TP3 : PL/SQL

But du TP :

- Gestion des exceptions

On reprend le schéma relationnel des TPs précédent :

```
Support (idSupport, nomSupport, TVA)
Concert (idConcert, nomConcert, lieuConcert, dateConcert)
InterpreterEnConcert (#idConcert,#idChanson, #idArtiste)`
Album (idAlbum, titreAlbum, anneeAlbum, label)
Chanson (idChanson, titreChanson, anneeChanson, #idAuteur, #idCompositeur)
Artiste (idArtiste, nomArtiste, prenomArtiste)
AppartenirGroupe (#idArtiste, #idGroupe, dateDebut, dateFin)
Piste (idPiste, noPiste, duree, #idChanson, #idAlbum)
Interpréter (#idPiste, #idArtiste)
DisponibleSurSupport (#idAlbum, #idSupport, prixDeVente)
```

[Q1] On reprend la procédure PL/SQL de la question Q1 du TP précédent (`ajoutConcert`). Il s'agissait d'insérer dans la table `Concert` et la table `InterpreterEnConert` un concert et son contenu (en paramètre : `idConcert, nomConcert, lieuConcert, dateConcert` et `nomArtiste, prenomArtiste`).

Vous traiterez les exceptions suivantes, en donnant dans l'exception un message d'erreur **clair** à l'utilisateur :

1. `nomArtiste et prenomArtiste` ne correspondant à aucun artiste existant (erreur SQL prédéfinie : `NO_DATA_FOUND` dans la table `Artiste`)
2. identifiant de concert (`idConcert`) déjà existant (erreur SQL pré-définie : `DUP_VAL_ON_INDEX`)
3. aucune chanson ne correspond à ce concert (erreur applicative : tester la présence d'au moins une chanson interprétée par le groupe sur un album de l'année du concert ou antérieur).
   Aide : pensez à utiliser un `select… count(*)`

Aucune insertion ne doit être faite dans les tables `Concert` et `InterpreterEnConcert` si l'une des exceptions précédente est levée.
Vous testerez bien entendu votre programme pour 'lever' toutes les exceptions possibles.

[Q2] Reprendre la procédure stockée `MAJPiste` de la question Q4 du TP précédent. Cette procédure, pour un identifiant et un numéro de piste passés en paramètre :

- affiche les titres de la chanson et de l'album associés
- effectue la mise à jour du numéro de piste de la piste (remplacer l'ancien numéro de piste par celui passé en paramètre, pour l'identifiant de piste donné).

Vous traiterez les exceptions suivantes, en donnant dans l'exception un message d'erreur **clair** à l'utilisateur :

1. l'identifiant de piste n'existe pas (erreur SQL pré-définie)
2. le nouveau numéro de piste existe déjà pour cet album (erreur applicative). En d'autres termes, il ne peut par exemple pas y avoir deux fois la piste no 2 pour l'album 1.
3. numéro de piste non compris dans l'ensemble [1,40] (erreur SQL non-prédéfinie).

Remarques :

- Pour récupérer le code erreur de la troisième exception, pensez à mettre une exception `OTHERS` qui affichera le `SQLCODE` et le `SQLERRM` de l'exception que vous voulez traiter (il suffira de provoquer l'exception).
- La deuxième exception aurait pu être gérée via une contrainte `UNIQUE` sur le couple `(noPiste, idAlbum)` de la table `Piste`. Si une telle contrainte avait existé, il aurait fallu gérer l'exception comme la troisième, en récupérant le code erreur généré.
- La mise à jour de la table `Piste` ne doit pas être faite si l'une des exceptions est levée.

[Q3] Ecrire une procédure stockée `AjoutChanson` permettant d'insérer un tuple dans la table `Chanson`. Cette procédure prendra en paramètre tous les champs de la table `Chanson`.
Vous gèrerez toutes les exceptions liées aux clés primaires et étrangères, en indiquant un message d'erreur clair à l'utilisateur.

Remarque importante : les exceptions de clés étrangères peuvent être gérées soit en utilisant l'exception `NO_DATA_FOUND`, soit avec l'erreur Oracle SQL non-prédéfinie -2291 (largement préférable). Dans ce dernier cas, afin de différencier les messages d'erreur, il est possible d'utiliser la syntaxe suivante :

```
if sqlerrm like
             '%(votre_nom_utilisateur.nom_de_la_contrainte_qui_est_transgressee)%'
then  ….
elsif sqlerrm like …
end if ;
```