Licence Informatique - Sorbonne Université

LU2IN009 - Bases de Données

Support de TME 2023

Table des matières

Site web : https://moodle-sciences-22.sorbonne-universite.fr/course/view.php?id=3855

masquer: 1

TME 4 : PREMIERS PAS EN SQL

1. SGBD H2

Le système utilisé pendant les TME est H2.

1. Commencez par vous connecter au serveur H2 en vous aidant de la documentation suivante: SGBD H2
2. Créer les tables et charger les données: BD-JeuxOlympiques-v1

On considère le schéma de la base JeuxOlympique2014 donné en TD où l'on a modifié le schéma de **RangEpreuve**:

- **Athlete** (*nom,prenom*,dateNaissance,pays)
- **RangEpreuve**(*sport,epreuve,categorie,nomAth*,prenomAth**,dateDebut,dateFin,rang)

2. PREMIÈRES REQUÊTES SOUS H2

Renvoie la liste des attributs de la relation `Athlete`.

2.1 Tous les n-uplets stockés dans la relation `Athlete`.

```
select * from Athlete;
```

2.2 Le nombre de n-uplets stockés dans la relation `Athlete`.

```
select count(*) from Athlete;
```

3. REQUÊTES

Exercice 1:

Exprimer les requêtes suivantes en SQL :

3.1 Les athlètes (nom, prénom) d'Inde

Résultat : *IQBAL Nadeem, KESHAVAN Shiva, THAKUR Himanshu*

3.2 Le prénom des athlètes dont le nom est 'GOLD'.

Résultat : *Arielle, Gracie, Taylor*

3.3 La nationalité de AONO Ryo.

Résultat : *Japon*

3.4 Le gagnant du médaille d'or de chaque épreuve. Triez le résultat par sport, puis par épreuve.

Résultat *: (74 lignes)*

3.5 Toutes les épreuves (sport, épreuve, catégorie) triées par la date de fin de l'épreuve

Résultat : *(73 lignes)*

3.6 Les athlètes nés entre 1990 et 1995.

Résultat : *(87 lignes)*

3.7 Les athlètes suisses ayant participé au sport 'Biathlon' et disqualifié à au moins une épreuve dece sport (solution sans jointure, avec une opération ensembliste).

Résultat : *GASPARIN Elisa*

3.8 Les épreuves dans lesquelles il n'y a pas eu une médaille d'argent (solution sans jointure, avec une opération ensembliste).

Résultat : Ski alpin Descente Femmes

Exercice 2:

On considère maintenant le schéma complet de la base JO2014

PAYS (CODEPAYS, NOMP)
SPORT (SID, NOMSP)
EPREUVE (EPID, SID*, NOMEP, CATÉGORIE, DATEDEBUT, DATEFIN)
ATHLETE (AID, NOMATH, PRENOMATH, DATENAISSANCE, CODEPAYS*)
EQUIPE (EQID, CODEPAYS*)
ATHLETESEQUIPE (EQID*, AID*)
RANGINDIVIDUEL (EPID*, AID*, RANG)
RANGEQUIPE (EPID*, EQID*, RANG)

La relation **PAYS** contient le code et le nom de tous les pays, même si ils n'ont pas participé aux Jeux Olympiques. Les sports (n-uplets de la relation **SPORT**) sont un ensemble d'épreuves (n-uplets de la relation **EPREUVE**). Pour chaque épreuve on connaît son nom et les date de début et fin de l'épreuve. Les épreuves peuvent être individuelles ou par équipe. Dans le premier cas, la participation des athlètes (n-uplets de la relation **ATHLETE**) est stocké dans la table **RANGINDIVIDUEL** qui contient en plus le rang qu'ils ont obtenu (1 pour la médaille d'or). Pour les épreuves par équipe les résultats sont stockés dans la relation **RANGEQUIPE**, alors que l'information sur le pays de chaque équipe et ses participants et stocké dans les relations **EQUIPE** et **ATHLETESEQUIPE**. Dans les relations **RANGINDIVIDUEL** et **RANGEQUIPE** l'attribut rang est égal à null si l'athlète ou l'équipe a été disqualifié.

Le schéma E/A pour la BD JO_v2 se trouve ici.

Créer les tables et charger les données: BD-JeuxOlympiques-v2

Exprimer les requêtes suivantes en SQL :

3.9. Les épreuves (sport, épreuve, catégorie) individuelles.

Résultat : (73 lignes)

3.10. Les résultats (nom, prénom, pays de l'athlète et rang) de l'épreuve '1000m' du 'Patinage de vitesse' pour les 'Femmes'.

Résultat : (36 lignes)

3.11. Le nom et prénom des athlètes qui forment l'équipe qui a gagné la médaille d'or dans l'épreuve 'relais 4x6km' de 'Biathlon' de 'Femmes'.

Résultat : SEMERENKO Vita, SEMERENKO Valj, DZHYMA Juliya, PIDHRUSHNA Olena

TME 5 : JOINTURES ET IMBRICATION AVEC IN ET EXISTS

masquer=1

BASE DE DONNÉES « JEUX OLYMPIQUES D'HIVER 2014 »

On considère le schéma de la base JEUXOLYMPIQUE2014 donné en TD :

1. Les sports auxquels LESSER Erik a participé en supposant qu'il a participé aux épreuves individuelles., r.epid

Résultat : Ski de fond, Biathlon

2. Les athlètes ayant participé aux épreuves individuelles de (au moins) deux sports différent.

Résultat : PETROVIC Milanko, TER MORS Jorien, LESSER Erik, PEIFFER Arnd

3. Les dates de début et de fin des Jeux Olympique 2014.

Résultat : 06/02/2014 – 23/02/2014

4. Les pays qui ont des participants de moins de 18 ans ou de plus que 40 au 24-02-2016 (Attention : il faut éliminer les athlètes dont on ne connaît pas la date de naissance).

