❽ Périgée : Logiciel de gestion des notes de l'IUT de Saint Denis

On s'intéresse ici au logiciel Périgée qui permet de gérer les notes des étudiants du département Informatique de l'IUT de Saint-Denis de la Réunion. Ce logiciel repose sur une base de données Oracle dont le schéma relationnel vous est communiqué ci-dessous.

PROMOTIONS (idPromotion, nomPromotion, *nbEtudiantsPromotion*)
GROUPES (idGroupe, idPromotion#)
ETUDIANTS (idEtudiant, nomEtudiant, prenomEtudiant, sexeEtudiant, dateNaissanceEtudiant, idGroupe#)
SEMESTRES (idSemestre, dateDebutSemestre, dateFinSemestre, idPromotion#)
ABSENCES (idAbsence, idEtudiant#, dateHeureDebutAbsence, dateHeureFinAbsence)
JUSTIFICATIFSABSENCES (idJustificatifAbsence, idEtudiant#, dateDebutJustificatif, dateFinJustificatif, motifJustificatif)
MODULES (idModule, nomModule, idSemestre#, *coefficientModule*)
MATIERES (idMatiere, nomMatiere, idModule#, coefficientMatiere)
NOTES (idEtudiant#, idMatiere#, note)

La table *Promotions* recense les différentes promotions d'étudiants de l'IUT (en l'occurrence, la promotion de première année et celle de seconde année) et la table *Groupes* inventorie tous les groupes de TD de ces promotions (les groupes Q1, Q2, Q3 et Q4 en première année, et les groupes D1 et D2 en seconde année). Dans la table *Etudiants*, on peut trouver toutes les informations qui concernent les étudiants qui sont affectés à ces groupes de TD.

La table *Semestres* nous indique à quelles dates commencent et se terminent les semestres 1, 2, 3 et 4 et à quelle promotion ils sont rattachés ; par exemple, les semestres 1 et 2 sont rattachés à la première année, et les semestres 3 et 4 à la seconde année. La table *Absences* permet de connaître la date et l'horaire des cours pour lesquels les étudiants ont été contrôlés absents ; avec pour chaque absence, la date et l'heure de début du cours (`dateHeureDebutAbsence`) et de fin du cours (`dateHeureFinAbsence`). Il est à noter que si lors d'une journée, un étudiant a été absent à plusieurs cours, il pourra avoir une absence pour chacun de ces cours (à condition que l'appel ait été fait).
Dans la table *JustificatifsAbsences* sont répertoriés les justificatifs d'absence que les étudiants ont donnés au secrétariat de l'IUT. Un justificatif d'absence possède une date de début (`dateDebutJustificatif`), une date de fin (`dateFinJustificatif`) et un motif d'absence (`motifJustificatif`).

Dans la table *Modules* on peut trouver tous les modules dispensés pendant les différents semestres. Par exemple, le module 111 "Initiation Informatique" est dispensé au premier semestre (de la première année) ; alors que le module 211 "Programmation Web" est dispensé au troisième semestre (de la seconde année).
La table *Matières* permet de connaître les matières qui sont enseignées (et donc évaluées) à l'intérieur d'un module. Par exemple, le module 212 "Culture Générale Informatique" (du semestre 1 de la deuxième année) comprend les matières « UE15 Refactoring Agile », « UE16 Qualité & Tests » et « UE17 eXtreme Programming ». Enfin, la table *Notes* permet de connaître les notes qu'ont obtenues les étudiants dans les différentes matières.

Vous trouverez sur le Moodle le fichier « Périgée.sql » qui vous permettra de générer les tables de la base de données sur votre schéma.

Dans les tables *Promotions* et *Modules* figurent respectivement les attributs `nbEtudiantsPromotion` et `coefficientModule` qui sont indiqués en italique dans le schéma relationnel. Ces attributs sont dits dérivés (au sens UML), c'est-à-dire qu'ils sont calculables à partir de requêtes SQL. Ils engendrent donc de la redondance dans la base de données mais ont tout de même été maintenus car leur contenu est relativement stable. Toutefois, dans le script « Périgée.sql » leur valeur n'a pas encore été initialisée et sur toutes les lignes, ces attributs ont pour le moment la valeur `NULL`.

