Oracle 12c für Entwickler

Markus Fiegler, ORDIX AG

Die neue Datenbank-Version 12*c* bringt Erweiterungen und Verbesserungen in den Bereichen „Anwendungssicherheit", „Funktionalität", „Kompatibilität" und „Performance". Dabei helfen die Neuerungen, eine PL/SQL-Anwendung sicherer, schneller, aber auch effektiver – also mit weniger Code – zu implementieren.

Automatische Inkrementierung von Werten

Das Generieren von eindeutigen Werten, etwa für eine Schlüssel-Tabellenspalte, ist in Oracle 12*c* deutlich einfacher. So können dafür zum einen „DEFAULT"-Constraints mit Sequenzen, zum anderen die sogenannten „Identity Columns" verwendet werden. Die Lösung mit einem „DEFAULT"-Constraint setzt eine explizit angelegte Sequenz voraus. Diese wird anschließend beim Anlegen eines „DEFAULT"-Constraint auf einer Schlüssel-Tabellenspalte mit der altbewährten „NEXTVAL"-Funktion verwendet. Identity Columns stellen eine weitere neue Möglichkeit zur Generierung von eindeutigen Werten in einer Spalte dar. Die Besonderheit bei dieser Variante ist, dass dafür keine explizit angelegten Sequenzen notwendig sind, denn diese werden von Oracle implizit erzeugt und verwaltet.

Der wesentliche Vorteil der beiden neuen Möglichkeiten der automatischen Inkrementierung von Werten ist die einfache Nutzung, da die „NEXTVAL"-Funktion einer Sequenz nicht mehr explizit aufgerufen werden muss. Eine Lösung mit Trigger und Sequenz pro Tabelle, die vor 12*c* häufig angewendet wurde, ist damit nicht mehr notwendig. *Abbildung 1* stellt die Ausprägungen der Identity Columns und die Unterschiede zum „DEFAULT"-Constraint mit einer Sequenz dar.

Blättern im SQL-Ergebnis: „OFFSET" und „FETCH"

Soll eine Teilmenge aus einer Ergebnismenge abgerufen werden, so kann die neue „OFFSET"- und „FETCH"-Syntax verwendet werden. Dabei besteht die Möglichkeit, eine bestimmte Anzahl von Datensätzen einer Ergebnismenge über die „OFFSET"-Klausel überlesen zu können. Die „LIMIT"-Klausel („FETCH FIRST|NEXT") gibt an, wie viele Datensätze aus der Ergebnismenge ausgegeben werden sollen *(siehe Abbildung 2)*. Die aufwändige Alternative mit Unterabfragen und „ROWNUM"-Attribut, die bis zur Version 11*g* für derartige Szenarien verwendet wurde, ist ab 12*c* nicht mehr notwendig.

Verbesserte native „OUTER JOIN"-Syntax

Bei der nativen „OUTER JOIN"-Syntax gab es bis 11*g* die Einschränkung, dass zwei oder mehrere Tabellen mit einer anderen Tabelle nicht über einen „OUTER JOIN" verknüpft werden konnten. In so einem Fall musste dann zum Beispiel die „ANSI OUTER JOIN"-Syntax herhalten. Ab 12*c* fällt diese Einschränkung bei der nativen „OUTER JOIN"-SQL-Syntax weg.

„ANSI JOIN"-Erweiterungen

Im Bereich der „ANSI"-Syntax sind in der Datenbank 12*c* drei neue Sprachelemente hinzugekommen: „CROSS APPLY", „OUTER APPLY" und „LATERAL". Diese neue Syntax zeichnet sich dadurch aus, dass ein sogenannter „Left Correlation Support" möglich ist. Mit diesem kann man beispielsweise aus einer Inline-View in der „FROM"-Klausel einen Bezug auf andere Tabellen aus der „FROM-Klausel" vornehmen. Ergebnismengen lassen sich somit schon in der „FROM"-Klausel einschränken anstatt erst in der „WHERE"-Klausel, wie es bis 11*g* nur möglich war. Zusätzlich können bei der „CROSS APPLY"- und „OU-

