日日是Oracle APEX

Oracle APEXを使った作業をしていて、気の付いたところを忘れないようにメモをとります。

2021年3月5日金曜日

エディションのライフサイクルの確認

エディションベース再定義を使う際のライフサイクルについて、APEXなしで確認してみました。

確認するシナリオは、以下です。

元々は、以下の定義の表TEST_TABLE1が存在していたと想定します。

```
create table test_table1(
   id number primary key,
   value1 varchar2(80),
   value2 varchar2(80)
);
```

新しいエディションでは、上記の表を以下のふたつの表TEST_TABLE1、TEST_TABLE2に分割します。

```
create table test_table1(
   id number primary key,
   value1 varchar2(80)
);

create table test_table2(
   id  number constraint test_table2_pid_fk
       references test_table1 on delete cascade,
   value2  varchar2(80)
);
```

エディションベースの再定義を使って、上記を実施するために必要な作業を確認していきます。

1. 現行を想定したエディションを準備する。

新規にエディションV01を作成し、そのエディションでアプリケーションが作成されていることを想定します。

ユーザー**ADMINで接続**し、エディションV01を作成し、それをデフォルトのエディションに設定します。

```
SQL> create edition v01 as child of ora$base;

Edition V01は作成されました。

SQL> alter database default edition = v01;
```

```
Databaseが変更されました。

SQL>
```

アプリケーションが使用するスキーマの定義は、ユーザー**APEXDEVで接続**して作業します。

アプリケーションから表TEST_TABLE1として見えるように、表はTEST_TABLE1_B、エディショニング・ビューをTEST_TABLE1として作成します。

作業を行うエディションを確認します。

```
SQL> show edition
EDITION
-------
V01
SQL>
```

表TEST_TABLE1_Bを作成します。

```
create table test_table1_b(
   id number primary key,
   value1 varchar2(80),
   value2 varchar2(80)
);
```

```
SQL> create table test_table1_b(
  2       id number primary key,
  3       value1 varchar2(80),
  4       value2 varchar2(80)
  5   );

Table TEST_TABLE1_Bは作成されました。

SQL>
```

エディショニング・ビューTEST_TABLE1を作成します。

```
create editioning view test_table1
as
select * from test_table1_b;
```

```
SQL> create editioning view test_table1
  2   as
  3   select * from test_table1_b;

View TEST_TABLE1は作成されました。

SQL>
```

データを 1 行挿入し、作成した表、エディショニング・ビューが想定どおり作成されているかを確認します。

```
SQL> insert into test_table1(id,value1,value2) values(1,'abc','123');

1行挿入しました。
```

```
SQL> commit;

コミットが完了しました。

SQL> select * from test_table1;

   ID    VALUE1    VALUE2
_____ _________ _________
    1 abc       123

SQL> select * from test_table1_b;

   ID    VALUE1    VALUE2
_____ _________ _________
    1 abc       123

SQL>
```

エディションV01のスキーマ定義が完了しました。アプリケーションはセッション・エディションをV01とすることで、このスキーマにアクセスします。

2. バージョンアップを行う。

ユーザー**ADMINで接続**し新規にエディションV02を作成します。デフォルト・エディションをV02へ変更します。

既存のアプリケーションはセッションの開始時にALTER SESSIONでエディションを指定するか、または、サービスにエディションを設定することでV01を維持するようにしておきます。

```
SQL> create edition v02 as child of v01;

Edition V02は作成されました。

SQL> grant use on edition v01 to apexdev;

Grantが正常に実行されました。

SQL> grant use on edition v02 to apexdev;

Grantが正常に実行されました。

SQL> alter database default edition = v02;

Databaseが変更されました。

SQL>
```

ユーザー**APEXDEVで接続**し、バージョンアップ作業を行います。表TEST_TABLE1_Bを分割しますが、元となる表TEST_TABLE1_Bは変更せずに、双方とも新たに作成します。

```
create table test_table1_b2(
   id number primary key,
   value1 varchar2(80)
);

create table test_table2_b2(
   id  number constraint test_table2_pid_fk
       references test_table1_b2 on delete cascade,
   value2  varchar2(80)
);

alter table test_table2_b2 modify id unique;
```

