Das Upgrade kritischer Applikationen kann schwierig sein. Eines der Hauptprobleme hinsichtlich der Verfügbarkeit liegt darin, dass lange Ausfallzeiten nicht regelmäßig eingeplant werden können. Aus diesem Grund ist es für solche Applikationen wünschenswert, Online-Upgrades zu implementieren. Dies erfordert, dass sowohl die fragliche Applikation als auch jegliche Software, die von der Applikation genutzt wird (etwa die Datenbank), Online-Upgrades unterstützen.

Edition-Based Redefinition

Christian Antognini, Trivadis AG

Oracle hat dieses Problem schon seit Jahren erkannt. Unglücklicherweise sind bis einschließlich 11*g* R1 nur eine beschränkte Anzahl von Funktionalitäten hinsichtlich dieser Anforderungen implementiert. Erst mit 11*g* R2 hat sich diese Situation geändert. Mit der sogenannten „Edition-Based Redefinition"-Funktionalität bietet die Oracle-Datenbank nun endlich eine echte Unterstützung für Online-Upgrades an. Der Artikel liefert einen Überblick und einige Tipps zu diesem neuen Feature.

In einer Oracle-Datenbank wird ein Objekt durch seinen Namen, den zugehörigen Benutzer (Schema) und seine Edition identifiziert. Ziel des Ganzen ist, und das ist neu mit 11*g* R2, mehrere Kopien (Versionen) eines bestimmten Objekts zu unterstützen. Die Objekte, für die mehrere Kopien angelegt werden können, heißen „Editioned Objects". Objekte, für die das nicht zutrifft beziehungsweise die keine Edition haben, werden als „Noneditioned Objects" bezeichnet; die Spalte „EDITION" in den Data Dictionary Views enthält dann den Wert „NULL".

Das Ziel, zwei oder mehrere Versionen eines bestimmten Objekts zu unterstützen, ist einfach und klar: Während die Applikation eine Gruppe von Objekten benutzt, kann man eine andere Gruppe derselben Objekte nicht nur redefinieren, sondern auch für Testzwecke nutzen. Es ist also möglich, eine Gruppe von Objekten zu redefinieren und in einer isolierten Umgebung zu testen, bevor diese live gehen.

Nicht alle, nur die folgenden Objekt-Typen unterstützen dieses neue Konzept, das heißt, sie sind „Editionable Object Types":

- FUNCTION
- LIBRARY
- PACKAGE und PACKAGE BODY
- PROCEDURE
- SYNONYM (PUBLIC SYNONYM ist ein „Noneditionable Object Type")
- TRIGGER
- TYPE und TYPE BODY
- VIEW

Im Wesentlichen sind dies alle Objekt-Typen, die durch die erweiterte Funktionalität des Datenbank-Kernels über PL/SQL-Code oder externe Libraries genutzt werden. Die einzigen Ausnahmen sind die Views und die Synonyme. Man beachte, dass weder Java-bezogene noch datenspeichernde Objekt-Typen (wie Tabellen und Indizes) editionierbar sind.

Alle Objekte, die keine Editionen unterstützen, werden „Noneditionable Object Types" genannt.

Editionserstellung für einen Benutzer ermöglichen

Der erste Schritt für die Nutzbarkeit des „Edition-Based Redefinition"-Features ist, den Eigentümer der zu redefinierenden Objekte freizuschalten. Nur Benutzer mit Berechtigung zur Editionserstellung können editionierte Objekte erstellen. Um dies zu ermöglichen, muss entweder die „ENABLE EDITIONS"-Klausel beim Erstellen eines Benutzers angegeben sein oder für einen bereits bestehenden Benutzer der „ALTER USER"-Befehl, wie im folgenden Beispiel dargestellt, ausgeführt werden.

```
SQL> ALTER USER cha ENABLE EDI-
TIONS;
```

Es ist wichtig zu erwähnen, dass die Berechtigung zur Editionserstellung nicht wieder entzogen werden kann. Mit anderen Worten, so etwas wie eine „DISABLE EDITIONS"-Klausel existiert nicht. Ob ein Benutzer die Möglichkeit hat, editionierbare Objekte zu erstellen, zeigt folgende Abfrage:

```
SQL> SELECT editions_enabled
  2  FROM dba_users
  3  WHERE username = ‚CHA';

EDITIONS_ENABLED
----------------
Y
```

Ist ein Benutzer dazu berechtigt, muss zur Identifizierung einer spezifischen Version von Datenbank-Objekten eine Edition angelegt werden.

