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。

2022年4月4日月曜日

DBMS_CLOUD.COPY_DATAとAPEX_DATA_LOADINGの比較

最近、DBMS_CLOUD.EXPORT_DATAとパッケージAPEX_DATA_EXPORTを使ったCSVファイルへのエクスポートについて比較したので、ついでにDBMS_CLOUD.COPY_DATAとパッケージAPEX_DATA_LOADINGも比較してみました。また、外部表を使用したインポートも行ってみました。

速度についてはDBMS_CLOUD.COPY_DATA（および外部表を使ったインポート）が高速です。大量のデータをインポートする場合は、こちらの手順を使うことになると思います。

テストに使用するデータを準備します。こちらの記事と同じ環境でテストを行います。

表BRICKSよりオブジェクト・ストレージに、10,000,000行のデータをエクスポートします。**オブジェクト名**は**bricks_10m_**で始まるようにします。同様に100,000行のデータをエクスポートします。**オブジェクト名**は**bricks_100k_**で始まるようにします。

オブジェクト・ストレージへのエクスポートには、以下のスクリプトを使用します。

```
declare
    C_REGION constant    varchar2(20) := 'us-ashburn-1';
    C_NAMESPACE constant varchar2(20) := 'ネームスペースに置き換える';
    C_BUCKET constant    varchar2(20) := 'download';
$IF false $THEN -- for DBMS_CLOUD.COPY_DATA
    C_FILENAME constant  varchar2(20) := 'bricks_10m';
    C_SQL constant varchar2(4000) := q'~select * from bricks where rownum < 10000001~';
$ELSE -- for APEX_DATA_LOADING
    C_FILENAME constant  varchar2(20) := 'bricks_100k';
    C_SQL constant varchar2(4000) := q'~select * from bricks where rownum < 100001~';
$END
    l_path     varchar2(400);
    l_operation_id number;
    l_status       varchar2(9);
    procedure set_checkpoint(ids varchar2)
    as
    begin
        dbms_output.put_line(systimestamp || ': ' || ids);
        dbms_session.set_identifier('csv_export-' || ids);
    end;
begin
    SET_CHECKPOINT('dbms_cloud.export_data');
    l_path := 'https://objectstorage.' || C_REGION || '.oraclecloud.com/n/' || C_NAMESPACE
```

```
        || '/b/' || C_BUCKET || '/o/' || C_FILENAME;
    dbms_cloud.export_data
    (
        credential_name => 'DEF_CRED'
        , file_uri_list => l_path
        , format => json_object(
            'type' value 'csv'
            , 'maxfilesize' value '214783648'
            , 'compression' value 'gzip'
        )
        , query => C_SQL
        , operation_id => l_operation_id
    );
    SET_CHECKPOINT('end');
    for i in 1..10
    loop
        select status into l_status from user_load_operations where id = l_operation_id;
        SET_CHECKPOINT(l_status);
        if l_status = 'COMPLETED' then
            exit;
        end if;
        dbms_session.sleep(1);
    end loop;
end;
/
```

export_obs_bricks.sql hosted with ❤️ by GitHub view raw

$IFのフラグをtrueとfalseに変更し、それぞれ１回ずつスクリプトを実行すると以下のように、オブジェクトが２つ、指定したバケット以下に作成されます。

DBMS_CLOUD.COPY_DATAを使ったデータ・ロード

DBMS_CLOUD.COPY_DATAを実行し、bricks_10m_xxxx.csv.gzを表BRICKS_IMPにロードします。

データをロードする表BRICKS_IMPを作成します。

create table bricks_imp as select * from bricks where 1<>1;

```
SQL> create table bricks_imp as select * from bricks where 1<>1;

Table BRICKS_IMPは作成されました。
```

```
SQL> desc bricks_imp

            名前     Nullかどうか             タイプ
_______________ ____________ _______________
BRICK_ID        NOT NULL     NUMBER
COLOUR          NOT NULL     VARCHAR2(6)
SHAPE           NOT NULL     VARCHAR2(8)
WEIGHT          NOT NULL     NUMBER
INSERT_DATE     NOT NULL     DATE
JUNK            NOT NULL     VARCHAR2(50)
SQL>
```