	kein Wert	Wert z.B. 5	NULL	DEFAULT	
DEFAULT MA_SEQ.NEXTVAL (ext. Seq.)	Seq.-Wert	5	NULL	Seq.-Wert	
DEFAULT ON NULL MA_SEQ.NEXTVAL (ext. Seq.)	Seq.-Wert	5	Seq.-Wert	Seq.-Wert	NOT NULL Cons.
GENERATED **ALWAYS AS IDENTITY**	Seq.-Wert	**ORA-32795**	**ORA-32795**	Seq.-Wert	
GENERATED **BY DEFAULT AS IDENTITY**	Seq.-Wert	5	**ORA-01400**	Seq.-Wert	
GENERATED **BY DEFAULT ON NULL AS IDENTITY**	Seq.-Wert	5	Seq.-Wert	Seq.-Wert	

Abbildung 1: Ausprägungen der Identity Columns und die Unterschiede zum „DEFAULT"-Constraint

nr	name
1	ma1
2	ma2
3	ma3
4	ma4
5	ma5

```
SELECT ... ORDER BY ...
FETCH FIRST 2 ROWS ONLY

SELECT ... ORDER BY ...
OFFSET 2 ROWS FETCH NEXT 2
ROWS ONLY
```

Abbildung 2: Die „LIMIT"-Klausel

```
CREATE TABLE projekt(
  nr      NUMBER,
  name    VARCHAR(30),
  PERIOD FOR prj_zeit
);
```

Abbildung 3: Die „PERIOD FOR"-Klausel

```
CREATE TABLE ma (
  nr       number,
  name     varchar2(20)
) ROW ARCHIVAL;
```

Abbildung 5: „IN-DATABASE ARCHIVING"

TER APPLY"-Syntax Collections verwendet werden, ohne die „TABLE"-Funktion angeben zu müssen.

Unsichtbare Spalten

Ab 12*c* können Spalten sowohl in Tabellen als auch in Views unsichtbar gemacht werden. Die Besonderheit daran ist, dass diese bei Operationen wie dem „DESCRIBE"-Befehl in SQL*PLUS in einem PL/SQL-„ROWTYPE" oder bei SQL-Statements ohne explizite Spaltenangabe wie „SELECT * " nicht berücksichtigt beziehungsweise nicht angezeigt werden. Sollen die unsichtbaren Spalten bei SQL-Statements verwendet werden, müssen diese explizit aufgeführt werden.

Mit den unsichtbaren Spalten können bestehende Tabellen um neue Spalten erweitert werden, ohne dass diese Auswirkungen auf bestehende Anwendungen haben. Außerdem können unsichtbare Spalten etwa für technische Partitionierungsspalten, für Audit-Spalten oder für Spalten verwendet werden, die für andere Datenbank-User oder Anwendungen nicht so sehr in den Vordergrund treten sollen.

Temporal Validity

Mit Temporal Validity lässt sich eine implizite Betrachtung des Gültigkeitszeitraums realisieren, die für die Anwendung völlig transparent ist. Dabei werden nur die Datensätze einer Tabelle selektiert, die zu einem vorgegebenen Zeitpunkt gültig waren. Um Temporal Validity nutzen zu können, muss beim Anlegen einer Tabelle die „PERIOD FOR"-Klausel angegeben sein *(siehe Abbildung 3)*. Damit werden unter anderem zwei unsichtbare Spalten für die Gültigkeitszeitraum-Betrachtung (Start- und End-Datum) mit dem Datentyp „TIMESTAMP WITH TIME ZONE" angelegt, die manuell mit Werten gefüllt werden müssen.

Anschließend können beim Selektieren der Daten aus so einer Tabelle über die einfache Flashback-Query-Syntax die zu einem bestimmten Zeitpunkt beziehungsweise in einem bestimmten Zeitraum gültigen Datensätze ermittelt werden. Die Besonderheit dabei ist, dass die beiden unsichtbaren Gültigkeitszeitraum-Spalten in der „WHERE"-Klausel nicht explizit angegeben werden müssen. Diese fügt Oracle implizit im Hintergrund dem explizit abgesetzten SQL-Statement hinzu.