Erstellen von Editionen

Jede Datenbank muss mindestens eine Edition haben. Dafür wird beim Anlegen einer neuen Datenbank eine Basis-Edition, die sogenannte „ORA$BASE" erstellt. Eine Datenbank unterstützt mehrere Editionen. Sie sind mithilfe einer Root-Edition über eine Hierarchie organisiert (Standardmäßig ORA$BASE) und durch eine Eltern-Kind-Beziehung mit den anderen Editionen verbunden. Abbildung 1 zeigt eine Datenbank mit vier Editionen, der Basis-Edition und drei benutzerdefinierten Editionen. Jede der drei letzteren bezieht sich in unserem Beispiel auf ein Applikations-Release mit den Bezeichnungen REL1, REL2 und REL3.

Für das Erstellen einer Edition ist der „CREATE EDITION"−Befehl er-

Abbildung 1: Editionen werden über Hierarchien organisiert

forderlich. Um dieses SQL-Statement auszuführen, ist das „CREATE ANY EDITION"−Systemprivileg notwendig. Für das Anlegen der Editionen aus Abbildung 1 müssen folgende SQL-Befehle ausgeführt werden. Die Eltern-Kind-Beziehung ist über die „AS CHILD OF"-Klausel spezifiziert.

```
SQL> CREATE EDITION rel1 AS
CHILD OF ora$base;
SQL> CREATE EDITION rel2 AS
CHILD OF rel1;
SQL> CREATE EDITION rel3 AS
CHILD OF rel2;
```

Standardmäßig ist eine neue Edition nur über den anlegenden Benutzer oder durch den SYS-Benutzer verwendbar. Man kann die nötigen Privilegien an andere Anwender über das „USE ON EDITION"-Privileg vergeben. Wie der folgende SQL-Befehl zeigt, können die Privilegien entweder an einen bestimmten oder an alle Benutzer über einen „GRANT TO PUBLIC" vergeben werden.

```
SQL> GRANT USE ON EDITION rel2
TO cha;
SQL> GRANT USE ON EDITION rel3
TO PUBLIC;
```

Das Anzeigen von Informationen hinsichtlich aller verfügbaren Editionen kann entweder über die View „ALL_EDITIONS" oder über „DBA_EDITIONS" selektiert werden. Das folgende Beispiel zeigt eine hierarchische Abfrage für einen solchen Zweck. Man beachte, dass Editionen nicht zu einem bestimmten Schema gehören und deshalb die beiden Views keine Spaltenspezifizierung eines Benutzers zu einer Edition zeigen.

```
SQL> SELECT *
  2  FROM all_editions
  3  CONNECT BY parent_edition_name = PRIOR edition_name
  4  START WITH parent_edition_name IS NULL
  5  ORDER BY level;

EDITION_NAME  PARENT_EDITION_NAME  USABLE
------------- -----------------
ORA$BASE                          YES
REL1          ORA$BASE            YES
REL2          REL1                YES
REL3          REL2                YES
```

Da eine Datenbank mehrere Editionen unterstützt, muss ein Benutzer (oder wahrscheinlicher eine Applikation) in der Lage sein, die Edition zu referenzieren, welche die korrekte Version der editionierten Objekte enthält. Mit anderen Worten, die sogenannte „Session Edition" muss ausgewählt sein.

Auswählen von Editionen

Die Auswahl einer Session Edition, von APIs (wie OCI und JDBC) oder Tools (wie SQL*Plus und Data Pump) ermöglichen die Spezifizierung während der Erstellung einer neuen Datenbank. Es ist zu beachten, dass, falls keine Edition angegeben wird, die Basis-Edition (definiert durch die Datenbankeinstellung des Parameters DEFAULT_EDITION) genutzt wird. Standardmäßig ist die Einstellung auf „ORA$BASE" gesetzt. Wie bei allen anderen Datenbank-Eigenschaften können diese über das „ALTER DATABASE"-Kommando geändert werden:

```
SQL> ALTER DATABASE DEFAULT
EDITION = rel1;

SQL> SELECT property_value
  2  FROM database_properties
  3  WHERE property_name = ‚DE
       FAULT_EDITION';

PROPERTY_VALUE
---------------
REL1
```