DBMS_CLOUD.COPY_DATAを使って、データをロードするスクリプトは以下になります。

```
declare
    C_REGION constant    varchar2(20) := 'us-ashburn-1';
    C_NAMESPACE constant varchar2(20) := 'ネームスペースに置き換える';
    C_BUCKET constant    varchar2(20) := 'download';
    C_FILENAME constant  varchar2(80) := 'bricks_10m_';
    l_path     varchar2(400);
    l_operation_id number;
    l_status       varchar2(9);
    procedure set_checkpoint(ids varchar2)
    as
    begin
        dbms_output.put_line(systimestamp || ': ' || ids);
        dbms_session.set_identifier('csv_import-' || ids);
    end;
begin
    SET_CHECKPOINT('dbms_cloud.copy_data');
    l_path := 'https://objectstorage.' || C_REGION || '.oraclecloud.com/n/' || C_NAMESPACE
        || '/b/' || C_BUCKET || '/o/' || C_FILENAME || '*';
    dbms_cloud.copy_data
    (
        table_name => 'BRICKS_IMP'
        , credential_name => 'DEF_CRED'
        , file_uri_list => l_path
        , format => json_object(
            'type' value 'csv'
            , 'compression' value 'gzip'
            , 'characterset' value 'AL32UTF8'
            , 'dateformat' value 'DD-MM-RR'
        )
        , operation_id => l_operation_id
    );
    SET_CHECKPOINT('end');
    for i in 1..10
    loop
        select status into l_status from user_load_operations where id = l_operation_id;
        SET_CHECKPOINT(l_status);
```

```
        if l_status in ('COMPLETED','FAILED') then
            exit;
        end if;
        dbms_session.sleep(1);
    end loop;
end;
/
```

copy_data hosted with ❤️ by GitHub view raw

10,000,000行のロードにかかっている時間は、大体30秒といったところです。

```
SQL> set serveroutput on
SQL> set time on timing on
13:18:47 SQL> @copy_data
22-04-04 04:18:50.369347000 +00:00: dbms_cloud.copy_data
22-04-04 04:19:22.944414000 +00:00: end
22-04-04 04:19:22.953348000 +00:00: COMPLETED


PL/SQLプロシージャが正常に完了しました。

経過時間: 00:00:32.951
経過時間: 00:00:33.122
13:19:23 SQL> select count(*) from bricks_imp;

   COUNT(*)
___________
   10000000

経過時間: 00:00:00.439
13:19:38 SQL> select * from bricks
  2  minus
  3* select * from bricks_imp;

行が選択されていません
経過時間: 00:00:36.553
13:20:27 SQL>
```

表BRICKS_IMPからすべての行を削除して、再度データのロードを実行しました。処理時間にそれほどの違いは出ていません。

```
13:20:27 SQL> delete from bricks_imp;

10,000,000行削除されました。

経過時間: 00:00:26.661
13:23:37 SQL> commit;

コミットが完了しました。

経過時間: 00:00:00.354
13:23:41 SQL> @copy_data
22-04-04 04:23:45.554795000 +00:00: dbms_cloud.copy_data
22-04-04 04:24:17.105518000 +00:00: end
22-04-04 04:24:17.106193000 +00:00: COMPLETED
```

```
PL/SQLプロシージャが正常に完了しました。

経過時間: 00:00:31.905
経過時間: 00:00:32.080
13:24:17 SQL>
```

セグメントのサイズを確認してみました。

全件削除したのち再インポートしたときのセグメントのサイズは、表BRICKS_IMPをトランケートした後にインポートしたサイズの２倍になっているので、内部的にはダイレクト・パス・インサートになっているようです。

```
13:24:17 SQL> select bytes from user_segments where segment_name = 'BRICKS_IMP';

      BYTES
___________
   51380224

経過時間: 00:00:00.447
13:28:11 SQL> truncate table bricks_imp;

Table BRICKS_IMPが切り捨てられました。

経過時間: 00:00:02.720
13:28:25 SQL> select bytes from user_segments where segment_name = 'BRICKS_IMP';

   BYTES
________
   65536

経過時間: 00:00:00.342
13:28:29 SQL> @copy_data
22-04-04 04:28:34.204638000 +00:00: dbms_cloud.copy_data
22-04-04 04:29:06.123755000 +00:00: end
22-04-04 04:29:06.124399000 +00:00: COMPLETED


PL/SQLプロシージャが正常に完了しました。

経過時間: 00:00:32.442
経過時間: 00:00:32.615
13:29:06 SQL> select bytes from user_segments where segment_name = 'BRICKS_IMP';

      BYTES
___________
   26214400

経過時間: 00:00:00.344
13:29:12 SQL>
```