Résultat : AUT Autriche, CAN Canada, GER Allemagne, ITA Italie, JPN Japon, NOR Norvège, RUS Russie

5. Les équipes n'ayant pas d'athlète (la base de données ne contient pas l'information sur les participants)

Résultat : (13 lignes)

6. Les équipes qui ont exactement 2 athlètes. Retourner l'équipe en question avec ses deux seuls athlètes.
Résultat : (114 lignes)

7. Les sports qui n'ont pas d'épreuves de catégorie Mixte.

Résultat : (12 lignes)

8. Les athlètes qui ont gagné une médaille d'or (au moins) mais pas de médaille d'argent ni de bronze

Résultat : (53 lignes)

9. Les athlète(s) qui ont fini dernier d'une épreuve individuelle. Indiquer leur nom, prenom, le nom du sport et de l'épreuve et leur rang. Attention : il faut filtrer les athlètes disqualifiés.

Résultat : (70 lignes si on projette sur nomAth, prenomAth, ou 71 si on rajoute rang)

masquer=1

TME 6 : Requêtes Imbriquées avec Exists, All et Any

Base de Données « Jeux Olympiques d'Hiver 2014 »

On considère le schéma de la base JeuxOlympique2014 donné en TD :

Pays (codePays, nomP)
Sport (SID, nomSp)
Epreuve (EPID, SID*, nomEp, catégorie, dateDebut, dateFin)
Athlete (AID, nomAth, prenomAth, dateNaissance, codePays*)
Equipe (EQID, codePays*)
AthletesEquipe (EQID*, AID*)
RangIndividuel (EPID*, AID*, RANG)
RangEquipe (EPID*, EQID*, RANG)

A. Tester les requêtes vues en TD sur la BD JO et finir tous les exercices.
B. Exprimer les requêtes suivantes en SQL :

1. Les sports dont toutes les épreuves ont duré un seul jour.
Résultat : Ski de fond, Ski alpin, Biathlon

2. Les sports qui n'ont pas d'épreuves de categorie Mixte.
Résultat : (12 lignes)

3. Les équipes dont aucun athlète n'a gagné de médaille aux épreuves individuelles.
Attention : il y a des équipes sans athlètes.
Résultat : (265 lignes avec les équipes sans athlètes - 252 lignes sans les équipes sans athlètes)

4. La nationalité de l'athlète le/la plus jeune. Attention : il y a des athlètes dont on ne connaît pas la date de naissance.
Résultat : ('29/11/1998','JPN')

5. Le plus jeune athlète de chaque pays.
Résultat : (26 lignes)

Base « Foofle »

Reprendre les requêtes du TD.

Les réponses :

7. Quelles équipes ont déjà joué au stade préféré de l'équipe des Piepla ?

Résultats : (2 lignes) Direkt , Piepla

8. Quels sont les stades où a déjà joué Manon Messi ?

Résultats: (3 lignes) GrandArena, Boulodrome, BukHall

9. A quelle date a eu lieu un match entre deux équipes sponsorisées par le même sponsor ?

Résultats :(4 lignes) 2015-05-16, 2015-05-15, 2015-06-15, 2015-05-12

10. Quel sponsor a financé deux joueurs différents ayant eu un match le même jour et dans des stades différents mais proches (moins de 50 km) ?

Résultats :(4 lignes) Air Monaco, Carouf, Robek, Adadis

masquer=1

TME 7 : REQUÊTES D'AGRÉGATION ET DIVISION

BASE « JEUX OLYMPIQUES D'HIVER 2014 »

On reprend le schéma schéma « Jeux Olympiques d'hiver 2014 ».

PAYS (codePays, nomP)
SPORT (SID, nomSp)
EPREUVE (EPID, SID*, nomEp, categorie, dateDebut, dateFin)
ATHLETE (AID, nomAth, prenomAth, dateNaissance, codePays*)
EQUIPE (EQID, codePays*)
ATHLETESEQUIPE (EQID*, AID*)
RANGINDIVIDUEL (EPID*, AID*, RANG)
RANGEQUIPE (EPID*, EQID*, RANG)

Écrivez et évaluez les expressions SQL pour répondre aux requêtes suivantes.

Fonctions d'agrégation « COUNT, SUM, AVG, MIN, MAX »

1. Le nombre d'athlètes.
   Résultat (1 ligne) : 2431
2. Le nombre d'athlètes ayant participé à au moins une épreuve en individuel.
   Résultat (1 ligne) : 1558
3. L'âge moyen des sportifs dont le code pays est 'FRA' (France) au 06/02/2014.
   Résultat (1 ligne) : 26,8
   Aide :
   - utilisez `round(valeur,nb)` pour garder seulement `nb` décimales à `valeur`
   - sous Oracle, utilisez: `to_date('06/02/2014','dd/mm/YYYY')`
4. La durée moyenne, minimale et maximale des épreuves.
   Résultat (1 ligne) : « Durée moyenne = 1,98 min = 1 max = 16 »
   Aide : utilisez l'opérateur de concaténation ||
   Attention : entre le 10/01/2014 et le 13/01/2014, il y a une durée de 4 jours (et non pas 3).
5. Le nombre moyen d'athlètes par pays, c'est-à-dire le nombre d'athlètes divisé par le nombre de pays (ayant au moins un athlète). Résultat (1 ligne) : 27,625