Dabei sind besonders zwei Punkte einer solchen Konfiguration hervorzuheben:

1. Wenn man die Basis-Datenbank-Edition ändert, vergibt die Datenbank automatisch das „USE ON EDITION"-Privileg für die spezifizierte Edition an „PUBLIC". Dies ist notwendig, weil die Basis-Datenbank-Edition für alle Benutzer verfügbar sein muss.
2. Das Konzept der Basis-Datenbank-Edition ist hauptsächlich aus Kompatibilitätsgründen für Applikationen gedacht, die während eines Aufbaus einer Datenbankverbindung nicht in der Lage sind, eine bestimmte Edition zu setzen. Deshalb sollte die Session Edition wenn möglich immer zum Verbindungszeitpunkt spezifiziert sein.

Ist die Datenbank-Verbindung erst einmal hergestellt, kann man die Session Edition selbstverständlich über ein „ALTER SESSION"-Kommando ändern. Beachten Sie, dass dieses SQL-Kommando nicht ausgeführt werden kann, wenn die aktuelle Session noch eine offene Transaktion beinhaltet. Das folgende Kommando setzt „REL1" als Session Edition:

```
SQL> ALTER SESSION SET EDITION
= rel1;
```

Ebenfalls noch wichtig ist, dass das vorherige SQL-Kommando ein Top-Level-SQL-Kommando ist; daher kann es nicht in PL/SQL ausgeführt werden. Wenn man die Session Edition über PL/SQL ändern möchte (beispielsweise in einem Logon-Trigger), ist die „SET_EDITION_DEFERRED"-Prozedur aus dem „DBMS_SESSION"-Paket zu verwenden. Zwei Methoden sind zur Anzeige der verwendeten Session Edition verfügbar:

1. Nur für die aktuelle Session kann das „SESSION_EDITION_NAME"-Attribut aus dem „USERENV"-Kontext benutzt werden.
2. Für alle Sessions kann die View „V$SESSION" verwendet werden.

Die folgenden Select-Befehle zeigen die unterschiedlichen Abfragemöglichkeiten:

```
SQL> SELECT sys_context(‚USERENV‘,‘SESSION_EDITION_NAME‘) edition_
name
  2  FROM dual;

EDITION_NAME
-------------
ORA$BASE

SQL> SELECT DISTINCT object_name AS edition_name
  2  FROM v$session, dba_objects
  3  WHERE session_edition_id = object_id;

EDITION_NAME
-------------
ORA$BASE
```

Wenn beide – der Eigentümer der editionierten Objekte sowie die korrekte Edition – gesetzt sind, ist es an der Zeit, die editionierten Objekte selbst anzulegen.

Editionierte Objekte erstellen

Editionierte Objekte werden genauso wie alle anderen Objekte angelegt. Das einzige, was gemacht werden muss – neben der Tatsache, dass ein Benutzer verwendet wird, der auch editionierte Objekte besitzen darf –, ist die Edition auszuwählen, mit der das Objekt assoziiert werden soll. Beispielsweise ist das folgende SQL-Kommando für die Erstellung einer Prozedur und die Zuordnung zur Edition „REL1“ möglich:

```
SQL> ALTER SESSION SET EDITION
= rel1;

SQL> CREATE OR REPLACE PROCE
      DURE hello IS
  2  BEGIN
  3    dbms_output.put_
      line(‚Hello from REL1‘);
  4  END;
  5  /
```

Eine andere Methode für das Erstellen von editionierten Objekten besteht darin, die Editionen für einen Benutzer freizuschalten, der bereits Objekte besitzt. In diesem Fall sind alle Objekte automatisch mit der Root-Edition assoziiert, obwohl die Basis-Edition sinnvoller wäre.

Redefinieren editionierter Objekte

Für die Redefinition eines Objekts ist der erste Schritt, eine neue Edition zu erstellen und – falls dieser Schritt nicht durch den Eigentümer des editionierten Objekts ausgeführt wird – die notwendigen Privilegien zu erteilen. Die einzelnen Schritte sind in den vorherigen Abschnitten bereits beschrieben. Wird eine neue Edition erstellt, so werden alle Objekte, die mit der Eltern-Edition assoziiert sind, von der Kind-Edition übernommen. Aber Vorsicht, die Objekte sind nicht mit der Kind-Edition assoziiert. Sie wurden für diese nur sichtbar gemacht. Beispielsweise ist, basierend auf der in Abbildung 1 dargestellten Hierarchie, eine in der REL1-Edition erstellte Prozedur ebenfalls in den Editionen REL2 und REL3 sichtbar (sofern sie nicht entfernt wurde).

