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。

2020年6月23日火曜日

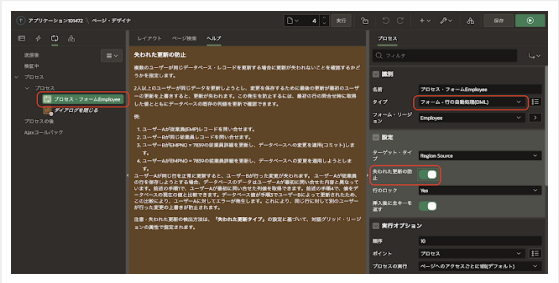
Oracle APEXにおける楽観的並行性制御の実装について

Oracle APEXでは、データベースに保存されているデータを更新するための標準コンポーネントとして、**対話グリッド**と**フォーム**が提供されています。これらの標準コンポーネントに組み込まれている楽観的並行性制御(Optimistic Concurrency Control)について説明します。

難しい話に聞こえるかもしれませんが、この機能は**デフォルトで有効**になっています。そのため、Oracle APEXのアプリケーションでは、楽観的並行性制御を意識して実装する必要はありません。

今回はフォームを例にとって説明します。対話グリッドにも同様の設定があります。正確には対話グリッドに実装されていた機能が、その後に開発されたフォーム・リージョンにも含まれました。

フォームまたは**対話グリッド**の操作を実行するプロセス、**行の自動処理(DML)**の**設定**にある**失われた更新の防止**を**ON**にすることで楽観的並行性制御が有効になります。先ほども説明しましたが、これはデフォルトでONになっています。

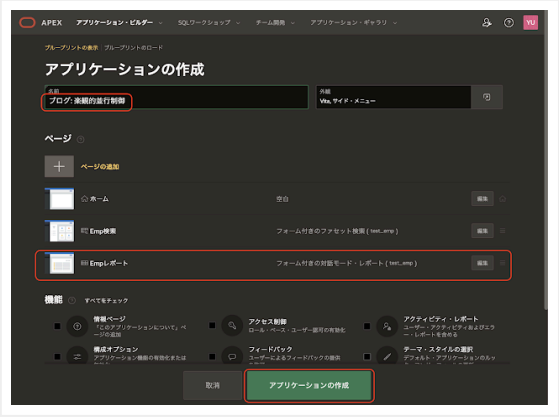


アプリケーションを作って動作を確認してみます。**SQLワークショップ**の**SQLコマンド**から以下のDDLを実行して、表**TEST_EMP**を作成します。

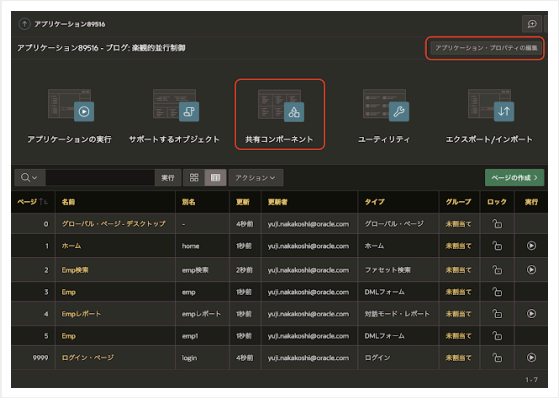
```
create table test_emp (
   id       number generated by default on null as identity
            constraint test_emp_id_pk primary key,
   ename    varchar2(80),
   job      varchar2(40),
   sal      number
);
```

表を作成した後、**オブジェクト・ブラウザ**から作成した表**TEST_EMP**を選択し、**アプリケーションの作成**を実行します。

アプリケーションの作成画面では、任意の名前を設定し、それ以外はウィザードに任せて、**アプリケーションの作成**を実行します。今回の確認で使用するのは、**Empレポート**(フォーム付き対話モード・レポート）のみです。



アプリケーションが作成されますが、実行する前に少々調整をします。最初に、**アプリケーション・プロパティの編集**を開いて、**アプリケーションの別名**を変更します。次に、**共有コンポーネント**の**認証スキーム**を開いて、**認証スキーム**を**公開資格証明**にします。

