anschließend entsprechende Tabellenzeilen. Wie bereits erwähnt, kann auch mit PL/SQL auf dem "Dateisystem" gearbeitet werden: Als Schnittstellen-Package dient dabei stets **DBMS_DBFS_CONTENT**.
Die Implementierung eines solchen Database Filesystems ist völlig frei: Für jede Dateisystem-Operation (**mkdir**, **ls**, **cat**, **rm**, ...) wird PL/SQL-Code implementiert, der die Einträge in eigenen Tabellen entsprechend pflegt. Abbildung 2 zeigt die Variante eines "Metadatengenerators".

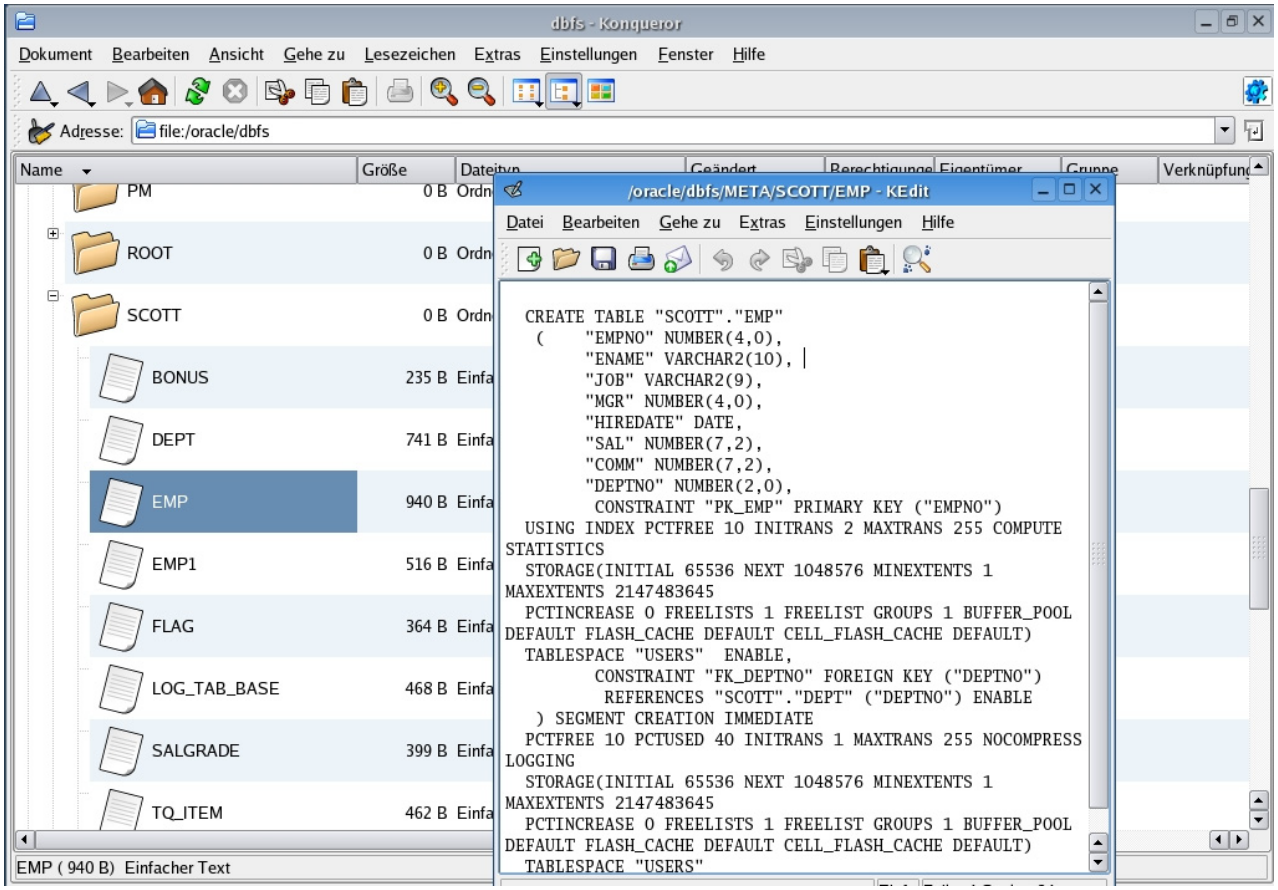

Abbildung 2: Fallstudie: Nutzung des Database Filesystem als "Metadatengenerator"

Oracle MultiMedia stellt in 11*g* Release 2 unter anderem eine Funktion für das *Image Watermarking* bereit. Damit können in der Datenbank gespeicherte Bilder mit einem "Wasserzeichen" (zum Beispiel ein Copyright-Vermerk) versehen werden (Abbildung 3). Das ist insbesondere nützlich, wenn Bilder im Internet publiziert werden – in Kombination mit

einem Trigger kann sichergestellt werden, dass alle Bilder einen geeigneten Vermerk bekommen.