Für die Redefinition eines editionierten Objekts muss die neue Edition ausgewählt und anschließend das editionierte Objekt entweder geändert, hinzugefügt, entfernt oder im Fall eines „Stored Programs“ durch reguläre DDL-Kommandos neu kompiliert werden. Beispielsweise wird durch die folgenden SQL-Kommandos die vorher in der REL1-Edition erstellte Prozedur in der REL2-Edition geändert und schließlich in der REL3-Edition gelöscht:

```
SQL> ALTER SESSION SET EDITION
= rel2;

SQL> CREATE OR REPLACE PROCE-
DURE hello IS
  2  BEGIN
  3    dbms_output.put_
      line(‚Hello from REL2‘);
  4  END;
  5  /

SQL> ALTER SESSION SET EDITION
= rel3;

SQL> DROP PROCEDURE hello;
```

Jetzt, da die Prozedur redefiniert ist, sind zwei unabhängige Objekte im Data Dictionary gespeichert. Infolgedessen kann man nun auswählen, welches von beiden Objekten man verwenden möchte. Für diese Zuordnung ist die entsprechende Edition, wie im folgenden Beispiel dargestellt, auszuwählen. Zu beachten ist, dass „USER_OBJECTS“ (das Gleiche gilt auch für „DBA_OBJECTS“ und „ALL_OBJECTS“) die Version anzeigt, die mit der Session Edition assoziiert ist.

```
SQL> ALTER SESSION SET EDITION = rel1;

SQL> SELECT edition_name
  2  FROM user_objects
  3  WHERE object_name = ‚HELLO‘;

EDITION_NAME
-------------
REL1

SQL> EXECUTE hello
Hello from REL1

SQL> ALTER SESSION SET EDITION = rel2;

SQL> SELECT edition_name
  2  FROM user_objects
  3  WHERE object_name = ‚HELLO‘;

EDITION_NAME
-------------
REL2

SQL> EXECUTE hello
Hello from REL2

SQL> ALTER SESSION SET EDITION = rel3;

SQL> SELECT edition_name
  2  FROM user_objects
  3  WHERE object_name = ‚HELLO‘;

no rows selected
```

Wenn man alle Versionen anzeigen möchte, die im Data Dictionary gespeichert sind, sind entweder die Views „USER_OBJECTS_AE“, „DBA_OBJECTS_AE“ oder „ALL_OBJECTS_AE“ zu selektieren. Es ist im folgenden Beispiel zu beachten, dass das mit „REL3“ assozi-

ierte Objekt weiter verfügbar ist, obwohl es als nicht existent (non-existent) markiert ist.

```
SQL> SELECT edition_name, object_type
  2  FROM user_objects_ae
  3  WHERE object_name = ‚HELLO‘;

EDITION_NAME  OBJECT_TYPE
------------- -------------
REL1          PROCEDURE
REL2          PROCEDURE
REL3          NON-EXISTENT
```

Zusammenfassend sind für das Redefinieren editionierter Objekte folgende typische Schritte auszuführen:

- Erstellen der neuen Edition und Setzen einer Session Edition
- Modifizieren der editionierten Objekte und Sicherstellen der Gültigkeit aller Objekte
- Prüfen, ob die Applikation wie erwartet mit der neuen Edition arbeitet
- Dauerhaft auf die neue Edition wechseln

Die bisher beschriebenen Techniken werden benutzt, um Objekte eines editionierbaren Objekt-Typs zu redefinieren. Das Redefinieren von Tabellen – diese sind nicht editionierbar – erfordert die Einführung eines anderen Konzepts, dem sogenannten „View Editionieren“.