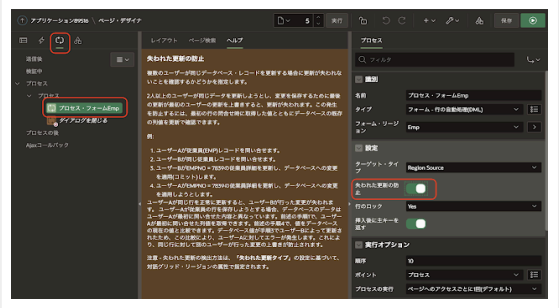


アプリケーションの別名は英数字と一部の記号に限定します。

認証スキームは検証作業を楽にするために、ユーザー名のみで認証できる**公開資格証明**をカレント・スキームにします。

対話モード・レポートから開かれるDMLフォームをページ・デザイナで開き、失われた更新の防止の設定を確認します。

左ペインにてプロセス・ビューを表示させ、フォームのプロセスを選択します。**失われた更新の防止**は**ON**になっています。

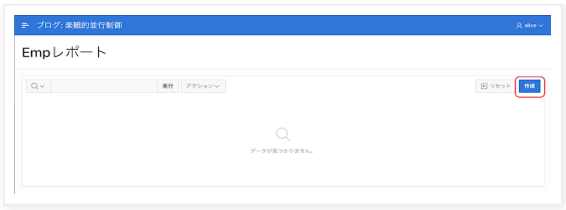


準備は以上で完了です。これからアプリケーションを実行し、動作の確認をしていきます。

最初にアプリケーションを実行し、ユーザー**alice**としてログインします。

Empレポートを開きます。

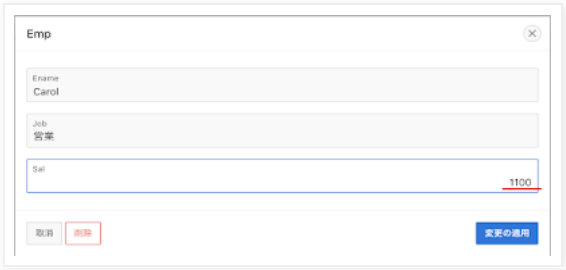
作成をクリックして、確認に使用するデータを登録します。



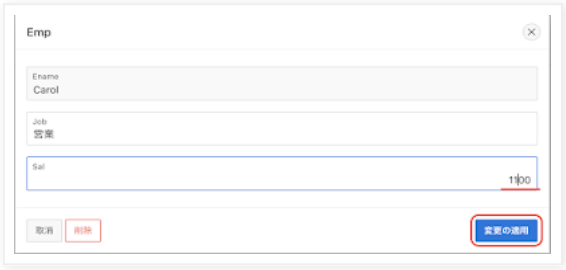
Enameを**Carol**、**Job**を**営業**、**Sal**(Salary)を**1000**として、一行登録します。

AliceとBobがそれぞれ、Carolの**Sal**に**+100**することを依頼されたとします。それぞれ+100ですので、両人が処理を実行した結果は**1200**になることが想定されます。

AliceのアカウントでCarolのデータの編集画面を開きます。**Sal**を**1100**に変更しますが、まだ、**変更の適用**は行いません。



別のブラウザで同じアプリケーションにBobのアカウントでアクセスします。こちらは、1000と表示されている**Sal**を**1100**に変更し、**変更の適用**を実行します。



変更は適用されて、Salは1100として保存されます。

Aliceの画面に戻って、**変更の適用**をクリックします。データベースから行を読み出した後に、保存されている行が変更されたため、エラーが発生して変更の適用が出来ません。エラー・メッセージ

は「**ユーザーが更新処理を開始してから、データベース内の現行バージョンのデータが変更されています。**」です。

一旦このフォームを**取消**をクリックして閉じ、再度、更新処理をやり直す必要があります。次にフォームを開いた時は、Salは1100になるため、+100して**1200**に更新することになり、想定した結果になります。

失われた更新の防止をOFFにして同様の操作を行うと、エラーにならずにSalが保存されることが確認できますので、試してみてください。これは大抵の場合で問題の起こる動作ですので、この設定をOFFにすることはないでしょう。Webアプリケーションを一から作成する場合は、このようなコーディングを自分で行う必要があります。

失われた更新の防止の仕組みですが、２種類の方法から選ぶことができます。フォーム・リージョンの**属性**に、その切り替えがあります。**失われた更新タイプ**がその設定です。

それぞれについて、ヘルプでは以下のように説明されています。

行の値

> データを最初に問い合せる場合、チェックサム値が各行に計算されます。 チェックサムは、すべての更新可能な列を文字列に連結し、一意の値を生成して計算されます。 更新されたレコードをコミットすると、このチェックサムが現在のデータベース・レコードのチェックサム値と比較されます。 同じでない場合、エラーが発生します。