Partitionnement « group by »

6. Pour chaque pays, le nom du pays et le nombre d'athlètes, ordonner par nombre d'athlètes croissant.
   Résultats (88 lignes) : (PAK,1) ; (HKG, 1) ; … ; (USA, 196) ; (CAN,221)
7. Le nombre moyen d'athlètes par pays (avec `group by`). Aide : compter le nombre d'athlètes dans chaque pays (ayant au moins un athlète), puis faire la moyenne.
   Résultat (1 ligne) : 27,625
8. Pour chaque équipe, l'eqid de l'équipe et le nombre d'athlètes, ordonner par nombre d'athlètes décroissant.
   Résultats (296 lignes) : (164,25) ; (165,25) ; (166,25) ; … ; (180,2) ; (181, 2) ; (182, 2)
9. Pour chaque catégorie, la catégorie et le nombre d'épreuves.
   Résultats (3 lignes) : (Femmes,43) ; (Mixte,6) ; (Hommes,49)
10. Pour chaque sport, le nom du sport et le nombre d'épreuves, ordonner par nombre d'épreuves décroissant.
    Résultats (15 lignes) : (Patinage de vitesse,12) ; (Ski de fond,12) ; … ;(Hockey sur glace,2)

11. Pour chaque pays, le code du pays, le nombre de médailles en épreuve individuelle gagnées et le nombre d'athlètes ayant gagnés au moins une médaille. Ordonner par nombre de médailles décroissant. Aide : 2 tables seulement sont nécessaires.
    Résultats (24 lignes) : (NOR, 24,19) ; (NED,22,15) ; …
12. Pour chaque pays et sport, le code du pays, le sid du sport, le nombre de médailles en épreuve individuelle gagnées, le nombre d'athlètes ayant gagnés au moins une médaille, ordonner d'abord par code pays, puis par nombre de médailles décroissant.
    Résultats (84 lignes) : (AUS,12,2,2); (AUS,15,1,1);(AUT,13,9,7);(AUT,15,2,2);...

Partitionnement avec « group by / having »

13. L'eqid de la ou des équipes qui sont composées :
    a) d'exactement 10 athlètes. Résultat (1 ligne) : 226
    b) du plus d'athlètes pour ces JO.
    Résultats (3 lignes) : 164 ; 165 ; 166
14. Le nombre d'épreuves en individuel où il y a eu au moins 100 participants.
    Résultat (1 ligne) : 2
15. Le nom des pays qui ont gagné au moins 20 médailles aux épreuves individuelles.
    Résultats (3 lignes) : Pays-Bas ; États-Unis ; Norvège

Division en SQL

16. Le sid des sports qui ont des épreuves dans toutes les catégories existantes.
    Résultats (3 lignes) : 1 ; 6 ; 7
17. Le nom des pays qui ont participé aux épreuves en individuel de tous les sports en individuel. Résultats (3 lignes) : (Russie,12) ; (États-Unis,12) ; (Italie,12)

BASE « FOOFLE »

On reprend le schéma "Foofle":

Sponsorise(<u>nsp, njo</u>, somme),
Joueur(<u>njo</u>, eq, taille, age),
EquipeF(<u>neq</u>, ville, couleur, stp)
Match(<u>eq1, eq2</u>, dateM, st),
Dist(<u>st1, st2</u>, nbKm)

Écrivez les expressions SQL pour répondre aux requêtes suivantes :

1. Pour chaque sponsor, le nom du sponsor, le nombre de joueurs sponsorisés, le montant total des sommes versées, ordonner par sommes versées décroissantes.
   Résultats (7 lignes) : (Adadis,6,2340) ; (Robek,5,1426) ;...
2. Quelles équipes ont joué au moins dans 3 stades différents ?
   Résultats (2 lignes) : Fortiches ; Direkt
3. Quels sponsors sponsorisent exactement un joueur pour chaque équipe qu'il sponsorise ?
   Résultats (2 lignes) : Air Monaco ; Palasse
4. Quel est le nombre total de kilomètres parcourus par chaque équipe. On suppose qu'après chaque match, chaque équipe se rend directement au stade où aura lieu son prochain match (d'après la date du match). Aide : il existe 2 matchs ordonnés par leur date pour la même équipe, mais il n'existe pas un 3ième match entre les dates des 2 matchs pour cette équipe.
   Résultats (4 lignes) : (Fortiches,516) ; (Direkt,671) ; (Piepla,124) ; (Sabar,360)

masquer=1

TME 8 : CREATION DE SCHEMAS- CONTRAINTES D'INTEGRITE

Ce TME se base sur le TD8 et vise à illustrer la modification des tables et l'interaction avec les contraintes d'intégrité. On utilise le schéma Entreprise vu en TD.

Exercice 1:

Créez le schéma Entreprise vue en TD et associez aux tables les contraintes d'intégrité correspondantes.

Dans ce qui suit, il vous est demandé d'effectuer des insertions de n-uplets dans des tables.