Views editionieren

Tabellen sind nicht editionierbar, daher kann man für die Struktur-Redefinition den Vorteil von Editionen nicht nutzen. Stattdessen kann jede Tabelle mit einer sogenannten „Editioning View“ versehen werden, und gleichzeitig können in der Applikation alle Tabellen-Referenzen durch Referenzen auf „Editioning Views“ ersetzt werden. Es ist wahrscheinlich nicht möglich, alle Referenzen zu ersetzen – zum Beispiel kann ein TRUNCATE-Befehl nicht auf eine View ausgeführt werden. So gesehen ist dies nicht unbedingt ein Problem, denn ein TRUNCATE-Befehl arbeitet unabhängig von einer Tabellenstruktur. Vereinfacht gesagt wird eine (transparente) Ebene zwischen der Applikation und dem physikalischen Datenbankdesign erstellt. Eine „Editioning View“ ist eine normale View mit einigen speziellen Eigenschaften:

- Der Zweck einer „Editioning View“ dient nur dem Selektieren von Daten und um optional Aliasse für eine bestimmte Untermenge von Spalten innerhalb einer Tabelle bereitzustellen. Operationen wie Joins, Subquerys, Set Operatoren, Aggregations- und hierarchische Abfragen werden nicht unterstützt.
- „Editioning Views” unterstützen DML-Trigger. Demzufolge sollten die Trigger, die normalerweise auf Tabellen-Level spezifiziert sind, auf View-Level präzisiert werden. Solche Trigger feuern nur, wenn man DML-Operationen auf „Editioning Views“ absetzt. Mit anderen Worten, sie werden nicht ausgeführt, wenn DML-Operationen auf die Tabelle selbst abzielen, auf der wiederum die „Editioning View“ basiert.
- Eine „Editioning View“ unterstützt keine “INSTEAD OF”-Trigger.

Für die Erstellung einer „Editioning View“ ist dem „CREATE VIEW“-Kommando das Schlüsselwort „EDITIONING“ anzuhängen. Das folgende SQL-Kommando zeigt beispielhaft, wie eine Tabelle durch eine „Editioning View“ die Zuordnung der physikalischen Spaltennamen zu logischen Namen vornimmt. Zusätzlich ist ein DML-Trigger mit der View selber assoziiert. Der gleiche Trigger wäre bei einer normalen View nicht möglich, siehe Listing unten.

Während eines anfallenden Upgrades können die „Editioning Views“ gegebenenfalls in den „read-only“- oder „read-write“-Modus gesetzt werden. Für die Vereinfachung eines Upgrades sollten alle auf Tabellen basierenden „Editioning Views“ in den „read-only“-Modus gesetzt sein. Jedoch kann die Applikation die Daten aus diesen Tabellen nur lesen und nicht ändern. Wenn der „read-only“-Modus nicht in Frage kommt, muss überlegt werden, ob es eventuell notwendig ist, sogenannte „Crossedition Trigger“ einzusetzen.

Crossedition Trigger

Ein Crossedition Trigger kommt zum Einsatz, wenn zwei Anforderungen zutreffen:

1. Die Struktur muss während eines Upgrades redefiniert werden.
2. Die „Editioning View“, die über einer zu redefinierenden Tabelle liegt, kann nicht in den „read-only“-Modus gesetzt werden.

Nehmen wir an, dass eine Applikation den vollständigen Zugriff auf eine

```
SQL> DESCRIBE persons
 Name                      Null?    Type
 ------------------------- -------- ----------------
 ID                        NOT NULL NUMBER(10)
 FIRST_NAME                NOT NULL VARCHAR2(100)
 LAST_NAME                 NOT NULL VARCHAR2(100)
 EMAIL                     NOT NULL VARCHAR2(100)

SQL> RENAME persons TO persons_tab;

SQL> CREATE EDITIONING VIEW persons AS
  2  SELECT id, first_name AS firstname, last_name AS lastname, email
  3  FROM persons_tab;

SQL> CREATE TRIGGER persons_bi_trg
  2  BEFORE INSERT ON persons FOR EACH ROW
  3  BEGIN
  4    :new.id := persons_seq.nextval;
  5  END;
  6  /
```

Tabelle benötigt, die eine Spalte beinhaltet, die wiederum E-Mail-Adressen speichert (wie die eine aus dem Beispiel im vorherigen Abschnitt). Die Aufgabe besteht nun darin, diese Spalte in zwei aufzuteilen: Eine Spalte beinhaltet den Empfänger-Namen und die andere den Domain-Namen. Das Hinzufügen der beiden Spalten stellt in den meisten Fällen für diese Tabelle kein Problem dar. Einen „Table Lock" auf eine stark frequentierte Tabelle zu bekommen ist vielleicht problematisch. Demzufolge muss eventuell ein Timeout mit dem Initialisierungsparameter „DDL_LOCK_TIMEOUT" gesetzt sein. Für die Redefinition einer Tabelle ist vielleicht auch das „DBMS_REDEFINITION"-Package hilfreich. Zum Beispiel kann man dafür das folgende SQL-Kommando ausführen:

```
SQL> ALTER TABLE persons_tab
ADD (
  2    email_recipient VAR
       CHAR2(100),
  3    email_domain VAR
       CHAR2(100)
  4  );
```