行バージョン列

> データベース表にデータベース・トリガー(可能な場合)によってレコードが更新されるたびに増える列が含まれている場合、チェックサムを計算するかわりにこの列を使用できます。 対話グリッドで複数の表のデータを更新する場合、このオプションを使用しないでください。
>
> 注意 - 行バージョン列をリージョンのSQLソースに含める必要があります。

失われた更新タイプが**行の値**のときは、 データベースからそれぞれの列の情報を取り出してフォームに表示する際に、それらのすべての列を連結して生成したチェックサムも同時にフォームに含めます。このチェックサムは画面には表示されませんが、フォームがサブミットされる際にページ・アイテムと一緒にサーバーへ送信されます。サブミットされたフォームを処理するプロセスは、デ

ータを更新する前に変更の対象となっている行をデータベースから読み出し、チェックサムを計算します。受信したチェックサムと一致している場合（つまり行が変更されていない）のみ、受け取ったページ・アイテムの情報で既存の行を更新します。

行バージョン列はちょっと馴染みがないかもしれません。**クイックSQL**を使って表のDDLを生成するときに指定可能なオプションに**行バージョン番号**というものがあります。行バージョン列というのはそれを指します。

SQLワークショップの**ユーティリティ**から**クイックSQL**を実行します。左の画面には以下の定義を指定します。

```
test_emp
  ename vc80
  job   vc40
  sal   number
```

設定を開いて、**追加列**の**行バージョン番号**にチェックを入れ、**変更の保存**を行います。

結果として以下のようなDDLが生成されます。

```
-- create tables
create table test_emp (
    id                          number generated by default on null as identity
                                constraint test_emp_id_pk primary key,
    row_version                     integer not null,
    ename                       varchar2(80),
    job                         varchar2(40),
    sal                         number
)
;

-- triggers
create or replace trigger test_emp_biu
    before insert or update
    on test_emp
    for each row
begin
    if inserting then
        :new.row_version := 1;
    elsif updating then
        :new.row_version := nvl(:old.row_version,0) + 1;
    end if;
end test_emp_biu;
/
```

追加されたrow_version列はトリガーによってアップデートが実行されるたびに、１づつ数値がインクリメントされます。そのためチェックサムの代わりにrow_versionを読み出し、データの更新

時にはrow_versionを比較することで行の変更を検知することができます。

Oracle APEXのアプリケーションではチェックサムを生成するコードを記述する必要はないため、デフォルトの設定である**行の値**を使用するケースがほとんどでしょう。行バージョン列はチェックサムより実装が容易なので、Oracle APEX以外の処理がある場合は採用を検討することになるかと思います。例えば、Oracle REST Data Servicesを使ったRESTサービスによる更新などです。

最後に**行のロック**について説明します。失われた更新の防止の下にあり、ヘルプの記載だけではその意味が非常に掴みにくい設定です。

先ほど、チェックサムの一致を確認する手順を記述しました。フォームの送信に含まれるチェックサムをプロセスが受け取ると、

1. 更新対象の行を読み出して、チェックサムを計算する。
2. 受信したチェックサムと計算したチェックサムを比較する。
3. 一致していれば、受信したページ・アイテムで行を更新する。

という順番で処理が行われます。**行のロック**が**Yes**である場合、1 の処理でチェックサムを計算するためにデータベースからデータを取り出す際に"SELECT FOR UPDATE"を実行し、行を排他ロックします。非常に短い瞬間ですが、チェックサムを計算してから実際に行のアップデートを行う間に行が更新されないように保護しています。ですので、できる限りYesにします。

フォームのソースが単純な表やSQLではなく、SELECTは実行できるがFOR UPDATEをつけるとエラーが発生する(例えばソースとなるSQLが複数の表をJOINしている)場合、またはデータ・ソースがリモート・データベースなどの場合に**No**を設定します。Noにする弊害が明確な場合は、**PL/SQL Code**を選択し、排他制御を行うためのコードを記述することができます。

Oracle APEXの楽観的並行性制御の実装についての説明は以上です。Oracle APEXの良いところは、Oracle APEXでアプリケーションを作成すると、意識しなくても、今まで説明してきた機能が実装されている、というところでしょう。

完

Yuji N. 時刻: 18:09

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