Außerdem besteht auf der Session-Ebene die Möglichkeit, den Gültigkeitszeitpunkt über die Prozedur „ENABLE_AT_VALID_TIME" aus dem Package „DBMS_FLASHBACK_ARCHIVE" festzulegen. Auch hier werden alle SQL-Statements, die auf einer Tabelle mit einer „PERIOD FOR"-Klausel abgesetzt wurden, implizit von Oracle um die Einschränkung auf den Gültigkeitszeitraum der beiden unsichtbaren Spalten erweitert *(siehe Abbildung 4)*.

Sind Gültigkeitszeitraum-Spalten bereits vorhanden, so kann „Temporal Validity" mit expliziter Angabe von Start- und Enddatum-Spalten bei der „PERIOD FOR"-Klausel verwendet werden. Mit dieser Neuerung lässt sich also eine Historisierung der Tabellendaten für die Anwendung transparent einführen, ohne die Anwendung anpassen zu müssen.

IN-DATABASE ARCHIVING

„IN-DATABASE ARCHIVING" macht Datensätze vor der Applikation transparent unsichtbar beziehungsweise archiviert diese, ohne die Datensätze zu löschen oder die Applikation anpassen zu müssen. Die Funktion wird über die „ROW ARCHIVAL"-Syntax beim Anlegen einer Tabelle aktiviert *(siehe Abbildung 5)*. Dabei legt Oracle eine unsichtbare Spalte „ORA_ARCHIVE_STATE" an, über die gesteuert wird, ob ein Datensatz aktiv (sichtbar) oder archiviert (unsichtbar) ist. Falls die Spalte den Wert „0" hat, handelt es sich um einen aktiven Datensatz. Alle Werte ungleich „0" deuten auf einen archivierten Datensatz hin *(siehe Abbildung 6)*.

SQL Pattern Matching

Mit SQL Pattern Matching können Muster beziehungsweise Zusammenhänge in Tabellendaten zeilenübergreifend gefunden werden, die durch einfaches Aggregieren nicht zu finden sind. Typische Anwendungsfälle sind die Auswertung einer Log-Datei bezüglich Web-Session-Klicks,

prj_zeit_start	prj_zeit_end	nr	name
01.06.2014	10.06.2014	1	prj1
10.06.2014	20.06.2014	2	prj2
	20.06.2014	3	prj3
10.06.2014		4	prj4
20.06.2014	25.06.2014	5	prj5

```
DBMS_FLASHBACK_ARCHIVE.ENABLE_
AT_VALID_TIME(level => 'ASOF',
query_time=>to_date('15062014'
... ;
SELECT * FROM projekt;
```

```
DBMS_FLASHBACK_ARCHIVE.ENABLE_
AT_VALID_TIME(level => 'ASOF',
query_time=>to_date('22062014'
... ;
SELECT * FROM projekt;
```

Abbildung 4: Die „PERIOD FOR"-Klausel

ora_archive_state	nr	name
0	10	Meier
0	11	Hansen
1	12	Baier

```
ALTER SESSION SET
  ROW ARCHIVAL VISIBILITY = ACTIVE;
SELECT * FROM ma;
```

```
ALTER SESSION SET
  ROW ARCHIVAL VISIBILITY = ALL;
SELECT * FROM ma;
```

Abbildung 6: Archivierter Datensatz

Bereiche	Beschreibung
MATCH_RECOGNIZE (	… FROM … **<MATCH_RECOGNIZE>** WHERE …
row_pattern_partition_by	Unterteilung der Daten in Partitionen
row_pattern_order_by	Sortierung der Daten innerhalb der Partitionen
row_pattern_measures	Ausgabe-Felder und Berechnungen
row_pattern_rows_per_match	Rückgabe der Zeilen (eine Zeile oder alle Zeilen pro Mustertreffer)
row_pattern_skip_to	Wiederaufnahme der Suche nach einem Mustertreffer
PATTERN (row_pattern) [row_pattern_subset_clause]	Musterdefinition, z.B. PATTERN (GELB ROT)
DEFINE row_pattern_definition_list)	Mustervariablen, z.B. ROT AS rot.farbe = 'ROT'