外部表を使ったデータ・ロード

次のスクリプトを実行して、外部表BRICKS_EXTを作成します。

```
declare
    C_REGION constant    varchar2(20) := 'us-ashburn-1';
    C_NAMESPACE constant varchar2(20) := 'ネームスペースに置き換える';
    C_BUCKET constant    varchar2(20) := 'download';
    C_FILENAME constant  varchar2(80) := 'bricks_10m_';
    l_path     varchar2(400);
begin
    l_path := 'https://objectstorage.' || C_REGION || '.oraclecloud.com/n/' || C_NAMESPACE
        || '/b/' || C_BUCKET || '/o/' || C_FILENAME || '*';
    dbms_cloud.create_external_table(
        credential_name => 'DEF_CRED'
        , table_name    => 'bricks_ext'
        , file_uri_list => l_path
        , format        => json_object(
            'type'   value 'csv'
            , 'dateformat' value 'DD-MM-RR'
            , 'compression' value 'auto'
          )
        , column_list =>
            'brick_id number,
            colour varchar2(6),
            shape varchar2(8),
            weight number,
            insert_date date,
            junk varchar2(50)'
    );
end;
/
```

external_table.sql hosted with ❤️ by GitHub view raw

CREATE TABLE AS SELECTを実行して、表BRICKS_EXTより表BRICKS_IMPを作成します。

create table bricks_imp as select * from bricks_ext;

```
13:44:20 SQL> create table bricks_imp as select * from bricks_ext;

Table BRICKS_IMPは作成されました。

経過時間: 00:00:33.023
13:45:08 SQL>
```

処理時間は33秒で、DBMS_CLOUD.COPY_DATAとそれほど変わりません。

データを表BRICKS_IMPから削除します。その後、INSERT SELECTにて表BRICKS_EXTより表BRICKS_IMPにデータをコピーします。

insert into bricks_imp select * from bricks_ext;

```
13:59:54 SQL> delete from bricks_imp;

10,000,000行削除されました。

経過時間: 00:00:26.859
14:00:31 SQL> commit;

コミットが完了しました。

経過時間: 00:00:00.365
14:00:36 SQL> select bytes from user_segments where segment_name = 'BRICKS_IMP';

     BYTES
----------
  26214400

経過時間: 00:00:00.352
14:00:38 SQL> insert into bricks_imp select * from bricks_ext;

10,000,000行挿入しました。

経過時間: 00:00:30.709
14:01:35 SQL> commit;

コミットが完了しました。

経過時間: 00:00:00.525
14:02:30 SQL> select bytes from user_segments where segment_name = 'BRICKS_IMP';

     BYTES
----------
  59768832

経過時間: 00:00:00.350
14:02:34 SQL>
```

INSERT SELECTでも処理時間はそれほど変わりません。インサート後のセグメントのサイズが倍増していることより、こちらもダイレクト・パス・インサートになっているようです。

データのロードを繰り返す場合は、DELETE文よりTRUNCATE文を使う方が、パフォーマンス面での問題が発生しにくいと思われます。

APEX_DATA_LOADINGを使ったデータ・ロード

パッケージAPEX_DATA_LOADINGに含まれるプロシージャLOAD_DATAを使うには、APEXのアプリケーションに作成された**データ・ロード定義**が必要です。

