TP 3 : Vues, contraintes d'intégrité et déclencheurs sous Oracle

L'objectif de ce TP est d'utiliser les mécanismes de vues, contraintes d'intégrité et déclencheurs disponibles sous Oracle pour améliorer la base de données du TP1.

A - Utilisation des vues

Création

Oracle supporte la création de vues telle qu'elle est spécifiée dans la norme SQL. La syntaxe utilisée est :

```
CREATE OR REPLACE VIEW nom_vue AS requête
```

La requête peut elle même référencer d'autres vues.

Questions

Les vues peuvent être très utiles pour réaliser des requêtes complexes.

A-1. Créez une vue correspondant à la requête E-5 du TP1. A partir de cette vue, écrivez une requête qui affiche le nombre moyen de buts lors de la saison 2015/2016 pour une ligue d'un championnat européen et le nombre moyen de buts pour une journée de championnat (vous ferez l'hypothèse que, pour chaque ligue, il y a 38 journées de championnat) .

A-2. Exécutez le fichier `view.sql` (sur `https://www.lias-lab.fr/ftppublic/enseignement/Oracle/L3/`). La vue `Equipe_Point` a été créée. Elle liste pour chaque équipe du championnat de ligue 1 2014/2015 les points qu'elle a obtenu pour chaque match joué. A partir de cette vue, créez-en une autre affichant le classement du championnat de ligue 1 2014/2015 (équipe, points, nombre de matchs joués). Vous afficherez le résultat par ordre décroissant de nombre de points. Vous vérifierez que vous obtenez la même chose que vous donne Google avec la requête `classement ligue 1 2014/2015`.

A-3. complétez les vues précédentes pour afficher les autres informations que vous donne Google (nombre de victoires, défaites, matchs nuls, buts marqués, buts encaissés et différence de buts).

Mises à jour de vues simples

Les vues peuvent aussi être très utiles pour restreindre l'accès aux données à un utilisateur. Les utilisateurs n'utilisent alors plus que des vues (et non des tables). Oracle permet de mettre à jour des vues simples, c'est-à-dire portant sur une seule table.

Questions

A-3. Créez une vue retournant tous les joueurs nés après 1990.

A-4. Testez la mise à jour (`INSERT`, `UPDATE` et `DELETE`) de la vue créée précédemment

A-5. Dans la définition d'une vue, une contrainte peut être ajoutée par la syntaxe :

```
CREATE OR REPLACE VIEW nom_vue AS requête
WITH CHECK OPTION CONSTRAINT nom_contrainte
```

Celle-ci spécifie qu'un tuple ne peut être inséré à travers cette vue que si celle-ci permet de retrouver ce tuple. Tester la mise à jour de la vue créée précédemment avec cette option.

Mise à jour de vues portant sur une requête avec jointure

Si l'option `WITH CHECK OPTION` est spécifiée sur une vue contenant une jointure, l'insertion d'un tuple au travers de cette vue est impossible. Sinon, la mise à jour de cette vue est possible sous certaines conditions.

Questions

A-6. Créez une vue qui affiche l'identifiant et le nom de chaque équipe ainsi que le nom et le pays de la ligue dans laquelle elle joue.

A-7. Essayez de mettre à jour la vue précédente. Quelles sont les opérations possibles sur chacun des attributs sélectionnés dans la vue ?

A-8. Vérifiez vos résultats grâce à la table `USER_UPDATABLE_COLUMNS` de la métabase d'oracle.