Abbildung 7: SQL Pattern Matching

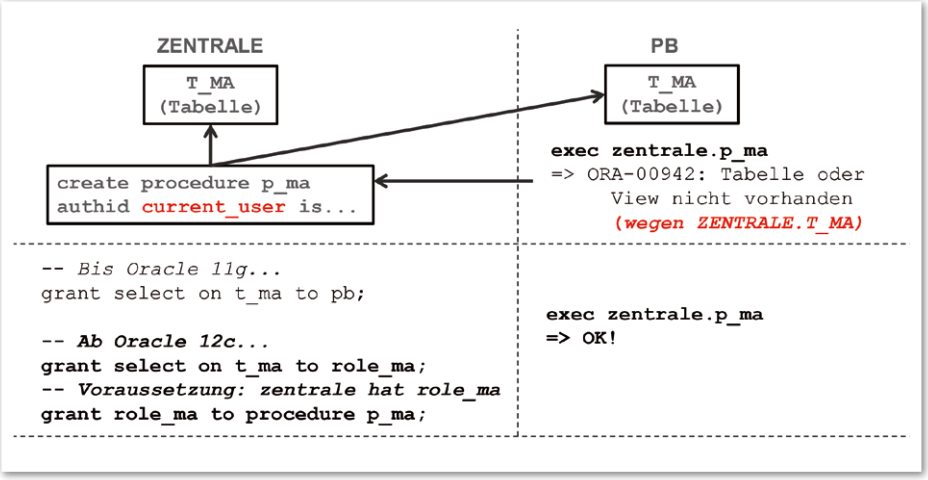

Abbildung 8: Rollen an PL/SQL-Objekte vergeben

die Analyse des Aktienkursverlaufs oder bestimmte Zusammenhänge bei Sensordaten. Bei all diesen Anwendungsfällen reicht die einfache Betrachtung einzelner Datensätze oder das klassische Aggregieren von Datensätze nicht mehr aus. Das SQL Pattern Matching bietet genau dafür, als Ergänzung zu den analytischen Funktionen, ein geeignetes Werkzeug, das rein deklarativ gesteuert wird *(siehe Abbildung 7)*.

Rollen an PL/SQL-Objekte vergeben

Im Bereich der Berechtigungsvergabe besteht die Möglichkeit, Rollen an PL/SQL-Objekte zu vergeben. Der Vorteil dieser Neuerung ist, dass beispielsweise bei einem PL/SQL-Objekt mit „invoker rights" der Aufrufer über keine expliziten Berechtigungen mehr verfügen muss, falls das PL/SQL-Objekt auf Aufrufer-fremde Datenbank-Objekte zugreifen sollte *(siehe Abbildung 8)*. In so einem Fall reicht es aus, wenn dem PL/SQL-Objekt eine Rolle mit den Berechtigungen auf die Aufrufer-fremden Datenbank-Objekte vergeben wird. Damit lassen sich Privilegien feingranularer an PL/SQL-Objekte vergeben, ohne die Rechte an den Aufrufer selbst vergeben zu müssen. Der Aufrufer nutzt die an ein PL/SQL-Objekt vergebene Rolle dabei nur während der Ausführung des PL/SQL-Objekts.

Vererbung von Rechten bei Views an PL/SQL-Funktionen

Bis zur Version 11*g* wurden PL/SQL-Funktionen, die aus einer View heraus aufgerufen wurden, unabhängig von der „DEFINER"- oder „CURRENT_USER"-Klausel in der PL/SQL-Funktion immer mit den „DEFINER"-Rechten ausgeführt. Ab 12*c* legt die neue Syntax „BEQUEATH [CURRENT_USER | DEFINER]" bei der View fest, ob die PL/SQL-Funktion, die aus einer View heraus aufgerufen wird, mit „DEFINER"- oder „CURRENT_USER"-Rechten ausgeführt werden soll. Die Rechte werden also quasi von der View weiter an die in der View vorhandene PL/SQL-Funktion vererbt.

Zugriffsbeschränkung mit „ACCESSIBLE BY"-Klausel

Die neue „ACCESSIBLE BY"-Klausel realisiert bei PL/SQL-Objekten eine Zugriffssteuerung in Form einer Whitelist. Diese Restriktion gilt übrigens auch für den Datenbank-Benutzer „SYS", der über keine Sonderrechte verfügt, um die Zugriffssteuerung der Whitelist zu umgehen.