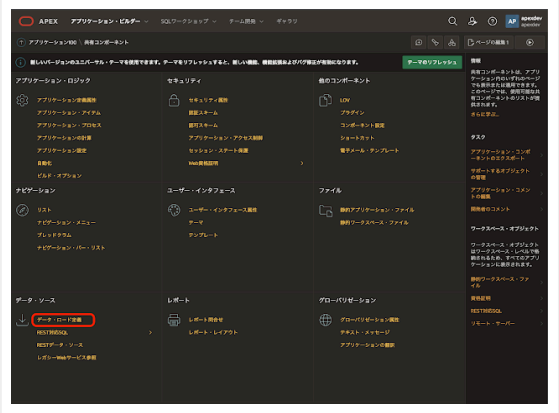
出力されているCSVファイルの、先頭から数行をコピーしておきます。データをCSV形式でエクスポートする際にNLS_LANGを指定していなかったので、日付のフォーマットが**DD-MON-RR**(英語)になっています。

```
1,blue,cylinder,2,01-JAN-11,xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
2,blue,pyramid,8,01-JAN-11,xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
3,blue,prism,3,01-JAN-11,xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
4,blue,cube,3,01-JAN-11,xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
5,blue,cylinder,4,01-JAN-11,xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
6,blue,pyramid,2,01-JAN-11,xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

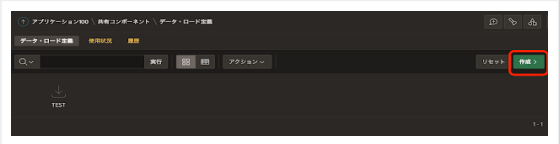
アプリケーション・ビルダーに移ります。

アプリケーションの**共有コンポーネント**に含まれる、**データ・ロード定義**を開きます。



作成済みのデータ・ロード定義が一覧されます。

作成をクリックします。



データ・ロードの作成として、**最初から**を選択します。

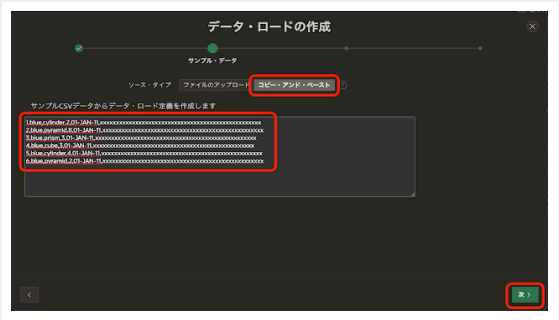
次へ進みます。

名前を**BRICKS**とします。**ターゲット・タイプ**は**表**、**表名**として**BRICKS_IMP**を選択します。

次へ進みます。

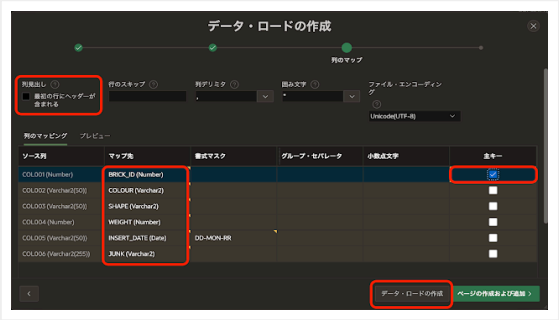
ソース・タイプとして**コピー・アンド・ペースト**を選択し、**サンプル**として、一部のデータを貼り付けます。

次へ進みます。