Utilisation des vues matérialisées (ou concrètes)

Les vues matérialisées (ou concrètes) permettent de stocker le résultat d'une requête. Nous allons utiliser cette structure pour optimiser une requête réalisant une jointure entre les tables `Equipe` et `Match`. La syntaxe pour créer une vue matérialisée est la suivante :

```
CREATE MATERIALIZED VIEW nom_vue
TABLESPACE USERS
BUILD IMMEDIATE
REFRESH type_de_rafraîchissement
ENABLE QUERY REWRITE
AS requête
```

Deux types de rafraîchissement sont possibles :

- `fast` : mise à jour incrémentale (seulement pour les vues sans jointure). Elle peut être effectuée à la demande (`ON DEMAND` via l'instruction `EXECUTE DBMS_MVIEW.REFRESH('nom_vue')`) où lorsqu'une validation de la transaction est effectuée (`ON COMMIT`) ;
- `complete` : mise à jour complète.

A-9. Notez le plan d'exécution de la requête suivante :

```
SELECT Equipe.nom_equipe, MATCH.dateMatch
  FROM Equipe, MATCH
 WHERE Equipe.equipeid = MATCH.equipelocale
```

A-10. Créez une vue matérialisée qui correspond à la requête précédente ;

A-11. Exécutez la requête de la question A-9. Normalement, l'optimiseur devrait directement utiliser la vue matérialisée (donc lire une table) au lieu de calculer la jointure (d'où l'optimisation).

A-12. Créez différentes versions d'une vue matérialisée pour la requête suivante. Ces versions correpondront aux différent types de rafraîchissement `fast` et `complete` et aux deux options `ON DEMAND` et `ON COMMIT` (vous devriez donc avoir 4 versions de la vue matérialisée).

```
SELECT ligueid, nom_ligue
  FROM Ligue;
```

Pour un rafraîchissement de type fast, il est nécessaire de créer un journal de la vue matérialisée (pour faire de l'incrémental). La commande est la suivante :

```
CREATE MATERIALIZED VIEW LOG ON  nom_table_base;
```

A-13. Montrez par des exemples à quel moment ces vues sont mises à jour lorsque des données sont ajoutées dans les tables utilisées par cette vue.

B - Contraintes d'intégrité

Dans le TP1, vous avez défini des contraintes de clés primaires et étrangères pour favoriser l'intégrité de la base de données. Oracle propose également les contraintes suivantes :

- indiquer qu'un attribut se comporte comme une clé primaire par la contrainte `UNIQUE` ;
- indiquer qu'un attribut est obligatoire par la contrainte `NOT NULL` ; il est alors souvent utile de définir une valeur par défaut (`DEFAULT`) pour cet attribut ;
- vérifier une condition portant sur les attributs du tuple inséré par la contrainte `CHECK`.

La syntaxe pour définir ce type de contrainte est la suivante :

```
ALTER TABLE nom_table
  ADD CONSTRAINT nom_contrainte UNIQUE(colonne);

ALTER TABLE nom_table
  ADD CONSTRAINT nom_contrainte CHECK(condition);
```

Questions

Définissez les contraintes suivantes sur la base de données du TP1 (si des données ne les satisfont pas, vous les modifierez). Pour chaque contrainte définie vous effectuerez des tests pour en vérifier la validité.

B-1. les joueurs ont une taille comprise entre 1,5m et 2,5m ;

B-2. le nom et le prénom des joueurs doit être précisés ;

B-3. la date de chaque match doit être précisée ;

B-4. chaque pays n'a qu'une seule ligue ;

B-5. le nom d'une équipe est différent de son nom court ;

B-6. Ajoutez une colonne `poste` à la table joueur et indiquez que le poste d'un joueur ne peut être que l'une des 4 valeurs suivantes : `Gardien`, `Défenseur`, `Milieu` ou `Attaquant`. Par défaut, le poste d'un joueur créé est `Défenseur` ;

B-7. un joueur à au moins 14 ans (par rapport à la date du 1er janvier 2008) ;

B-8. une équipe ne peut pas jouer contre elle-même ;

B-9. les dates de match sont comprises entre juin 2008 et juin 2018 ;

B-10. soit le score d'un match est inconnu, soit on connaît le nombre de buts marqués par les deux équipes.