Syntaxe des insertions :

```
insert into Table values ('val1', 'val2', ….) ;
```

par exemple, pour insérer une employée 'LARS Anna', qui habite 'Paris' et qui est née le 25- 08-1975, il suffit d'exécuter l'instruction ci-dessous

insert into employe values('21456','LARS', 'Anna',null, 'paris', parsedatetime('25-08-1975', 'dd-MM-yyyy'));

et de constater que la système retourne bien le message

Update count: 1

- Syntaxe des suppressions :

  ```
  delete from Table ;
  ```

  supprime tous les nuplets de la table (la table existera toujours mais sera vide)

Exercice 2:

Insérez dans chaque table au moins un n-uplet qui vérifie les contraintes d'intégrité. Vous avez la liberté de choisir les valeurs que vous voulez. Par exemple, l'instruction suivante insère un employé qui vérifie les contraintes

/*insertion d'un Employe*/

insert into employe(NUMSS,NOME,PRENOME, VILLEE , DATENAISS) values (12345, 'Smith', 'John', 'Paris', parsedatetime('25-08-1975', 'dd-MM-yyyy'));

Exercice 3:

Proposez des insertions qui violent les contraintes d'intégrité définies pour chaque table.

Par exemple, l'instruction ci-dessous:

```
insert into employe values(null,'LARS', 'Anna',null,'paris',
parsedatetime('25-08-1975', 'dd-MM-yyyy' ));
```

viole la contrainte de clé primaire de Employe car elle tente d'insérer un employé sans valeur pour l'attribut clé primaire

Si vous tentez d'exécuter cette instruction, vous constaterez l'erreur

NULL not allowed for column "NUMSS"; SQL statement:

Répondre aux questions suivantes :

1. Proposer une insertion dans la table Employé qui ne respecte pas la contrainte de clé primaire.

2. Proposer une insertion dans la table Employe qui ne respecte pas la contrainte de limite d'âge.

3. Proposer une insertion dans la table Employe qui ne respecte pas la contrainte de longueur de l'attribut NumSS.

4. Proposer une insertion dans la table Employe qui ne respecte pas la contrainte sur les villes possibles.

5. Insérer dans la table Employé deux employés avec le même nom et le même prénom.

6. Proposer une insertion dans la table Grille_SAL qui ne respecte pas la contrainte sur le salaire qui ne doit pas dépasser 90 000.

7. Proposer une insertion dans la table Projet qui ne respecte pas la contrainte référentielle vers Employe : insérer un responsable de projet qui n'est pas dans la table Employé

8. Proposer une insertion dans la table Embauche qui ne respecte pas une des contraintes référentielles : par exemple, associer un employé existant à un projet qui n'existe pas.

MASQUER=1

TME 9 : MODIFICATION DES DONNEES – CONTRAINTES D'INTEGRITE

Ce TME se base sur le TD9 et vise à illustrer la modification des tables et l'interaction avec les contraintes d'intégrité. Le schéma Entreprise du TD8 est utilisé. Les instructions de création du shéma et d'insertion de tuples doivent etre récupérées de Moodle comme il sera indiqué.

Ce TME comporte deux parties :

- la première utilise un schémas où les contraintes référentielles empechent, par défaut, de modifer/supprimer des tuples référencés dans d'autres tables, le but étant de controurner ces contraintes pour effectuer la modification/suppression sur les tuples non référencés en rajoutant les conditions nécessaires
- la seconde partie utilise un schéma ou ces contraintes ont été définies avec on delete cascade ce qui permet de supprimer les tuples référencés et entrainer une suppression en cascade des tuples qui les référencent.

PREMIÈRE PARTIE

Exécuter les instructions de création du schéma se trouvat dans TME9-creations-H2.sql

Insertions rejetées

1. Proposer une insertion dans la table Employe qui ne respecte pas la contrainte de clé primaire (pk_emp).

2. Proposer une insertion dans la table Employe qui ne respecte pas la contrainte de limite d'âge.

3. Proposer une insertion dans la table Employe qui ne respecte pas la contrainte de longueur de l'attribut NumSS.

4. Proposer une insertion dans la table Employe qui ne respecte pas la contrainte sur les villes possibles.

Pour la suite des questions, il vous est demandé d'insérer dans Employe les nuplets suivants :

22334, 'Adam', 'Funk', 'Paris', '01-12-1982'

45566, 'Rachid', 'Allaoui', 'Lyon', '13-04-1986'

77889, 'Florent', 'Girac' , 'Marseille', '04-11-1990'

5. Proposer une insertion dans la table Projet qui ne respecte pas la contrainte référentielle vers Employe (fk_resp).

6. Proposer une insertion dans la table Projet qui ne respecte pas la contrainte de ville du responsable d'un projet (resprojet).

A présent, il est vous demandé d'insérer dans Projet les nuplets suivants :

12333, 'ADOOP', 22334, 'Paris', 120000

75777, 'SKALA', 45566, 'Lyon', 180000

89000, 'BAJA', 22334, 'Paris', 24000

7. Peut on insérer des nuplets dans Embauche en l'état ?

Insérer les nuplets suivants dans la table Grille_sal

'Admin', 80000

'Deve', 45000

'Tech', 35000

8. Proposer une insertion dans Embauche qui ne respecte pas au moins une contrainte parmi celles définies pour cette table.

La suite des questions utilise la base dans un état E0 qu'il faudra obtenir en executant les instructions du fichier TME9-insertions-H2.sql (disponible sur Moodle)

Vérifier que que les nombre de nuplets des tables Employe, Projet, Grille_sal et embauche sont : 10, 3, 3, 3, dans cet ordre.