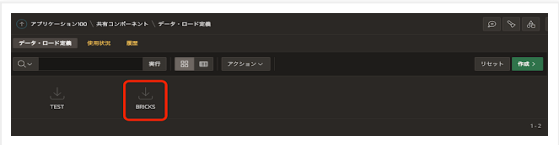


DBMS_CLOUD.COPY_DATAの出力にヘッダーが含まれていないため、**列見出し**の**最初の行にヘッダーが含まれる**の**チェックは外します。マップ先**はCSVのデータ位置に合わせて、**表BRICKS_IMPの列を指定**します。**INSERT_DATE**の**書式マスク**として**DD-MON-RR**を指定します。列**BRICK_ID**の**主キー**に**チェック**を入れます。

ページの作成は不要です。**データ・ロードの作成**をクリックします。



以上でデータ・ロード定義が作成されます。作成されたデータ・ロード定義の細部を調整するために、データ・ロード定義**BRICKS**を開きます。

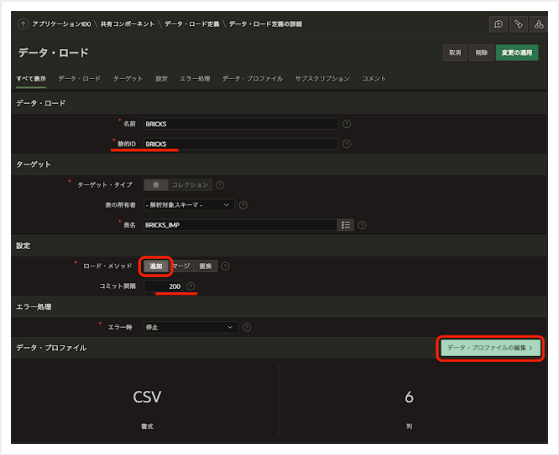


APEXのデータ・ロードの場合、**ロード・メソッド**を**追加**、**マージ**、**置換**から選ぶことができます。DBMS_CLOUD.COPY_DATAとの比較という意味では、追加が近い動作になるため、**追加**を選びま

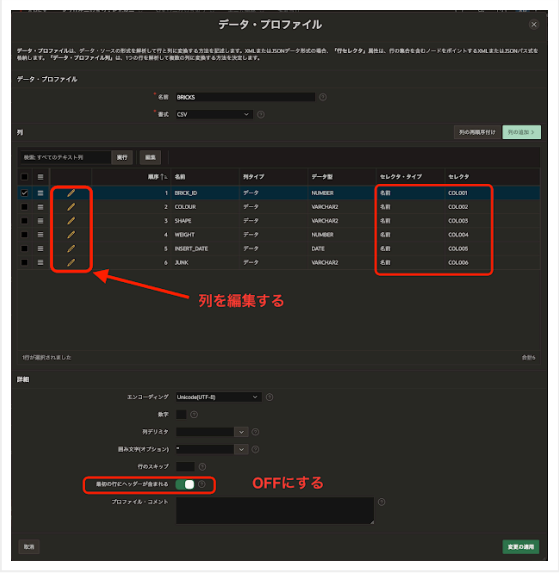
す。**コミット間隔**の数値を大きくすると速度は上がりますが、今回はデフォルトの**200**のままにします。

静的IDは、コードからデータ・ロードを呼び出す際に使用します。

データ・プロファイルの編集をクリックして、編集画面を開きます。

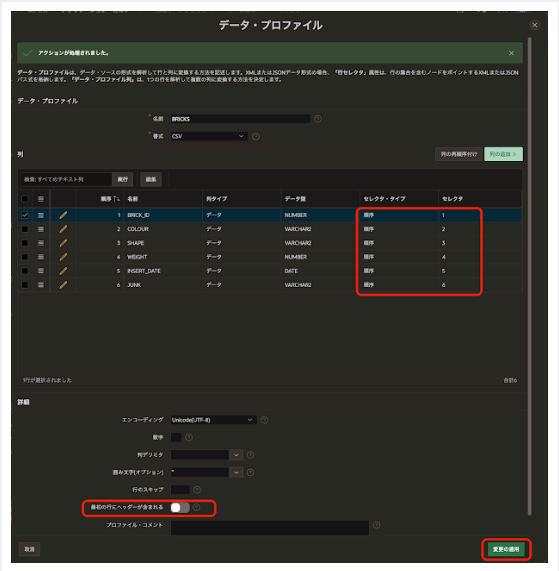


残念なことに、**詳細**の**最初の行にヘッダーが含まれる**が**ON**になっています。また、**列**の**セレクタ・タイプ**と**セレクタ**の設定も、ウィザードで指定した通りになっていません。

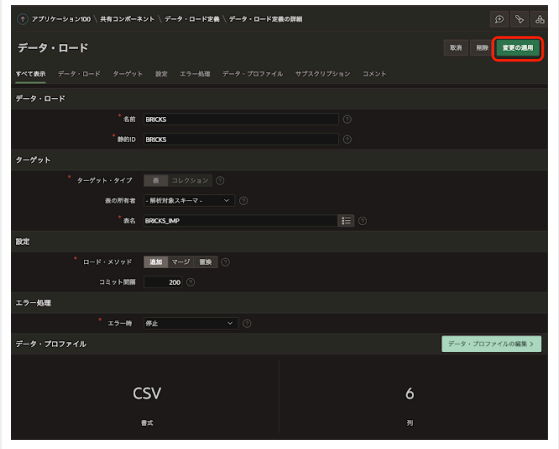


最初の行にヘッダーが含まれるを**OFF**にします。また、列をそれぞれ開き、**セレクタ・タイプ**を**順序**、**セレクタ**に列の位置を**数値**で設定します。

以上の変更を行い、**変更の適用**をクリックします。



データ・ロード定義の画面も閉じます。



以上で、データ・ロード定義の作成は完了です。

APEX_DATA_LOADINGを使ったインポートは、以下のスクリプトで実行します。

