M2106 – TD n°0 – Révisions

A. Algorithmique et programmation

Exercice n°1 – Instruction conditionnelle

Dans chaque cas proposé, donner les valeurs finales des variables de type entier **x** *et* **y** *:*

a)
```
01 begin
02   x:=0; y:=5;
03   if x<y then x:=x+y;
04   elsif x=y then x:=2*x;
05   else x:=x*y;
06   end if;
07 end;
```
x =
y =

b)
```
01 begin
02   x:=5; y:=5;
03   if x<y then x:=x+y;
04   elsif x=y then x:=2*x;
05   else x:=x*y;
06   end if;
07 end;
```
x =
y =

c)
```
01 begin
02   x:=10; y:=5;
03   if x<y then x:=x+y;
04   elsif x=y then x:=2*x;
05   else x:=x*y;
06   end if;
07 end;
```
x =
y =

d)
```
01 begin
02   x:=0; y:=5;
03   if x<y then x:=x+y; end if;
04   if x=y then x:=2*x; end if;
05   if x>y then x:=x*y; end if;
06 end;
```
x =
y =

e)
```
01 begin
02   x:=0; y:=0; c :='a' ; -- où c a été déclaré comme caractère
03   if c='c' then x:=x+10; end if;
04   if c='a' or c='b' then x:=x+100; end if;
05   if not (c='a') then y:=y+10; end if;
06 end;
```
x =
y =

f)
```
01 begin
02   x:=10; y:=5;
03   if x<y+1 then x:=x+y;
04   elsif 2*x=y then x:=2*x;
05   elsif x+y=0 then x:=x+10;
06   end if;
07 end;
```
x =
y =

g)
```
01 begin
02   x:=10; y:=5;
03   if x<=y then
04     x:=x+y;
05     else
06       y:=2*y;
07       if x<=y then
08         x:=2*x;
09       end if;
10   end if;
11 end;
```
x =
y =

Exercice n°2 – Procédures et fonctions

a) Conversion

Soit la fonction **`c2f()`** *suivante :*

```
01 function c2f(tcel : in float) return float is
02 -- {} => {résultat = température en degrés Fahrenheit équivalente à
03                                  la température en degrés Celsius, tcel}
04   tfah : float ;
05 begin
06   tfah := tcel*9.0/5.0 + 32.0 ;
07   return tfah;
08 end c2f;
```

Écrire la fonction inverse **`f2c()`**, *qui convertit une température fahrenheit en degrés.*

b) Calcul de prix

Les différents tarifs pour une place de cinéma sont décrits dans le tableau ci-dessous :

Age	*Jusqu'à 17 ans*	*A partir de 18 ans*	*Lunettes 3D*
Prix normal	*8€*	*10€*	*+1,50€*
Prix abonné	*7€*	*8€*	*+1€*

Écrire une fonction **`Tarif()`** *qui calcule le prix d'une place de cinéma pour une personne en fonction de 3 paramètres (l'âge, la possession d'un abonnement, la location de lunettes 3D) :*

```
function Tarif (age : in integer; abo, lun : in boolean) return float is
-- {} => {résultat = prix de la place}
```

c) Permutation

Ecrire une procédure **`permut3()`** *qui effectue la permutation circulaire de trois entiers passés en paramètres.*

Exercice n°3 – Compréhension et simplification d'algorithmes

a) Itérations

Corriger la procédure suivante permettant de calculer le factoriel d'un entier donné en paramètre.

```
01 procedure factoriel (n: out integer; res: out integer) is
02   i : integer; -- variable d'itération
03 begin
04   i:=0;
05   while (i <= n) loop
06     res := res * i;
07     i := i+1;
08   end loop ;
09 end factoriel;
```

b) Jeu de Nim

Expliquer ce que fait la procédure suivante et la simplifier pour :

- *Ne plus utiliser* **`FinJeu`** *;*
- *Déplacer le code du bloc* **`else`** *(l. 22 à 30) dans une procédure ou une fonction.*

```
01 with p_esiut; use p_esiut;
02 procedure nim is
03   Tas    : integer; -- nb allumettes restantes
04   MAX    : constant integer := 3;
05   N      : integer; -- nb allumettes prises
06   J      : integer; -- numéro du joueur courant
07   FinJeu  : boolean; -- contrôle de fin du jeu
08 begin
09   -- début du jeu : initialisations
10   FinJeu := false;
11   J := 1;
12   Ecrire("Joueur" & image(J) & ", choisissez un nombre initial d'allumettes : ");
13   Lire(Tas);
```

```
14   while not FinJeu loop
15     J := J mod 2 +1;
16     Ecrire("Reste " & image(Tas)& "    ");
17     if Tas = 0 then
18       J := J mod 2 + 1;
19       Ecrire("BRAVO Joueur" & image(J) & ", vous avez GAGNE!!!");
20       FinJeu := true;
21     else
22       loop
23         Ecrire("Joueur" & image(J) & ", vous prenez ? ");
24         Lire(N);
25         if N > MAX or N > Tas then
26           Ecrire("Impossible de prendre autant d'allumettes !");
27         end if;
28       exit when (N <= MAX) and (N <= Tas);
29       end loop;
30       Tas := Tas-N;
31     end if;
32   end loop;
33 end nim;
```