Lorsque l'on souhaite insérer l'ensemble des joueurs du championnat, il peut être utile de différer la vérification d'une contrainte afin d'optimiser le traitement. Oracle permet de faire ceci pour les contraintes définies avec le qualificatif `DEFERRABLE`. Au cours d'une transaction, il suffit alors de saisir :

```
SET CONSTRAINT nom_contrainte DEFERRED;
```

pour retarder sa vérification jusqu'à la validation de la transaction (`COMMIT`).

Une autre solution est d'activer ou désactiver une contrainte :

```
ALTER TABLE nom_table
      MODIFY CONSTRAINT nom_contrainte DISABLE -- ou ENABLE
```

Questions

B-11. Différez la contrainte portant sur l'age des joueurs. Que se passe-t-il si un joueur ne respectant pas cette contrainte est inséré.

B-12. Même question que la précédente mais en utilisant le mécanisme de désactivation des contraintes.

B-13. En utilisant les résultats des questions précédentes, comparez les deux mécanismes permettant de retarder la vérification d'une contrainte en termes de temps d'exécution et d'intégrité de la base de données.

C - Déclencheurs (Triggers)

Les contraintes ne peuvent pas toutes s'exprimer avec les mécanismes de base (`CHECK`, `UNIQUE`, etc.). Dans ce cas, des déclencheurs peuvent être utilisés. Un déclencheur définit un traitement à effectuer avant ou après un ordre `INSERT`,`UPDATE`, `DELETE` sur une table. Cet ordre peut être exécuté une seule fois pour la modification d'une table ou pour chaque ligne modifiée.

La syntaxe de définition d'un déclencheur est la suivante :

```
CREATE OR REPLACE TRIGGER nom_trigger
  BEFORE -- ou AFTER
    INSERT OR DELETE OR UPDATE ON nom_table
  REFERENCING NEW AS alias_nouveau_tuple
              OLD AS ancien_tuple
  FOR EACH ROW WHEN (condition)
  -- la ligne précédente est à mettre si on veut
  -- que le déclencheur s'exécute pour chaque ligne modifiée
BEGIN
  ... -- code PL/SQL (~ADA)
END;
/
```

Les variables `NEW` et `OLD` permettent de consulter ou de modifier le tuple qui a provoqué l'exécution du trigger. Un alias peut être défini sur ces variables dans la clause `REFERENCING`. Dans la partie entre `BEGIN` et `END` ces variables doivent être préfixées par le caractère deux-points (:)

La compilation du déclencheur peut lever des erreurs syntaxiques. Ces erreurs peuvent être affichées par la commande :

```
SHOW ERRORS
```

Pour supprimer un déclencheur la commande est :

```
DROP TRIGGER nom_trigger
```

Pour le désactiver ou l'activer :

```
ALTER TRIGGER nom_trigger DISABLE -- ou ENABLE
```

Questions

C-1. Créez un déclencheur qui affiche un message pour l'administrateur dès que la table contenant les joueurs est modifiée.

C-2. Créez un déclencheur qui affiche le score d'un match (par exemple, AS Monaco 0-0 Olympique de Marseille) dès que celui-ci est connu.

C-3. Ajoutez une colonne `nbVictoire` à la table `Joueur`, initialisée à 0 par défaut. Créez un déclencheur qui augmente le nombre de victoire d'un joueur dès que son équipe gagne un match.

C-4. Créez un déclencheur qui, lorsqu'un score est saisi concernant Bordeaux, augmente (si nécessaire) le nombre de buts que cette équipe a marqué pour qu'elle remporte le match.

C-5. Créez un déclencheur pour vérifier la contrainte suivante : un joueur qui marque un but dans un match doit appartenir à l'une des deux équipes qui a joué le match.

Pour afficher un message, vous utiliserez la fonction `DBMS_OUTPUT.PUT_LINE(chaîne à afficher)` après avoir attribué la valeur `ON` à l'option `SERVEROUTPUT`.