Die „ACCESSIBLE BY"-Klausel stellt sicher, dass Hilfsobjekte, die gezielt für ein Programm geschrieben wurden, auch nur von diesem Programm aufgerufen werden können. Damit lassen sich Hilfsobjekte vor Unbefugten und vor Ausführung in einem falschen Kontext schützen *(siehe Abbildung 9)*.

Das „UTL_CALL_STACK PL/SQL"-Package

Mit dem neuen „UTL_CALL_STACK"-Package stellt Oracle eine neue Schnittstelle zur Verfügung, um Call-Stack-, Error-Stack- und Error-Backtrace-Informationen zu ermitteln. Der Vorteil gegenüber den altbekannten Funktionen aus dem „DBMS_UTILITY"-Package wie „FORMAT_CALL_STACK", „FORMAT_ERROR_STACK" und „FORMAT_ERROR_BACKTRACE" besteht darin, dass auf die einzelnen Ausgaben der Funktionen strukturiert zugegriffen werden kann, ohne die Ausgaben parsen zu müssen.

Das Package kann also genutzt werden, um etwa eine Auswertung über die in einer Programmeinheit geworfenen Fehler zu erstellen. Eine andere Anwendungsmöglichkeit könnte eine Art „bedingtes Exception Handling" sein, bei dem in Abhängigkeit vom Call-Stack (Aufrufhierarchie) ein unterschiedliches Verhalten im Fehlerfall implementiert werden kann. Zudem lassen sich mit dem „UTL_CALL_STACK"-Package die Namen von aufgerufenen Subprogrammen bei einem Call-Stack ausgeben. Somit können der Call-Verlauf und die Call-Stack-Informationen von verschachtelten Objekten leichter nachvollzogen werden.