Exercice n°4 – Exceptions

Compléter la trace d'exécution du programme suivant :

```
01 procedure essai is
02
03   EX : exception;
04
05   procedure ProcA is
06   begin
07     Ecrire_Ligne("Procédure A : EX va être déclenchée");
08     raise EX;
09     Ecrire_Ligne("Procédure A : fin de la procédure") ;
10   exception
11     when EX => Ecrire_Ligne("Procédure A : EX traitée localement");
12   end ProcA;
13
14   procedure ProcB is
15   begin
16     Ecrire_Ligne("Procédure B : EX va être déclenchée");
17     raise EX;
18     Ecrire_Ligne("Procédure B : fin de la procédure") ;
19   end ProcB;
20
21   procedure ProcC is
22   begin
23     Ecrire_Ligne("Procédure C : EX va être déclenchée");
24     raise EX;
25     Ecrire_Ligne("Procédure C : fin de la procédure") ;
26   exception
27     when EX =>
28       Ecrire_Ligne("Procédure C:EX traitée localement puis propagée");
29       raise;
30   end ProcC;
31
32 begin
33   ProcA;
34   begin
35     ProcB;
36     Ecrire_Ligne("Procédure essai : fin du bloc");
37   exception
38     when EX => Ecrire_Ligne("Procédure essai : EX traitée dans le bloc");
39   end;
40   ProcC;
41   Ecrire_Ligne("Procedure essai : fin du programme");
42
43 exception
44   when EX => Ecrire_Ligne("Procédure essai : EX traitée globalement");
45 end essai;
```

B. SQL

Nous rappelons le schéma relationnel de la base de données vue au semestre précédent, et utilisée par le directeur d'un zoo pour faciliter la gestion de celui-ci.

Animal (noma, especе, sexe, pays, datea, #numc)
*Pour chaque animal, il désire connaître son **nom**, à quelle **espèce** il appartient, son **sexe**, son **pays d'origine**, sa **date de naissance**, et le **numéro de la cage** dans laquelle il se trouve.*

Estmalade (#noma, nommal, datem, remede)
*De plus, les diverses maladies contractées par les animaux depuis leur arrivée au zoo sont enregistrées. Sont conservés, le **nom de l'animal**, le **nom de la maladie contractée**, la **date** à laquelle la maladie a été diagnostiquée et le **remède prescrit**.*

Cage (#numc, lieu, typec)
*Chaque cage est **identifiée par un numéro**, est située dans une **région du zoo**, et son **type** la rend compatible avec certaines espèces animales.*

Gardien (numg, nomg, adresse, dateg)
*Le directeur gère également l'embauche des gardiens dont il souhaite conserver le **nom**, **l'adresse** et **la date de naissance**. Pour éviter les problèmes d'homonymie, chaque gardien **est identifié en base par un numéro**.*

Soccupe (#numg, #numc)
Enfin, un gardien peut avoir à s'occuper de plusieurs cages et à l'inverse, certaines cages nécessitent la présence de plusieurs gardiens pour des raisons de sécurité.

Ecrire les requêtes permettant :

1. D'afficher tous les cas de typhus, dans l'ordre anti-chronologique. Il n'est pas demandé d'afficher le remède prescrit.

2. D'afficher le numéro des cages vides.

3. D'afficher, pour chacune des cages non vides de type `'fauves'`, le nombre d'animaux qu'elle contient. Les cages seront affichées de la moins remplie à la plus remplie.

4. Même question mais en ne considérant que les cages contenant au moins deux animaux.

5. D'insérer le lion écossais gilderoy, né le 15/06/1965, dans la cage 5.

6. D'enregistrer le fait que tous les animaux de la cage 5 ont contracté aujourd'hui le typhus, mais sans remède.

7. Que tous les animaux d'origine inconnue (`pays='inconnu'`), deviennent français.

8. D'effacer tous les animaux de la cage 5.

M2106 – TD n°1 – Procédures

Pour ce TD nous utilisons à nouveau la base de données utilisée par le directeur d'un zoo, dont le schéma relationnel a été rappelé au TD précédent. Il s'agit d'écrire des procédures stockées pour faciliter la manipulation de la base par le directeur.