Das Redefinieren dieser Tabelle ist ein guter Anfang. Aber was passiert mit den darin gespeicherten Daten? Grundsätzlich müssen für das korrekte Ausführen der Redefinition zwei Anforderungen erfüllt sein:

1. Die Daten, die während des Upgrades durch die Applikation geändert werden, müssen in der neuen Struktur gespeichert werden
2. Die Daten, die bereits in der zu redefinierenden Tabelle gespeichert sind, müssen ebenfalls in die neue Struktur konvertiert werden

Für die Erfüllung der ersten Anforderung, kann man in der neuen Edition einen „Forward Crossedition Trigger" erstellen (niemals die alte Edition ändern). Das Ziel eines solchen Triggers besteht darin, eine Zeile von der alten Struktur in die neue zu überführen. In dem hier diskutierten Beispiel erfüllt das folgende SQL-Kommando diese Anforderung:

```
SQL> CREATE TRIGGER persons_fc_trg
  2    BEFORE INSERT OR UPDATE ON persons_tab FOR EACH ROW
  3    FORWARD CROSSEDITION
  4    DISABLE
  5  BEGIN
  6    :new.email_recipient :=
  7      regexp_substr(:new.email,'(.*)@',1,1,NULL,1);
  8    :new.email_domain :=
  9      regexp_substr(:new.email,'@(.*)',1,1,NULL,1);
 10  END;
 11  /
```

Es ist zu beachten, dass selbst, wenn der Trigger in der neuen Edition erstellt wurde, dieser erst feuert, wenn ein DML-Kommando durch eine Session aus einer Vorgänger-Edition, in der der Trigger erstellt wurde, ausgeführt wird. Aus diesem Grund muss durch die „FORWARD CROSSEDITION"-Klausel explizit definiert werden, dass es sich um einen Forward Crossedition Trigger handelt. Zu beachten ist auch, dass dieser Trigger im Status „Disabled" zu erstellen ist. Das ist insofern wichtig, als ein ungültiger Trigger nicht die Verfügbarkeit der Applikation stört. Infolgedessen sollte man ihn nur aktivieren, wenn man sicher weiß, dass er gültig ist. Für die Erfüllung der zweiten Anforderung muss die Transformation auf die bereits gespeicherten Daten angewendet sein. Für dieses Szenario gibt es mehrere Möglichkeiten. Das Einfachste, aber nicht das Beste aus Performance-Sicht ist, den Trigger für jede der bereits in der redefinierten Tabelle gespeicherten Zeile auszulösen. Das „DBMS_SQL"-Package stellt, wie im folgenden PL/SQL-Block zu sehen ist, eine neue Version der „PARSE"-Funktion bereit. Es ist zu beachten, wie ein Dummy-„UPDATE"-Kommando dafür benutzt wird, den durch den „APPLY_CROSSEDITION_TRIGGER"-Parameter spezifizierten Trigger auszulösen, siehe Listing unten.

Es ist essenziell, zur Vermeidung eines inkonsistenten Zustands zu verstehen, dass die erforderlichen Operationen, die diese beiden Anforderungen erfüllen sollen, in der folgenden Reihenfolge ausgeführt werden müssen:

- Aktivieren des Forward Edition Triggers:

```
SQL> ALTER TRIGGER persons_fc_
trg ENABLE;
```

- Warten auf jede Transaktion der redefinierten Tabelle – entweder muss diese bestätigt oder zurückgesetzt sein. Zu diesem Zweck wurde eine neue Funktion, die sogenannte „WAIT_ON_PENDING_DML"-Funktion, innerhalb des „DBMS_UTILITY"-Package hinzugefügt. Wenn dieser Schritt nicht ausgeführt wird, können die Transaktionen, die wäh-

```
SQL> DECLARE
  2    c INTEGER;
  3    r INTEGER;
  4  BEGIN
  5    c := dbms_sql.open_cursor;
  6    dbms_sql.parse(
  7      c => c,
  8      statement => ,UPDATE persons SET email_domain = email_domain',
  9      language_flag => dbms_sql.native,
 10      apply_crossedition_trigger => ,persons_fc_trg'
 11    );
 12    r := dbms_sql.execute(c);
 13    dbms_sql.close_cursor(c);
 14    COMMIT;
 15  END;
 16  /
```