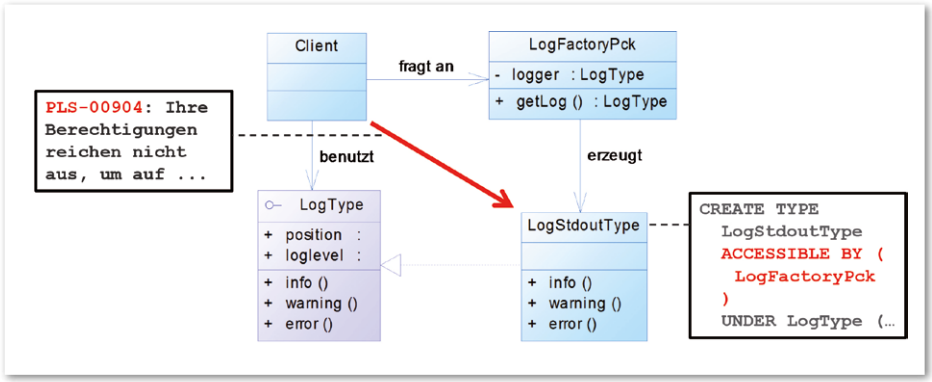

Abbildung 9: Die „ACCESSIBLE BY“-Klausel

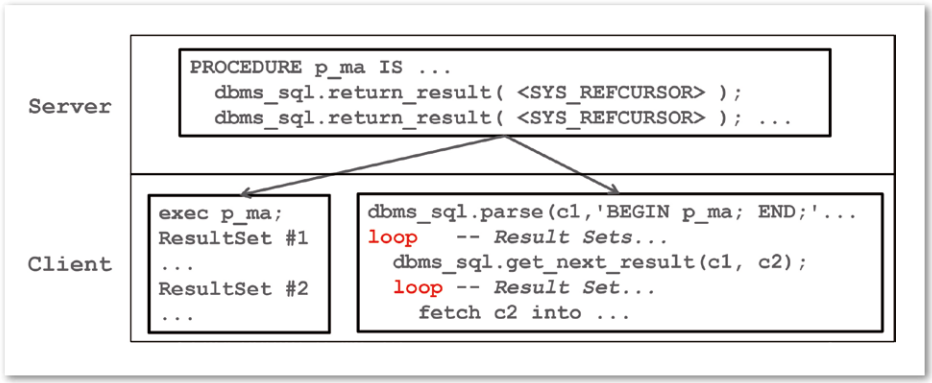

Abbildung 10: Die „GET_NEXT_RESULT“-Prozedur

Die „PL/SQL WITH“-Klausel

Die Klausel stellt eine Erweiterung der herkömmlichen „SQL WITH“-Klausel dar und ermöglicht die Verwendung von PL/SQL-Code in einem „SELECT“-Statement. Die Ausführung ist im Vergleich zu einem „SELECT“-Statement mit einer Stored Function deutlich schneller, da bei einem „PL/SQL WITH SELECT“-Statement der Kontextwechsel zwischen der SQL- und der PL/SQL-Engine optimiert ist.

Sollen bestehende Stored Functions, die in einem SQL-Statement verwendet werden, in der Ausführungszeit beschleunigt werden, kann ab 12*c* die neu eingeführte „PRAGMA UDF“ (User Defined Function) genutzt werden. Der Vorteil dieser „PRAGMA“ ist wie bei der „PL/SQL WITH“-Klausel die Minimierung des Kontextwechsels zwischen der SQL- und der PL/SQL-Engine.

Implicit Result Sets

Für die Übergabe und Rückgabe von Result Sets stellt Oracle „REF CURSOR“ zur Verfügung. Damit können Result Sets entweder explizit über die Signatur einer Programmeinheit oder, wie es ab Oracle 12*c* auch möglich ist, implizit von einer Programmeinheit an die andere Programmeinheit mithilfe der „RETURN_RESULT“-Prozedur aus dem „DBMS_SQL“-Package übergeben werden. Der wesentliche Vorteil der impliziten Result-Sets-Übergabe ist die Tatsache, dass die Anzahl und Art der Result Sets erst zur Laufzeit bestimmt werden kann und damit eine sehr flexible Schnittstelle zwischen zwei Programmeinheiten realisiert werden kann. Die aufrufende Programmeinheit hat dann anschließend die Möglichkeit, mithilfe der „GET_NEXT_RESULT“-Prozedur aus dem „DBMS_SQL“-Package die Result Sets einzeln abrufen und verarbeiten zu können *(siehe Abbildung 10).*

Data Redaction

Mit Data Redaction besteht die Möglichkeit, sensible Daten bei der Ausgabe unkenntlich zu machen. Dabei findet die Maskierung der Daten in der Datenbank statt und ist für die Anwendung völlig transparent. Der Vorteil ist unter anderem, dass eine Anwendung um eine Maskierungsfunktionalität erweitert werden kann, ohne die Anwendung anpassen zu müssen.

Ein weiteres Plus von Data Redaction besteht darin, dass die Maskierung im Gegenteil zu einer Maskierungsfunktionalität, die in der Anwendung implementiert ist, nicht nur für eine bestimmte Anwendung, sondern für alle möglichen Zugriffe von einem Datenbank-User auf die Datenbank gilt. Die Aktivierung von Data Redaction findet dabei rein deklarativ in Form einer sogenannten Policy statt („DBMS_REDACT“-Package). Unter anderem die Verwendung von Datenbank-Views, um eine Art Maskierung vor Version 12*c* zu erreichen, kann damit entfallen.

Fazit

Die Datenbank-Version 12*c* bringt viele interessante und hilfreiche Neuerungen in den Bereichen „Anwendungssicherheit“, „Funktionalität“, „Kompatibilität“ und „Performance“. Viele Problem-Szenarien, die bis 11*g* mit aufwändigem Coding realisiert werden mussten, können jetzt mit relativ einfach überschaubaren SQL-Befehlen umgesetzt werden.

Besonders hervorzuheben ist die Möglichkeit einer für die Applikation transparenten Funktionserweiterung, etwa mit Temporal Validity, IN-DATABASE ARCHIVING oder Data Redaction, ohne die Anwendung selbst anpassen zu müssen. Damit ist in der Anwendung weniger Code zu implementieren, was zu einer besseren Wartbarkeit und einer deutlich geringeren Fehleranfälligkeit führt.

Markus Fiegler
info@ordix.de