1. Ecrire une fonction `DernierCas` qui, étant donné le nom d'une maladie, retourne la date à laquelle cette maladie a été contractée par un animal du zoo pour la dernière fois.
2. Appeler la fonction précédente pour le `'typhus'`.
3. Modifier la fonction pour lever une exception si la maladie n'a encore jamais été contractée par aucun animal du zoo.
4. Ecrire une fonction `DerniersCas` qui, étant donné le nom d'une maladie, retourne la liste des animaux ayant déjà contracté cette maladie, de la date la plus récente à la plus ancienne. Chacun animal sera affiché autant de fois qu'il a contracté la maladie.
   BONUS : Modifier la fonction pour que chaque date soit ajoutée au nom de l'animal.
5. Ecrire une fonction `CageDispo` qui, étant donné un type de cage, renvoie le numéro d'une cage de ce type contenant le moins d'animal. On supposera que toute cage contient au moins un animal.
   BONUS : modifier la fonction pour traiter la possibilité qu'il y ait une cage vide ; c'est alors son numéro qui devait être retourné par la fonction.
6. Ecrire une fonction `InsererLionFauves` qui, étant donné le nom, le sexe, le pays d'origine et la date de naissance d'un nouveau lion, l'enregistre dans la table `animal` avec le numéro de la cage de type `'fauves'` la moins remplie.
   Indication : utiliser la fonction `CageDispo`.

Questions supplémentaires :

7. Ecrire une fonction `InsererLionFauvesAdv` qui fasse le même travail que `InsererLionFauves` mais ajoute un numéro au nom du lion si un autre animal possède le même nom. Par exemple, s'il existe déjà un animal nommé `gilderoy`, un nouveau lion de même nom sera enregistré avec le nom `gilderoy2`, le suivant avec le nom `gilderoy3` etc.
8. Ecrire une fonction `RemedesUtilises` qui, pour un nom de maladie donné, renvoie le remède le plus utilisé pour la soigner. Si plusieurs remèdes satisfont cette propriété, alors il faut tous les afficher.

Types utilisés :

```
numg, numc                          numeric(2)
nomg, espece, pays, remede          varchar(20)
noma, nommal, adresse, lieu, type   varchar(10)
dateg, datea, datem                 date
sexe                                char(1)
```

M2106 – TD n°2 – Curseurs

Exercice d'illustration de cours :

Pour cet exercice, nous travaillerons sur un sous-ensemble de la base Cinéma :

Salle (nomcine, numsalle, prix, titre) Une salle projette un film à un certain tarif.

Nous souhaitons écrire une fonction **`MajPrix()`** sans paramètres ni valeur de retour, qui augmente le prix des places de manière proportionnelle au nombre de salles du cinéma : le prix est multiplié par **`(1.0 + nbs/10.0)`** où **`nbs`** est le nombre de salles du cinéma.

1. Ecrire une telle fonction avec un curseur déclaré et ouvert explicitement ;
2. Ecrire une telle fonction avec un curseur déclaré explicitement mais ouvert implicitement ;
3. Ecrire une telle fonction avec un curseur non explicitement déclaré.

Exercice de TD : Tour de France

Les organisateurs du Tour de France ont créé une base de données pour la gestion des coureurs et des étapes. Le schéma relationnel est le suivant :

EQUIPE (CodeEquipe, NomEquipe, DirecteurSportif)
COUREUR (NuméroCoureur, NomCoureur, CodeEquipe, NomPays)
ETAPE (NuméroEtape, DateEtape, VilleDép, VilleArr, NbKm, TypeEtape)
PARTICIPATION (NuméroCoureur, NuméroEtape, TempsRéalisé)
TRANSFERT (VilleSoir, VilleMatin, DateTransfert)

Ecrivez, en utilisant des curseurs, les procédures stockées suivantes.