```
declare
    C_REGION constant    varchar2(20) := 'us-ashburn-1';
```

```
    C_NAMESPACE constant varchar2(20) := 'ネームスペースに置き換える';
    C_BUCKET constant    varchar2(20) := 'download';
    C_FILENAME constant  varchar2(80) := 'bricks_100k_1_20220404T034543Z.csv.gz';
    l_path     varchar2(400);
    l_blob blob;
    l_load_result apex_data_loading.t_data_load_result;
    procedure set_checkpoint(ids varchar2)
    as
    begin
        dbms_output.put_line(systimestamp || ': ' || ids);
        dbms_session.set_identifier('csv_import-' || ids);
    end;
begin
    l_path := 'https://objectstorage.' || C_REGION || '.oraclecloud.com/n/' || C_NAMESPACE
        || '/b/' || C_BUCKET || '/o/' || C_FILENAME;
    SET_CHECKPOINT('dbms_cloud.get_object');
    l_blob := dbms_cloud.get_object(
        credential_name => 'DEF_CRED'
        , object_uri => l_path
        , compression => 'AUTO'
    );
    dbms_output.put_line('Length: ' || dbms_lob.getlength(l_blob));
    SET_CHECKPOINT('apex_session.create_session');
    apex_session.create_session(
        p_app_id => 100
        , p_page_id => 1
        , p_username => 'APEXDEV'
    );
    SET_CHECKPOINT('apex_data_loading.load_data');
    l_load_result := apex_data_loading.load_data(
        p_static_id => 'BRICKS'
        , p_data_to_load => l_blob
    );
    SET_CHECKPOINT('end');
end;
/
```

apex_import.sql hosted with ❤️ by GitHub view raw

列INSERT_DATEの月の表記は英語なので、スクリプトの実行時にNLS_LANGUAGEをAmericanに変更します。

```
16:42:05 SQL> alter session set NLS_LANGUAGE = 'American';

Sessionが変更されました。

経過時間: 00:00:00.526
16:42:20 SQL> @import
22-04-04 07:42:22.534203000 +00:00: dbms_cloud.get_object
Length: 8094330
```

```
22-04-04 07:42:22.807388000 +00:00: apex_session.create_session
22-04-04 07:42:22.818248000 +00:00: apex_data_loading.load_data
22-04-04 07:50:31.205545000 +00:00: end


PL/SQLプロシージャが正常に完了しました。

経過時間: 00:08:09.042
経過時間: 00:08:09.219
16:50:31 SQL>
```

APEX_DATA_LOADING.LOAD_DATAでは100,000行で8分程度時間がかかっています。コミットに時間がかかっているわけではないので、コミット間隔を増やしてもそれほど効果はなさそうです。

10万行のロードで8分かかる、ということは画面からの実行であればタイムアウトが発生するでしょう。タイムアウトに関しては、データのロード処理をバックグラウンドで実行する（こちらの記事の最後の方で紹介）ことにより対応できます。とはいえ、あまり大量の行をAPEX_DATA_LOADINGを使ってインポートするのは現実的ではなく、Autonomous DatabaseであればDBMS_CLOUDパッケージや外部表、それ以外では外部表やSQL*Loaderなどの利用が有効と言えます。

完

Yuji N. 時刻: 17:37

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