Suppressions rejetées

9**.** Proposer une suppression de nuplets de Employe qui ne respecte pas une contrainte référentielle d'une autre table. Précisez laquelle puis essayer de contourner les contraintes jusqu'à pouvoir supprimer les nuplets de Employe référencés nulle part.

10. Même question pour Projet.

11. Même question pour Grille_sal.

12. Est-ce possible de supprimer les nuplets de Embauche sans rencontrer les mêmes problèmes que précédemment ?

Remettre la base à l'état E0 en executant les instructions du fichier TME9-insertions-H2.sql

Mises à jour rejetées

13. Proposez un mise à jour de Employe qui ne respecte pas les contraintes référentielles puis tenter de trouver une mise-a-jour qui les respecte toutes.

14. Même question avec Projet.

15. Proposez une mise-a-jour de Projet qui ne respecte pas la contrainte resprojet selon laquelle le responsable d'un projet habite la ville du projet dont il est responsable.

16. Proposez une mise-a-jour de profil dans Grille_sal qui ne respecte pas les contraintes référentielles.

Seconde Partie

Pour cette partie, on utilisera un schéma obtenu en executant les instructions du fichier TME9-creations-cascade-H2.sql. Mettez la base à l'état E0 en exécutant les instructions de TME9-insertions.sql.

Comme indiqué précédemment, le but de cette partie d'illustrer un cas où les mises-a-jour ne sont pas rejetées mais plutôt propagées.

Pour le voir, supprimer les nuplets de Employe et vérifier que tous les nuplets des autres tables qui référencent des employes venant d'être supprimés seront supprimés eux aussi.

masquer=1

TME 10 : PL/SQL

Le SGBD ORACLE

Dans ce TME nous allons utiliser le SGBD Oracle. Suivez les instructions du document Oracle avec SQLWorkbench pour vous connecter au serveur Oracle.

Requêtes a resultat unique

Oracle peut exécuter des commandes SQL (SELECT, INSERT, DELETE UPDATE) et des *blocs PL/SQL*

Une suite de commandes contenant un ou plusieurs blocs PL/SQL doit être terminée par un /.

Bloc PL/SQL

Un bloc PL/SQL a la structure suivante (les [..] encadrant les éléments optionnels) :

```
[ DECLARE
  -- déclaration de variables ou d'exceptions ]
 BEGIN
  -- instructions PL/SQL
[ EXCEPTION
  -- actions déclenchées par les évènements répertoriés]
 END;
 /
```

Remarque : Une suite de commandes contenant un ou plusieurs blocs PL/SQL doit être terminée par un / pour être compilé.

Variables

1) Variables SQL*Plus

La commande ACCEPT de SQL*Plus permet de définir une variable et d'afficher un texte avant sa saisie. Cette commande doit figurer à l'extérieur d'un bloc PL/SQL. La valeur de la variable peut ensuite être utilisée partout, en ajoutant un & en préfixe à son nom. Par défaut, la variable est de type CHAR .

2) Variables PL/SQL

Les autres variables utilisées dans un bloc PL/SQL doivent être déclarées dans la section DECLARE de ce bloc. Contrairement aux variables SQL*Plus, elles sont ensuite directement désignées par leur nom, sans préfixe &. Elles peuvent avoir, pour type, n'importe quel type du langage SQL. L'attribut %TYPE permet de faire référence au type d'une colonne donnée.

3) Assignation de variables PL/SQL

On utilise l'opérateur := pour assigner une valeur à une variable, lors de sa déclaration ou dans une instruction de la section BEGIN.

Fonctions de conversion

Les fonctions *to_char* et *to_number* permettent de convertir une valeur numérique en chaîne de caractères, et inversement. On peut également appliquer des *fonctions de conversion* pour modifier les valeurs de variables ; Par exemple les fonctions UPPER et LOWER transforment leur arguments (des chaînes de caractères) en majuscule ou minuscule.

Opérateurs

L'opérateur || concatène deux chaînes de caractères.

Instructions PL/SQL :

Une e*xpression SQL* est une instruction PL/SQL. Par exemple, on peut utiliser l'ordre UPDATE pour modifier la base de données. On suppose que chaque requête SQL retourne au maximum un nuplet. La valeur de cette nuplet peut être copiée dans des variables déclarées dans la section DECLARE. Par exemple :

```
select Jour, Heure, Salle
into JourI, HeureI, SalleI
from TD
where niveau=:new.niveau and UE=:new.UE and NoTD=:new.NoTD;
```

Il est possible d'utiliser les structures de contrôle suivants :

- `if (<condition>) then <bloc_psql1> [ else <bloc_plsql2> ] end if;`
- `loop <bloc_psql> end loop;`
- `for <var> in <seq> loop <bloc_psql> end loop ;`
- `while <condition> loop <bloc_psql> end loop ;`

Les commandes `exit` et `exit when <cond>` permettent de sortir d'une boucle.

Package dbms_output

Ce package fournit la fonction *put_line* qui permet d'afficher un texte à l'écran, si l'option SERVEROUTPUT de *sqlplus* est positionnée. **Remarque** : il n'est pas possible d'utiliser ce package avec SQLWorkbench (nous allons écrire des bloc qui retournent des résultats de requêtes).

Exceptions définies par l'utilisateur

Une telle exception doit être déclarée dans la section DECLARE du bloc PL/SQL concerné. Lorsqu'elle est levée (commande *raise*), *sqlplus* exécute l'instruction prévue dans la section EXCEPTION et met fin à l'exécution du bloc concerné.