Abbildung 3: Der Text wurde mit "Image Watermarking" ins Bild gesetzt

Listing 5 zeigt den PL/SQL-Code, mit dem das Wasserzeichen in Abbildung 3 generiert wurde:

```
-- Parameter für das "Wasserzeichen"
props := ordsys.ord_str_list(
  'font_name=Times New Roman',
  'font_style=bold',
  'font_size=40',
  'text_color=green',
  'position=bottomright',
  'transparency=0.6'
);

-- Wasserzeichen hinzufügen
ORDSYS.ORDImage.applyWatermark(
  imageblob          => src_blob,
  added_text         => 'Oracle 11g Release 2',
  dest               => dst_blob,
  logging            => logging,
  watermark_properties => props
);
```

Listing 5: PL/SQL-Code für Image Watermarking

Deferred Segment Creation, Präprozessor für externe Tabellen

Primär für den DBA vorgesehen, doch auch für jeden Entwickler wichtig, ist die neue Klausel zur verzögerten Erstellung von Segmenten im CREATE TABLE-Kommando. Bislang allokiert Oracle bereits beim Erstellen einer Tabelle Extents für diese. Erstellt man nun sehr viele Tabellen, so wird initial sehr viel Plattenplatz allokiert. Mit der neuen Klausel **SEGMENT CREATION DEFERRED** kann dies verzögert werden: Nach dem CREATE TABLE-Kommando ist (noch) kein Extent allokiert, das geschieht erst beim Einfügen der ersten Zeile: Leere Tabellen belegen also keinen Platz mehr. Für partitionierte Tabellen steht dieses Feature vorerst allerdings nicht zur Verfügung. Passend dazu belegen auch Indizes, die *UNUSABLE* sind, in Oracle11*g* Release 2 keine Extents und somit keinen Plattenplatz mehr.
Externe Tabellen gibt es seit Oracle9*i*. Eine externe Tabelle ist eine Datei, welche die Datenbank wie eine Tabelle zur Verfügung stellt. Die Struktur der Datei wird der Datenbank mit einer SQL*Loader-ähnlichen Syntax bekannt gemacht. Natürlich können solche Tabellen nur gelesen werden, und Indizes sind nicht möglich.
Neu in 11*g* Release 2 ist die Möglichkeit, einen *Präprozessor* für eine externe Tabelle anzugeben – so kann man bspw. das Executable **unzip** als Präprozessor festlegen, so dass die Datenbank komprimierte Dateien direkt verarbeiten kann.

Weitere neue Funktionen und Möglichkeiten

Wie jedes Release bringt auch Oracle11*g* Release 2 neue PL/SQL-Pakete mit. Eine kleine Auswahl sei hier vorgestellt:

- **UTL_MATCH** erlaubt das Bestimmen der Ähnlichkeit zweier Zeichenketten zueinander. Dabei werden die sog. "Levenshtein-Distanz" und der "Jaro-Winkler Algorithmus" unterstützt. Listing 6 zeigt ein Beispiel.
- **DBMS_METADATA_DIFF** ermöglicht das Bestimmen von Unterschieden zwischen zwei Datenbankschemas mit PL/SQL. Werkzeuge wie der SQL Developer oder der Enterprise Manager können dies schon länger – nun steht ein Interface für den Programmierer bereit. Zur Nutzung muss das *Change Management Pack* lizenziert werden.
- **DBMS_PARALLEL_EXECUTE** macht es schließlich möglich, UPDATE-Kommandos auf eine Tabelle parallel auszuführen.

```
select utl_match.JARO_WINKLER_SIMILARITY(
  'Carsten Czarski', 'Thomas Czarski'
) similarity
from dual;

       SIMILARITY
--------------------
              79

select utl_match.JARO_WINKLER_SIMILARITY(
  'Carsten Czarski', 'Ulrike Schwinn'
) similarity
```

```
from dual;

      SIMILARITY
-----------------
              49
```

Listing 6: Anwendungsbeispiel für das UTL_MATCH-Paket

Weitere Informationen

An erster Stelle seien hier die deutschsprachigen Communities zu nennen – dort werden die Funktionen des neuen Release in den nächsten Monaten nach und nach auch detailliert erläutert:

- Community für APEX und PL/SQL-Entwickler
  http://www.oracle.com/global/de/community
- DBA Community
  http://www.oracle.com/global/de/community/dbadmin

Weiterhin sind das Oracle Technet und die Online-Dokumentation bei Fragen und Problemen nützliche Ratgeber:

- Oracle Technet: Application Development
  http://otn.oracle.com/products/database/application_development
- Oracle Dokumentation
  http://www.oracle.com/pls/db112/portal.portal_db?selected=3

Kontakt:

Carsten Czarski
ORACLE Deutschland GmbH

E-Mail: Carsten.Czarski@oracle.com
Blog: http://sql-plsql-de.blogspot.com
http://oracle-text-de.blogspot.com

Ulrike Schwinn
ORACLE Deutschland GmbH

Ulrike.Schwinn@oracle.com