Première partie – Les Blocs PL/SQL :

1) Bloc PL/SQL simple et paquetage `DBMS_OUTPUT`.
   - Ecrire un bloc PL/SQL qui, grâce au paquetage `DBMS_OUTPUT`, affiche le nombre d'étudiants qui se trouvent dans le groupe qui a pour identifiant 'Q1'.

     On rappelle que pour que le paquetage `DBMS_OUTPUT` fonctionne dans l'interface iSQL*Plus, il faut l'activer avec la commande `SET SERVEROUTPUT ON` avant la première instruction du bloc PL/SQL (cette option sera valable durant toute la session iSQL*Plus).
   - Résultat attendu :

```
Il y a 6 étudiant(s) dans le groupe Q1
```

2) Interactivité grâce aux variables de substitution.
   - En s'inspirant du bloc PL/SQL écrit lors de la question précédente, écrire un nouveau bloc qui affiche le nombre d'étudiants d'un groupe dont l'identifiant a été saisi au clavier.

     Pour cela on utilisera les variables de substitution vues en cours. On rappelle qu'à l'intérieur du `BEGIN … END`, les variables de substitution qui font référence à une chaîne de caractères doivent être encadrées par des 'quotes'. Par exemple, `'&s_idGroupe'`
   - Résultats attendus :

```
Entrer l'identifiant du groupe : Q1
Il y a 6 étudiant(s) dans le groupe Q1

Entrer l'identifiant du groupe : Q2
Il y a 5 étudiant(s) dans le groupe Q2

Entrer l'identifiant du groupe : Q4
Il y a 0 étudiant(s) dans le groupe Q4
```

3) Structures de contrôle : la conditionnelle (`IF … THEN … ELSE … END IF`).
   - Dans la question précédente il y a un léger problème. En effet, si l'utilisateur saisit un identifiant d'un groupe qui n'existe pas (par exemple le Q5), il est indiqué qu'il y a 0 étudiant dans le groupe en question. Et on aurait eu la même réponse si on avait saisi l'identifiant d'un groupe qui existe bien mais qui n'a aucun étudiant affecté (le Q4).

     En s'inspirant du bloc PL/SQL écrit lors de la question précédente, écrire un nouveau bloc qui indiquerait le cas échéant qu'aucun groupe ne possède l'identifiant qui a été saisi au clavier.

     Pour cela, vous utiliserez la structure de contrôle (`IF … THEN … ELSE … END IF`)
   - Résultats attendus :

```
Entrer l'identifiant du groupe : Q4
Il y a 0 étudiant(s) dans le groupe Q4

Entrer l'identifiant du groupe : Q5
Il n'y a pas de groupe Q5
```

4) Les Exceptions : l'exception `NO_DATA_FOUND`.
   - Ecrire un nouveau bloc PL/SQL qui fait exactement la même chose que la question précédente mais en utilisant cette fois-ci la section `EXCEPTION` à la place de la structure de contrôle `IF`.

     On pourra utiliser l'exception prédéfinie `NO_DATA_FOUND` qui est levée automatiquement lorsque une requête `SELECT … INTO` ne retourne rien (il est à noter que si la requête retourne plusieurs lignes, c'est alors l'exception `TOO_MANY_ROWS` qui est levée).

5) Les variables %ROWTYPE.
   - Ecrire un bloc PL/SQL qui permet d'afficher toutes les informations nominatives de la table *Etudiants* qui concernent l'étudiant E1.

     Pour se simplifier la tâche, et pour faciliter les éventuelles futures maintenances de l'application, on récupèrera les données qui concernent l'étudiant E1 dans une variable %ROWTYPE.

   - Résultat attendu :

```
Identifiant étudiant : E1
Nom étudiant : Alizan
Prénom étudiant : Gaspard
Sexe étudiant : M
Date naissance étudiant : 14/07/01
Groupe étudiant : Q1
```

Deuxième partie – Les Procédures et Fonctions stockées :

Fonctions stockées

6) Fonction stockée nbEtudiantsParGroupe.
   - En vous inspirant de ce que vous avez écrit lors des questions 3 ou 4, écrire une fonction stockée nbEtudiantsParGroupe qui retourne le nombre d'étudiants qui appartiennent au groupe dont l'identifiant p_idGroupe est passé en paramètre. Cette fonction doit avoir la signature suivante :

```
FUNCTION nbEtudiantsParGroupe(p_idGroupe IN Groupes.idGroupe%TYPE)
                                                    RETURN NUMBER
```

     Si on passe en paramètre de cette fonction un identifiant de groupe qui n'existe pas, on souhaite que la fonction retourne NULL (et non pas 0).