Procédures et fonctions stockées

On crée (ou modifie) une procédure par la commande :

```
CREATE OR REPLACE PROCEDURE Nom_Proc
     (Paramètre1 type1, … , Paramètrek typek)
IS
Bloc PL/SQL
/
```

Les types possibles des paramètres sont ceux des variables PL/SQL; on peut notamment utiliser l'attribut %TYPE. Le bloc PL/SQL, constituant le corps de la procédure, peut commencer par une section de déclaration de variables, mais sans le mot réservé DECLARE.

Les fonctions permettent de retourner des résultats. On crée (ou modifie) une fonction par la commande :

```
CREATE OR REPLACE FUNCTION Nom_Fonction RETURN type_resultat
     (Paramètre1 type1, … , Paramètrek typek)
IS
Bloc PL/SQL (contenant les instructions RETURN)
/
```

A l'exécution de cette commande, la procédure est compilée et stockée dans la base. Elle est alors utilisable par différents programmes, qui pourront l'appeler par la commande:

execute *Nom_Proc*(*Valeur1*, …, *Valeurk*) ;

Si la compilation de la procédure s'est terminée sur erreur, on peut lancer la commande SQL*Plus *show errors* pour obtenir des précisions.

Exercices

Les exercices suivants portent sur les tables GAIN, JOUEUR et RENCONTRE de la base TENNIS, implémentée dans la base commune *oracle*. Chaque exercice donne lieu à la création d'une procédure et d'un programme de test de cette procédure. On pourra rassembler les commandes de création des procédures, dans un même fichier, en éditant ce fichier sous Xemacs, et en lançant la création de chaque nouvelle procédure par sélection du code correspondant. Chaque programme de test fera l'objet d'un fichier séparé ; on l'exécutera par la commande *@Nom_de_Fichier*, sous SQL*Plus.

Préparation :

1. *Lancez SQL Workbench dans le repertoire workbench_avec_driver*
2. *Connectez vous à Oracle en suivant les instructions du document Oracle avec SQLWorkbench*
3. *Chargez le script SQL suivant Oracle en tapant la commande suivante dans l'éditeur SQL WorkBench (vous pouvez ouvrir plusieurs onglets):*

```
@tennis
```

1. Créer la fonction moyprime. Copier-coller le code suivant dans l'éditeur :

```
CREATE OR REPLACE FUNCTION moyprime(
  P_lieutournoi GAIN.lieutournoi%TYPE,
  P_annee GAIN.annee%TYPE)
RETURN VARCHAR2 IS
V_moyenne GAIN.prime%TYPE;
E_fin EXCEPTION;
BEGIN
  select AVG(prime) into V_moyenne
  from GAIN
  where lieutournoi=P_lieutournoi and annee=P_annee;
  IF V_moyenne IS NULL THEN RAISE E_fin; END IF;
  RETURN(P_lieutournoi||' '||to_char(P_annee)||': '
                          ||to_char(V_moyenne));
  EXCEPTION
  WHEN E_fin THEN RETURN(P_lieutournoi||' '||to_char(P_annee)
                        ||' non répertorié');
END;
/
```

Exécutez les deux requêtes suivantes et expliquez le résultat (la table dual contient un seul nuplet et permet d'effectuer un appel de fonction à travers une requêtes SQL

```
select distinct moyprime(lieutournoi,annee) as reponse
from GAIN;

select moyprime('Wimbledon',1990) as reponse
from dual;
```

Ajouter un commentaire au code de création de la procédure, pour indiquer ce qu'elle fait (-- *Affiche ...*).

2. Définir une fonction moyprime2 produisant les mêmes effets que la procédure précédente, sans utiliser d'exception E_fin. La tester avec un script moyprime2.sql.

REQUÊTES A RÉSULTAT MULTIPLE

Rappels

Boucle

Une boucle PL/SQL est encadrée par les mots *loop* et *end loop*. Précédée de la clause *for*, elle est exécutée un certain nombre de fois ; précédée de la clause *while*, elle est exécutée tant qu'une condition reste satisfaite ; en l'absence de clause *for* ou *while*, elle s'arrête à la rencontre d'une instruction *exit*.

On peut placer une boucle à l'intérieur d'une autre boucle. Il est alors préférable de nommer chaque boucle, en plaçant une étiquette juste avant le début de la boucle (<<*nom_boucle*>> loop …. exit *nom_boucle* when… …. end *nom_boucle* ;).

Curseur

Lorsqu'une requête est susceptible de produire plusieurs lignes en résultat, on utilise un curseur pour accéder à chacune de ces lignes. Le curseur doit être défini dans la section DECLARE du bloc où il est utilisé. L'instruction *open* positionne le curseur sur la première ligne du résultat. L'instruction *fetch* permet de recopier les valeurs des attributs de la ligne courante dans des variables, et positionne le curseur sur la ligne suivante. L'instruction *close* libère les ressources allouées au curseur.

Un curseur possède des attributs :

%FOUND a la valeur vrai si une instruction *fetch* a effectivement pu lire une ligne ;

%NOTFOUND a la valeur vrai dans le cas contraire ;

%ROWCOUNT comptabilise le nombre de lignes effectivement lues par une série d'instructions *fetch*.