rend des Aktivierens des Forward Edition Triggers geöffnet waren, in einen inkonsistenten Zustand gelangen:

```
SQL> DECLARE
  2    r BOOLEAN;
  3    scn INTEGER;
  4  BEGIN
  5    r := dbms_utility.wait_on_
       pending_dml(
  6            tables => ,CHA.PER
       SONS_TAB‘,
  7            timeout => NULL,
  8            scn => scn
  9          );
 10  END;
 11  /
```

- Start des PL/SQL-Blocks, der den Forward Edition Trigger für jede Zeile auslöst

Sobald diese Operationen abgeschlossen sind, sind vielleicht auch einige neue Constraints zu erstellen. Beispielsweise sind in diesem Fall die neuen Spalten als „NOT NULL“ gesetzt.

```
SQL> ALTER TABLE persons_tab MODIFY (
  2    email_recipient NOT NULL,
  3    email_domain NOT NULL
  4  );
```

Das bisher gezeigte Beispiel legt dar, was getan werden kann, um die Änderungen, die von einer mit der alten Edition arbeitenden Applikation auf die von der neuen Edition genutzten Datenstrukturen zu propagieren. Eine weitere Situation, die berücksichtigt werden muss: Was passiert, wenn aktuell beide, die alte und die neue Edition, gleichzeitig genutzt werden? Dies tritt auf, wenn ein sogenannter „Hot Rollover“ auf die neue Edition erforderlich ist. Für die Erfüllung dieser neuen Anforderung muss ein anderer Typ von Trigger verwendet werden, der sogenannte „Reverse Crossedition Trigger“. Einfach gesagt ist der Zweck eines solchen Triggers, das Gegenteil eines Forward Crossedition Triggers zu erreichen. Mit anderen Worten hat dieser die Funktion, eine Zeile der neuen in die alte Struktur zu überführen. Das folgende SQL-Kommando zeigt ein Beispiel dafür, siehe Listing unten.

In diesem Fall ist der Trigger ebenfalls in der neuen Edition zu erstellen (auch hier gilt, dass man niemals die alte Edition ändern soll). Zu beachten ist außerdem, dass ein Reverse Crossedition Trigger nur ausgelöst wird, wenn eine Session ein DML-Kommando ausführt, in welcher der Trigger erstellt wurde (oder ein Nachfolger davon). Aus diesem Grund ist explizit über die „REVERSE CROSSEDITION“-Klausel zu definieren, dass es sich um einen Reverse Crossedition Trigger handelt.

Die hier gezeigten Beispiele zu Crossedition Triggern sind sehr einfach. Wenn die Komplexität der Redefinition steigt, beispielsweise wenn SQL-Kommandos innerhalb des Triggers ausgeführt werden müssen, sind verschiedene andere Fragen zu berücksichtigen. Da diese kurze Einführung nicht alle diese Details abdecken kann, wird empfohlen, für zusätzliche Informationen die am Ende des Papiers gelisteten Dokumente zu referenzieren.

```
SQL> CREATE TRIGGER persons_rc_trg
  2    BEFORE INSERT OR UPDATE ON persons_tab FOR EACH ROW
  3    REVERSE CROSSEDITION
  4    DISABLE
  5  BEGIN
  6    :new.email := :new.email_recipient || ‚@‘ || :new.email_domain;
  7  END;
  8  /
```

Entfernen von Editionen

Das „DROP EDITION“-Kommando entfernt eine Edition. Für die Ausführung dieses Kommandos ist das „DROP ANY EDITION“-Privileg erforderlich. Zusätzlich müssen folgende Bedingungen erfüllt sein:

- Die Edition darf nicht die letzte Datenbank-Edition sein
- Die Edition ist nicht die Basis-Datenbank-Edition
- Die Edition wird nicht als Session Edition verwendet
- Die Edition ist entweder Root oder Kind der Hierarchie
- Wenn die Edition Kind der Hierarchie ist und sie noch assoziierte Objekte besitzt, dann muss die CASCADE-Option spezifiziert werden
- Wenn die Edition Root der Hierarchie ist, darf die nachfolgende Edition keine Objekte der Root Edition übernehmen. Mit anderen Worten müssen alle übernommenen Objekte durch die nachfolgende Edition redefiniert werden

Das Entfernen einer nicht mehr genutzten Edition ist optional. Statt eine nicht mehr genutzte Edition zu entfernen, kann sie auch über das Widerrufen des „USE ON EDITION“-Privilegs für alle Benutzer außer Dienst gestellt werden.