1. Ecrire une procédure sans paramètres **`Programme()`** qui affiche toutes les étapes sous la forme :

```
NOTICE : 2014-07-05 : Etape 1 de type plaine
NOTICE :          Leeds > Harrogate (191 km)
NOTICE : 2014-07-06 : Etape 2 de type plaine
NOTICE :          York > Sheffield (198 km)
```

2. Ecrire une procédure sans paramètres **`CreerTransfert()`** qui complète la relation TRANSFERT (initialement vide) : à chaque fois qu'une ville d'arrivée n'est pas la ville de départ de l'étape suivante, un n-uplet est inséré avec les détails du transfert : noms des 2 villes et date du soir du transfert.
3. Ecrire une procédure **`ResultatsPartiels()`** qui étant donnée une date (passée), renvoie pour chaque étape ayant déjà eu lieu à cette date, la date et le type de l'étape ainsi que les noms des 3 meilleurs coureurs avec leur classement. *Indications : on poura définir la signature d'une telle fonction comme suit :*

```
create or replace function Resultatspartiels(inout datee date, out typee varchar(20),
                                              out rang integer, out nomc varchar(20))
                  returns setof record
```

chaque n-uplet sera renvoyé avec la commande : **`return next ;`**

Question supplémentaire :

4. En cas de non respect du règlement par un coureur, tous les coureurs de son équipe peuvent se voir infliger une pénalité à une étape. Ecrire une procédure **`Penalite()`** qui, étant données une étape et une équipe, rajoute à chacun des coureurs de l'équipe la durée totale de ses étapes depuis le début du Tour divisée par 10.

Types utilisés :

`varchar(20)` pour les chaînes, **`numeric(3)`** pour les numéros, **`interval`** pour les temps réalisés.

M2106 – TD n°3 – Triggers

A. Triggers sur tables

Exercice d'illustration de cours :

Pour cet exercice, nous travaillerons sur un sous-ensemble de la base Zoo :

Animal (noma, espece, sexe, pays, datea, numc) l'animal *noma* né le *datea* est dans la cage *numc*

Pour garder une trace des opérations effectuées par les utilisateurs de la base sur cette table, nous créons la table suivante :

```
CREATE TABLE animal_log(
  modifie_par text      DEFAULT current_user,
  modifie_a   timestamp DEFAULT now(),
  operation   text);
```

En pratique, plusieurs n-uplets peuvent être l'objet d'une même instruction à une même date ; en revanche, un utilisateur ne peut effectuer qu'une seule instruction à une date de type `timestamp` *donnée.*

1. Nous souhaitons enregistrer dans `animal_log`, chaque insertion (`INSERT`) dans `animal`. Quel effet l'exécution de la requête suivante par l'utilisateur *cruella,* doit-elle avoir sur `animal_log` ?

```
INSERT INTO animal VALUES ('Toulouse', 'chat', 'm', 'France', current_date-10, 1),
                          ('Berlioz', 'chat', 'm', 'France', current_date-20, 1);
```

2. Ecrire un trigger et sa fonction associée pour réaliser ces enregistrements.

3. Nous décidons d'enregistrer chaque instruction plutôt que chaque insertion, et ajoutons la clé primaire :
```
ALTER TABLE animal_log ADD PRIMARY KEY (modifie_par, modifie_a) ;
```
Modifier le trigger défini à la question 1 pour qu'il soit compatible avec cette nouvelle contrainte.

4. Si l'on remplaçait l'instruction `RETURN new;` par `RETURN NULL;` dans la fonction définie précédemment, quelles en seraient les conséquences lorsque :

   a. la fonction est déclenchée par le trigger évoqué dans la question précédente?

   b. la fonction est déclenchée par le trigger défini à la question 1 (sans clé primaire pour `animal_log`)?

5. Dans un autre scénario, la table de log sert plutôt au suivi des placements des chats dans les cages. La création de la table donnée en début d'énoncé est alors suivie de :

```
ALTER TABLE animal_log
   ADD noma varchar(10),
   ADD ancienne_cage numeric(2),
   ADD nouvelle_cage numeric(2),
   ADD num_mv serial PRIMARY KEY;
```

   Ecrire fonction et trigger adaptés à ce nouveau scénario.

6. Modifier fonction et trigger pour que non seulement les insertions d'animaux soient enregistrées, mais également les suppressions d'animaux et les changements de cage (pour ce dernier point, on supposera que la clé `noma` n'est jamais modifiée).

Questions supplémentaires :

Nous considérons deux autres tables de la base Zoo :

Estmalade (noma, nommal, datem, remede) l'animal *noma* est atteint de *nommal* depuis *datem* et a reçu tel *remede.*

Soccupe (numg, numc) le *numg*-ème gardien s'occupe de la *numc*-ème cage.