Un curseur peut être défini avec des paramètres dont la valeur est fixée lors de l'ouverture:

cursor *nom_curseur* (*param1 type1*, *param2 type2,* ...) is … ;
open *nom_curseur* (*valeur1*, *valeur2*, …) ;

Fonctions de manipulation de chaînes de caractères

La fonction *length* retourne le nombre de caractères d'une chaîne.
La fonction *rpad* répète un motif donné, à la droite d'une chaîne, pour obtenir une chaîne de longueur donnée; la fonction *lpad* répète le motif à gauche de la chaîne.

Exercices

3. Expérimenter la fonction et la requête de test suivants ; puis ajouter un commentaire indiquant ce que fait la procédure.

```
CREATE or REPLACE FUNCTION maxprime
```

```
( P_Annee1 GAIN.annee%TYPE, P_Annee2 GAIN.annee%TYPE )
RETURN CLOB
IS
V_Nom Joueur.nom%type;
V_MaxPrime Gain.Prime%type;
V_reponse CLOB;
cursor C_MaxPrime is
select nom, max(prime)
from GAIN, JOUEUR
where annee between P_Annee1 and P_Annee2
  and gain.NuJoueur=joueur.NuJoueur
group by nom;
BEGIN
open C_MaxPrime;
V_reponse:='';
loop
  fetch C_MaxPrime into V_Nom, V_MaxPrime;
  exit when C_MaxPrime%NOTFOUND;
  if C_Maxprime%ROWCOUNT=1 then
    V_reponse:=V_reponse||'Plus forte prime gagnée entre '
        ||P_Annee1||' et '||P_Annee2||':';
  end if;
  V_reponse:=V_reponse||chr(10)
        ||rpad(V_Nom,14,'.')||lpad(to_char(V_MaxPrime),8,'.')||' F';
end loop;
if C_MaxPrime%ROWCOUNT=0
 then V_reponse:='Aucun tournoi n''est répertorié entre ces dates';
end if;
close C_MaxPrime;
return V_reponse;
END;
/

-- @WbOptimizeRowHeight lines=100
select maxprime(1990,1994) as réponse from dual;

select maxprime(1970,1975) as reponse from dual;
```

4. Modifier la procédure précédente de façon à obtenir une fonction maxprime2 admettant en paramètre le nom d'un sponsor, et qui :
   - détermine la première et la dernière année où ce sponsor a figuré dans un tournoi,
   – puis affiche, pour chaque joueur, la plus forte prime qu'il a touchée durant cette période (les joueurs pris en compte sont ceux qui ont participé à, au moins, un tournoi dans la période considérée, qu'ils aient, ou non, été sponsorisés par le sponsor indiqué). La donnée d'un sponsor inconnu dans la base provoque l'arrêt immédiat du programme, par levée d'une exception.

```
-- @WbOptimizeRowHeight lines=100
select maxprime2('Peugeot') as réponse from dual;

RÉPONSE
Plus forte prime donnée par Peugeot
entre 1989 et 1994:
FORGET..........400000 F
SAMPRAS........1400000 F
LECONTE.........350000 F
FLEURIAN........600000 F
```

```
-- @WbOptimizeRowHeight lines=100
select maxprime2('Reebok') as réponse from dual;


RÉPONSE
Plus forte prime donnée par Reebok entre
1993 et 1994:
PIERCE.........350000 F
FORGET.........600000 F
SAMPRAS........1800000 F
LARSSON.........800000 F
GRAF............400000 F
LECONTE........1000000 F
```

5. Ecrire une fonction *jouspon*, qui affiche les noms des joueurs qui ont été sponsorisés par le sponsor dont le nom est transmis en paramètre. Tester la procédure avec un script JouSpon.sql dont l'exécution donnera, par exemple :

```
-- @WbOptimizeRowHeight lines=100
select jouspon('Reebok') as réponse from dual;


RÉPONSE
Joueurs sponsorisés
par Reebok:
PIERCE
FORGET
SAMPRAS
LARSSON
GRAF
LECONTE


-- @WbOptimizeRowHeight lines=100
select jouspon('Peugeot') as réponse from dual;


RÉPONSE
Joueurs sponsorisés
par Peugeot:
FORGET
SAMPRAS
LECONTE
FLEURIAN



-- @WbOptimizeRowHeight lines=100
select jouspon('Head') as réponse from dual;


RÉPONSE
Head n'a sponsorisé
```

```
personne
```

6. Compléter la procédure précédente, de façon à ce qu'elle indique, pour chaque joueur figurant en résultat, les tournois qu'il a gagnés (avec le sponsor donné, ou un autre) ; les joueurs n'ayant gagné aucun tournoi apparaîtront, dans la liste, avec leur seul nom, comme auparavant. On obtiendra ainsi, par exemple :

```
-- @WbOptimizeRowHeight lines=100
select jouspon2('Reebok') as réponse from dual;

RÉPONSE
Joueurs sponsorisés par Reebok:
GRAF vainqueur de Wimbledon et de Wimbledon
1992
LARSSON n'a pas gagné de tournoi
LECONTE n'a pas gagné de tournoi
PIERCE n'a pas gagné de tournoi
FORGET n'a pas gagné de tournoi
SAMPRAS vainqueur de Roland Garros et de
Wimbledon 1993


-- @WbOptimizeRowHeight lines=100
select jouspon2('Peugeot') as réponse from dual;

RÉPONSE
Joueurs sponsorisés par Peugeot:
LECONTE n'a pas gagné de tournoi
FORGET n'a pas gagné de tournoi
FLEURIAN n'a pas gagné de tournoi
SAMPRAS vainqueur de Roland Garros et de
Wimbledon 1993



-- @WbOptimizeRowHeight lines=100
select jouspon2('Head') as réponse from dual;

RÉPONSE
Head n'a sponsorisé
personne
```

masquer=1

TME 11 : TRIGGERS

PL/SQL

Sous Oracle, le bloc d'instructions d'un trigger est un bloc PL/SQL. Dans ce TME nous allons utiliser un sous-ensemble minimal de PL/SQL pour définir et modifier des variables et exécuter des ordres SQL (rappels sur la syntaxe PL/SQL dans le TME 10).