Einschränkungen

11*g* R2 hat zwei wichtige Implementierungs-Einschränkungen, die sich auf die Hierarchien beziehen, die mit Editionen erstellt wurden:

- Jede Edition kann höchstens eine nachfolgende Edition haben. Demzufolge ist eine Hierarchie, wie auf der linken Seite in Abbildung 2 gezeigt, nicht erlaubt.
- Jede Datenbank besitzt nur eine Root Edition. Folglich kann eine Datenbank keine zwei Hierarchien, wie auf der rechten Seite in Abbildung 2 gezeigt, beinhalten.

Während die erstgenannte Einschränkung nicht ausschlaggebend ist, kann die letztere kritisch sein. In der Tat ist es nicht erlaubt, unabhängige Editionen für jede Applikation, die die gleiche Datenbank für Konsolidierungszwecke nutzt, zu verwenden. Für das Beispiel in Abbildung 2 könnte man eine Hierarchie für die CRM-Applikation nutzen und eine andere für die HR-Applikation.

Ein Feature, das aus Sicht des Autors fehlt, ist die Möglichkeit, die Basis-Edi-

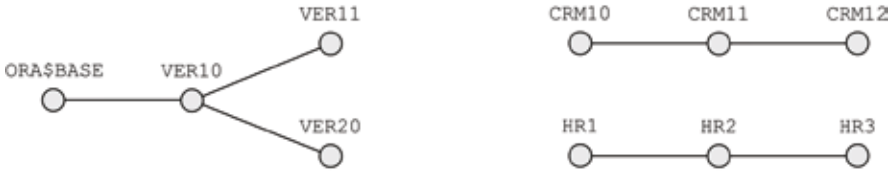

Abbildung 2: Beispiele für nicht zulässige Hierarchien

tion mit einem Datenbank-Service zu verknüpfen (und nicht nur mit der Datenbank, wie in der aktuellen Implementierung). Dies könnte, wenn mehrere Applikationen in einer einzigen Datenbank konsolidiert werden, sehr nützlich sein.

Die übrigen wichtigen Einschränkungen beziehen sich auf Indizes und Constraints. Da beide nur auf einer Tabelle erstellt werden können, gibt es keine Möglichkeit, den Vorteil von Editionen für diese zu nutzen. Im Falle von neuen Indizes kann ihre Auswirkung durch das Setzen auf „unsichtbar" („Invisible") limitiert werden. Mit Constraints hingegen ist es eventuell nicht möglich, eine Definition zu verwenden, die korrekt mit mehr als einer Edition funktioniert. Die einzige generelle Ausnahme bezieht sich auf Check Constraints. Tatsächlich kann für diese ihre Logik auf der Session Edition basieren.

Fazit

Obwohl es immer noch einige Einschränkungen gibt, offeriert Edition-Based Redefinition komplett neue Features, die Online-Upgrades ermöglichen. Nichtsdestotrotz darf man die Komplexität solcher Upgrades nicht unterschätzen.

Referenzen

1. Oracle White Paper, Edition-Based Redefinition in 11*g* R2
2. 11*g* R2-Documentation, Advanced Application Developer's Guide
3. 11*g* R2-Documentation, SQL Language Reference
4. 11*g* R2-Documentation, PL/SQL Packages and Types Reference

Christian Antognini
christian.antognini@trivadis.com
Trivadis AG

KeepTool mit neuer Version 10
Das handliche Werkzeug für Oracle™-Datenbanken
DDL HTM SQL
Zahlreiche neue Funktionen, z.B.
- Darstellung Ihrer Daten als Pivottabelle, ggf. mehrstufig.
- Praktische Hinweistexte bei der Datenerfassung.
- Überwachung und Steuerung der Optimizer-Statistiken.
- Data Pump-Schnittstelle.
- Jumplist für den Windows 7™ Taskbar.
Laden Sie die kostenlose Testversion unter www.keeptool.com herunter.
ORACLE Gold Partner
keeptool