最初にエディションを確認します。

```
SQL> show edition
EDITION
-------
V02
SQL>
```

続いて表を作成します。

```
SQL> create table test_table1_b2(
  2       id number primary key,
  3       value1 varchar2(80)
  4   );

Table TEST_TABLE1_B2は作成されました。

SQL> create table test_table2_b2(
  2       id  number constraint test_table2_pid_fk
  3            references test_table1_b2 on delete cascade,
  4       value2  varchar2(80)
  5   );

Table TEST_TABLE2_B2は作成されました。

SQL> alter table test_table2_b2 modify id unique;

Table TEST_TABLE2_B2が変更されました。

SQL>
```

作成した表をTEST_TABLE1、TEST_TABLE2としてアプリケーションより使用するために、エディションニング・ビューをそれぞれ作成します。

```
create or replace editioning view test_table1
as
select * from test_table1_b2;

create or replace editioning view test_table2
as
select * from test_table2_b2;
```

```
SQL> create or replace editioning view test_table1
  2  as
```

```
  3  select * from test_table1_b2;

View TEST_TABLE1は作成されました。

SQL> create or replace editioning view test_table2
  2  as
  3  select * from test_table2_b2;

View TEST_TABLE2は作成されました。

SQL>
```

フォワード・クロスエディション・トリガーを設定します。トリガーが正しくコンパイルできたことを確認して、有効にします。

```
create or replace editionable trigger forward_to_v02
before insert or update or delete
on test_table1_b
for each row
forward crossedition
disable
declare
   l_id number;
   row_already_present exception;
   pragma exception_init(row_already_present, -38911);
begin
   if applying_crossedition_trigger then
      /* すでに行が存在する場合は無視する */
      insert /*+ IGNORE_ROW_ON_DUPKEY_INDEX(test_table1_b2(id)) */
      into test_table1_b2(id, value1) values(:new.id, :new.value1);
      if sql%rowcount = 1 then
         insert /*+ IGNORE_ROW_ON_DUPKEY_INDEX(test_table2_b2(id)) */
         into test_table2_b2(id, value2) values(:new.id, :new.value2);
      end if;
   else
      if inserting then
         /* 対応する行がある場合はアップデートを行う */
         begin
            insert /*+ CHANGE_DUPKEY_ERROR_INDEX(test_table1_b2(id)) */
            into test_table1_b2(id, value1) values(:new.id, :new.value1);
            insert into test_table2_b2(id, value2) values(:new.id, :new.value2);
         exception
            when row_already_present then
               update test_table1_b2 set value1 = :new.value1 where id = :new.id;
               update test_table2_b2 set value2 = :new.value2 where id = :new.id;
         end;
      elsif updating then
         update test_table1_b2 set value1 = :new.value1 where id = :old.id;
         update test_table2_b2 set value2 = :new.value2 where id = :old.id;
      elsif deleting then
         delete from test_table2_b2 where id = :old.id;
         delete from test_table1_b2 where id = :old.id;
      end if;
    end if;
end;
/
```

```
SQL> create or replace editionable trigger forward_to_v02
  2  before insert or update or delete
  3  on test_table1_b
```

```
  4  for each row
  5  forward crossedition
  6  disable
  7  declare
  8      l_id number;
  9      row_already_present exception;
 10      pragma exception_init(row_already_present, -38911);
 11  begin
 12      if applying_crossedition_trigger then
 13          insert /*+ IGNORE_ROW_ON_DUPKEY_INDEX(test_table1_b2(id)) */
 14          into test_table1_b2(id, value1) values(:new.id, :new.value1);
 15          if sql%rowcount = 1 then
 16              insert /*+ IGNORE_ROW_ON_DUPKEY_INDEX(test_table2_b2(id)) */
 17              into test_table2_b2(id, value2) values(:new.id, :new.value2);
 18          end if;
 19      else
 20          if inserting then
 21              /* 対応する行がある場合はアップデートを行う */
 22              begin
 23                  insert /*+ CHANGE_DUPKEY_ERROR_INDEX(test_table1_b2(id)) */
 24                  into test_table1_b2(id, value1) values(:new.id, :new.value1);
 25                  insert into test_table2_b2(id, value2) values(:new.id, :new.value2);
 26              exception
 27                  when row_already_present then
 28                      update test_table1_b2 set value1 = :new.value1 where id = :new.id;
 29                      update test_table2_b2 set value2 = :new.value2 where id = :new.id;
 30              end;
 31          elsif updating then
 32              update test_table1_b2 set value1 = :new.value1 where id = :old.id;
 33              update test_table2_b2 set value2 = :new.value2 where id = :old.id;
 34          elsif deleting then
 35              delete from test_table2_b2 where id = :old.id;
 36              delete from test_table1_b2 where id = :old.id;
 37          end if;
 38       end if;
 39  end;
 40  /

Trigger FORWARD_TO_V02がコンパイルされました

SQL> alter trigger forward_to_v02 enable;

Trigger FORWARD_TO_V02が変更されました。

SQL>
```