Un trigger peut détecter une situation où il faut annuler une transaction (une transaction est une séquence de lectures et de mises-à-jour qui sont exécutées ou annulées ensemble). On ne peut pas annuler une transaction à l'intérieur d'un trigger mais on peut déclencher une exception en utilisant la fonction `RAISE_APPLICATION_ERROR` (qui peut ensuite être traitée par la procédure qui a déclenchée le trigger) :

```
RAISE_APPLICATION_ERROR(code,'message') ;
```

Le code d'une exception utilisateur doit être compris entre -20999 et 20000.

Limitations : Un trigger AFTER déchenché après un update (ou un insert) de la table T n'est pas autorisé à modifier la table T (ne pas écrire d'instruction update T dans le corps du trigger). Pour palier cette limitation, utiliser un trigger BEFORE.

SCHÉMA « INSCRIPTIONS »

On considère une base de données avec des informations sur les enseignements, les enseignants et les étudiants. Elle comprend deux relations suivantes que vous devez créer :

```
CREATE TABLE TD (
    noTD smallint not null,
    codeUE varchar(10),
    niveau varchar(10),
    salle varchar(10),
    jour varchar(8) not null,
    heure varchar(5) not null,
    noEns smallint,
    PRIMARY KEY ( NoTD, codeUE ),
    UNIQUE (salle,jour,heure)) ;

CREATE TABLE  INSCRIPTION (
    noEtud smallint,
    noTD smallint,
    codeUE varchar(10),
    PRIMARY KEY (noEtud, codeUE)) ;
```

La clé primaire de relation TD est (noTD, codeUE) et celle de la relation INSCRIPTION est (noEtud, codeUE). Chaque n-uplet de la relation TD renseigne sur un TD d'une UE donnée, pour un niveau donné. (ex. le TD 2 de l'UE BD2 du niveau L3). Les TD ont lieu au rythme d'une fois par semaine, pendant toute l'année universitaire. Les attributs SALLE, JOUR et HEURE donnent le lieu, le jour et l'horaire de début du TD. On suppose que tous les TDs de toutes les UEs sont synchronisés au niveau des horaires et de la durée (il n'y a pas de conflit possible entre deux TDs qui ne commencent pas à la même heure et n'ont pas lieu dans la même salle). L'enseignant qui assure le TD est identifié par NOENS. Les étudiants s'inscrivent à chaque UE, séparément, et choisissent un seul TD pour cette UE. L'étudiant est identifié par NOETUD.

EXERCICES

Pour chacune des questions suivantes, vérifier que le trigger est créé sans erreurs et proposer des insertions/suppressions/mises à jour pour vérifier qu'il fonctionne correctement.

1. Créer deux triggers qui assurent que les valeurs des attributs NIVEAU et CODEUE, entrées dans la table TD et INSCRIPTION, soient en majuscules, quelle que soit la casse utilisée dans les instructions d'insertion ou de mise à jour (utilisez la fonction UPPER). Vérifiez que les triggers fonctionnent correctement.

2. Créez un trigger qui empêche qu'un étudiant s'inscrit dans plus que 6 UEs. érifiez que le trigger fonctionne correctement.
3. Créez un trigger qui empêche que le nombre de groupes de TD par UE dépasse 4. Vérifiez que le trigger fonctionne correctement.
4. Créez un trigger qui empêche qu'un étudiant suive deux TDs enseignés par le même enseignant. Vérifiez que le trigger fonctionne correctement.

5. Pour empêcher un étudiant de s'inscrire à des TD incompatibles (i.e. ayant lieu en même temps), on introduit une nouvelle contrainte d'intégrité dans la base : un étudiant ne peut pas avoir deux inscriptions qui lui imposent d'être, au même moment, à deux endroits différents.

   Écrire un trigger qui assure cette contrainte, de la façon suivante :

   - en préambule à chaque inscription, le trigger inscrit le numéro de l'étudiant, avec le jour, l'heure et la salle du TD qu'il a choisi, dans *une table supplémentaire* LOCETUD que vous devez créer avec une contrainte d'intégrité qui assure l'unicité du triplet (NoETUDIANT, JOUR, HEURE) :

```
CREATE TABLE LOCETUD (
     NoEtud smallint,
     Jour varchar(8),
     Heure varchar(5),
     Salle varchar(10),
     PRIMARY KEY ( NoEtud, Jour, Heure ));
```

   - si l'étudiant a déjà pris une inscription correspondant au même jour et à la même heure, la contrainte de table précédente provoque le rejet du nouveau quadruplet (NoETUDIANT, JOUR, HEURE, SALLE), le trigger est arrêté sur erreur, et l'instruction d'insertion d'un nouvel élément dans la table INSCRIPTION est, par suite, arrêtée sur erreur.

6. (optionnel) Trouvez une solution pour la question précédente qui évite d'introduire la table supplémentaire LOCETUD. Conseil : il faut faire une jointure entre les tables TD et INSCRIPTION.