     Si lors de la création de votre fonction, vous avez une erreur de compilation (Warning: Function created with compilation errors) vous pouvez demander à Oracle de vous afficher la liste des erreurs grâce à l'instruction SHOW ERRORS.

     On rappelle que sous Oracle, pour tester une fonction stockée, on peut utiliser la pseudo-table DUAL.

   - Résultats attendus :

```
SELECT nbEtudiantsParGroupe('Q1')
FROM DUAL ;

nbEtudiantsParGroupe('Q1')
--------------------------
6

SELECT nbEtudiantsParGroupe('Q5')
FROM DUAL ;

nbEtudiantsParGroupe('Q5')
--------------------------
// quand le résultat est NULL, par défaut, iSQL*Plus n'affiche rien
```

7) Fonction stockée nbEtudiantsParPromotion.
   - Ecrire une fonction stockée nbEtudiantsParPromotion qui retourne le nombre d'étudiants qui appartiennent à une promotion dont l'identifiant p_idPromotion est passé en paramètre. Cette fonction doit avoir la signature suivante :

```
FUNCTION nbEtudiantsParPromotion(
                    p_idPromotion IN Promotions.idPromotion%TYPE)
                                                    RETURN NUMBER
```

     Dans le corps de cette fonction, on pourra appeler la fonction qui a été réalisée à la question précédente (nbEtudiantsParGroupe). Cette fonction pourra être appelée dans le SELECT d'une requête !

On ne vous demande pas de gérer explicitement le cas où l'identifiant de promotion qui est passé en paramètre n'existe pas, ou bien le cas où la promotion existe mais qu'elle n'a pas de groupe (dans ces cas là, votre fonction devrait retourner `NULL` par défaut sans que vous soyez obligé de le programmer).

- Résultat attendu :

```
SELECT nbEtudiantsParPromotion('A1')
FROM DUAL ;

nbEtudiantsParPromotion('A1')
-----------------------------
16
```

- Ecrire une requête SQL qui, en utilisant la fonction écrite ci-dessus, met à jour l'attribut `nbEtudiantsPromotion` de toutes les lignes de la table *Promotions*. Puis, vérifier que dans la table *Promotions* on a bien 16 étudiants en A1 et 11 en A2.

Procédures stockées

8) Procédure stockée affichageInfosEtudiant.
   - En vous inspirant de ce que vous avez écrit lors de la question 5, écrire une procédure stockée `affichageInfosEtudiant` qui affiche toutes les informations nominatives qui concernent l'étudiant dont l'identifiant `p_idEtudiant` est passé en paramètre. Cette procédure doit avoir la signature suivante :

```
PROCEDURE affichageInfosEtudiant(
                              p_idEtudiant IN Etudiants.idEtudiant%TYPE)
```

   On rappelle que sous Oracle, on peut appeler une procédure stockée avec l'instruction `CALL`.

   - Résultat attendu :

```
CALL affichageInfosEtudiant('E1');

Identifiant étudiant : E1
Nom étudiant : Alizan
Prénom étudiant : Gaspard
Sexe étudiant : M
Date naissance étudiant : 14/07/01
Groupe étudiant : Q1
```

9) Procédure stockée miseAJourCoefficientModules.
   - Ecrire une procédure stockée qui met à jour l'attribut `coefficientModule` de toutes les lignes de la table *Modules*. Le coefficient d'un module doit être égal à la somme des coefficients des matières qui sont incluses dans le module. Cette procédure doit avoir la signature suivante :

```
PROCEDURE miseAJourCoefficientModules
```

   - Résultat attendu :

```
CALL miseAJourCoefficientModules() ;


SELECT idModule, coefficientModule
FROM Modules

idModule  coefficientModule
--------- ------------------
M111      6
M112      6
M113      3
M121      6
M122      6
M123      3
M211      6
M212      6
M213      3
M221      6
M222      6
```