applying_crossedition_triggerが真の条件で記述されているコードが、この後、DBMS_SQL.PARSEプロシージャを使って既存データを移行するときに呼び出されます。一般には元表のすべてのデータを対象とするため、すでに移行済みの行があると重複としてエラーになります。そのためIGNORE_ROW_ON_DUPKEY_INDEXヒントを加えて、重複エラーを無視しています。

通常のINSERT時では、重複があった場合はORA-00001ではなくORA-38911が上がるようにCHANGE_DUPKEY_ERROR_INDEXヒントを加えています。ORA-00001は元表(今回はTEST_TABLE1_B)で発生するもので、トリガー内での操作でORA-00001が上がるのは適切ではないためです。

今回は検証のためひとつのトリガー内ですべての処理を行っています。本番環境を想定した場合は、パフォーマンス面を考慮して、条件ごとのトリガーを作成した方が良いでしょう。

作成したトリガーの処理を確認してみます。セッション・エディションをV01へ変更し、いくつかDMLを実行します。

```
SQL> alter session set edition = v01;

Sessionが変更されました。

SQL> select * from test_table1;

   ID    VALUE1    VALUE2
_____ _________ _________
    1 abc       123

SQL> insert into test_table1(id,value1,value2) values(2,'def','222');

1行挿入しました。

SQL> commit;

コミットが完了しました。

SQL> select * from test_table1;

   ID    VALUE1    VALUE2
_____ _________ _________
    1 abc       123
    2 def       222

SQL> select * from test_table1_b2;

   ID    VALUE1
_____ _________
    2 def

SQL> select * from test_table2_b2;

   ID    VALUE2
_____ _________
    2 222

SQL> update test_table1 set value2 = '456' where id = 2;

1行更新しました。

SQL> commit;

コミットが完了しました。

SQL> select * from test_table1;
```

```
   ID    VALUE1    VALUE2
_____ _________ _________
    1 abc       123
    2 def       456

SQL> select * from test_table2_b2;

   ID    VALUE2
_____ _________
    2 456

SQL> delete from test_table1 where id = 2;

1行削除されました。

SQL> commit;

コミットが完了しました。

SQL> select * from test_table1;

   ID    VALUE1    VALUE2
_____ _________ _________
    1 abc       123

SQL> select * from test_table1_b2;

行が選択されていません
SQL> select * from test_table2_b2;

行が選択されていません
SQL>
```

エディショニング・ビューTEST_TABLE1の操作によって、表TEST_TABLE1_B2、TEST_TABLE2_B2の操作も同時に行われていることが確認できます。

定義済みのフォワード・クロスエディション・トリガーを使用して、今までに表TEST_TABLE1_Bに保存されていたデータを、表TEST_TABLE1_B2、TEST_TABLE2_B2へ移行します。

```
declare
  c     number := dbms_sql.open_cursor();
  retval number;
begin
    dbms_sql.parse(
        c => c,
        statement => 'update test_table1_b set id = id',
        language_flag => dbms_sql.native,
        edition => null,
        apply_crossedition_trigger => 'FORWARD_TO_V02',
        fire_apply_trigger => TRUE
    );
    retval := dbms_sql.execute(c);
    dbms_sql.close_cursor(c);
    commit;
```

```
end;
/
```

```
SQL> alter session set edition = v02;

Sessionが変更されました。

SQL> declare
  2    c     number := dbms_sql.open_cursor();
  3    retval number;
  4  begin
  5      dbms_sql.parse(
  6           c => c,
  7           statement => 'update test_table1_b set id = id',
  8           language_flag => dbms_sql.native,
  9           edition => null,
 10           apply_crossedition_trigger => 'FORWARD_TO_V02',
 11           fire_apply_trigger => TRUE
 12       );
 13       retval := dbms_sql.execute(c);
 14       dbms_sql.close_cursor(c);
 15       commit;
 16  end;
 17  /

PL/SQLプロシージャが正常に完了しました。

SQL>
```

移行された内容を確認します。

```
SQL> select * from test_table1_b2 where id = 1;

   ID    VALUE1
_____ _________
    1 abc

SQL> select * from test_table2_b2 where id = 1;

   ID    VALUE2
_____ _________
    1 123

SQL>
```

以上でスキーマ定義およびデータの移行が完了したので、アプリケーションはすべてエディションV02で動作させることができます。

3. エディションのリタイア

エディションV01は不要なのでリアイアをさせます。

最初にユーザー**APEXDEVで接続**し、不要になったフォワード・クロスエディション・トリガーを削除します。

```
SQL> show edition
EDITION
-------
V02
SQL> drop trigger forward_to_v02;

Trigger FORWARD_TO_V02が削除されました。

SQL>
```

ユーザー**ADMINで接続**します。USE権限を持っているユーザーを確認します。

```
SELECT GRANTEE, PRIVILEGE
FROM DBA_TAB_PRIVS
WHERE TABLE_NAME = 'V01' AND TYPE = 'EDITION'
/
```

```
SQL> SELECT GRANTEE, PRIVILEGE
  2  FROM DBA_TAB_PRIVS
  3  WHERE TABLE_NAME = 'V01' AND TYPE = 'EDITION'
  4  /

   GRANTEE    PRIVILEGE
__________ _____________
ADMIN      USE
APEXDEV    USE

SQL>
```

リストされたユーザーから権限を除きます。最初にセッション・エディションを確認します。

```
SQL> revoke use on edition v01 from apexdev;

Revokeが正常に実行されました。

SQL>
```

エディションをドロップします。

```
SQL> drop edition v01;

次のコマンドの開始中にエラーが発生しました : 行 1 -
drop edition v01
エラー・レポート -
ORA-38810: Implementation restriction: cannot drop edition that has a
parent and a child

SQL>
```

エディションV01は最初のエディションではなく、また、最後のエディションでもないため、ドロップの試みてもエラーになります。

現在のルート・エディションはORA$BASEなので、それを最初に削除します。

```
SQL> drop edition ora$base;

次のコマンドの開始中にエラーが発生しました : 行 1 -
drop edition ora$base
エラー・レポート -
ORA-38811: need CASCADE option to drop edition that has actual objects

SQL>
```

オブジェクトが存在するためにエラーになりました。CASCADEオプションを追加して削除します。

```
SQL> drop edition ora$base cascade;

Edition ORA$BASEが削除されました。

SQL>
```

続けてエディションV01を削除します。

```
SQL> drop edition v01 cascade;

Edition V01が削除されました。

SQL>
```

以上でエディションV02のみでデータベースが稼働している状態になりました。

以上の作業を繰り返すことでアプリケーションとスキーマを更新していくことができます。

スキーマの変更を伴うアプリケーションの更新は、まとめて一度に行うことが多いように思います。そのため作業量が多くなり、実施を敬遠しがちになるのではないでしょうか。エディションベースの再定義の使用を前提とした場合は、できるだけ小さな単位で細かく更新をする方が向いているため、より、アジャイルな対応が可能になるように感じました。

完

Yuji N. 時刻: 16:29

共有

自己紹介

Yuji N.

日本オラクル株式会社に勤務していて、Oracle APEXのGroundbreaker Advocateを拝命しました。
こちらの記事につきましては、免責事項の参照をお願いいたします。

詳細プロフィールを表示