7. Un chat malade de 'H5N1' doit avoir reçu un remède, ou être placé à l'infirmerie (`remede='infirmerie'`).

8. Si un gardien ne s'occupe d'aucune cage, une notice doit être levée.

B. Triggers sur vues

Exercice d'illustration de cours :

9. Créer une vue `NouveauxNes` donnant les informations sur les animaux nés aujourd'hui, à l'exception de leur date de naissance.

10. Créer un trigger et sa procédure permettant d'insérer un nouveau-né dans la vue `NouveauxNes` en exécutant simplement une commande telle que
    ```
    insert into NouveauxNes(noma) values ('Berlioz') ;
    ```
    Quel n-uplet sera inséré ? Quel n-uplet aurait été inséré en l'absence du trigger ?

Questions de TD :

11. Nous supposons qu'un des gardiens a pour nom votre login postgresql. Créer une vue `MesAnimaux` qui regroupe les animaux dont vous avez la charge.

12. Créer des fonctions et un trigger qui permettent l'insertion et la mise à jour de cette vue.

M2106 – TD n°4 – Utilisateurs et droits

Exercice d'illustration de cours :

Pour cet exercice, nous travaillerons sur la base Quête :

Chevalier (nom, pays_naissance)
Graal (id, authentique, decouvreur)

Répondre aux questions suivantes en proposant les commandes les plus courtes possibles :

1. Créer les users *invite*, *arthur* et *lancelot* (la méthode par défaut est `md5`).
2. Créer le groupe *admin* qui doit contenir *arthur* et *lancelot*.
3. Faire en sorte que les 3 utilisateurs créés à la question 1 soient les seuls (à l'exception des *superusers*) à pouvoir se connecter à la base *quete*.
4. Donner le droit à tout le monde de pouvoir consulter la table *graal*.
5. Donner aux membres du groupe *admin* tous les droits sur les deux tables sauf celui de supprimer des n-uplets et celui de modifier l'identifiant de chaque n-uplet de la table.
6. Donner à *invite* le droit de consulter les n-uplets de *chevalier* correspondant aux chevaliers ayant découvert au moins un graal (Indice : utiliser une vue que vous pourrez nommer *heros*).

Exercice : Offre d'emploi

Le schéma relationnel suivant permet à Pôle Emploi de gérer les offres d'emploi proposées par des sociétés ainsi que les demandeurs d'emploi qui répondent à ces offres.

Offre (numOffre, dateOffre, secteur, expérience, qualification, salaire, société, disponible)
*Une offre est déterminée par un **numéro,** et caractérisé par une **date** de début, un **secteur** d'activité concerné, une **expérience** professionnelle exigée, une **qualification** professionnelle, un **salaire** proposé et le nom de la **sociéte** qui la propose. Enfin, un booléen indique si cette offre est toujours **disponible**.*

Demandeur (ninsee, nom, prénom, dateNaissance, adresse, qualification, expérience, conseiller)
*Un demandeur d'empoi est identifié par son numéro de sécurité sociale **ninsee**, et est caractérisé par ses **nom**, **prénom**, **date de naissance**, **adresse**, **qualification** et **expérience** professionnelles. Enfin, une fois inscrit, il est suivi par un **conseiller** Pôle Emploi (avant inscription, ce champ vaut `NULL`).*

Candidature (ninsee, numCand, numOffre, étape, dateEtape)
*Pour chaque demandeur **ninsee**, on enregistre les différentes candidatures qu'il a faites, ordonnées chronologiquement (**numCand** vaut 1 pour la première candidature puis 2 ...). On conserve pour chaque candidature l'identifiant de **l'offre** à laquelle le demandeur postule, l'**étape** du recrutement (demandeRV, RVfixé, RVpassé, sélectionné, refusé) et la date **dateEtape** à laquelle a été enregistrée cette étape. Chaque demandeur ne pouvant candidater qu'une fois à une offre donnée, le couple (ninsee, numOffre) est une autre clé candidate pour cette relation (contrainte `UNIQUE`).*

Lors de l'étude organisationnelle, on a identifié les règles organisationnelles suivantes :

Un demandeur d'emploi avant inscription : Lorsqu'il s'inscrit, c'est le demandeur lui-même qui enregistre son numéro de sécurité sociale, son nom et son prénom, sa date de naissance et son adresse. Il ne peut ni consulter ni modifier les données de la base.

Un demandeur d'emploi inscrit : Une fois inscrit, il reçoit par courrier une première convocation avec le conseiller Pôle Emploi qui va le suivre. Lors de cet entretien, il reçoit son identifiant (son numéro de sécurité social) et son mot de passe et il peut alors interagir avec la base. Il peut consulter ses informations mais il ne peut pas les modifier lui-même. Il a accès en lecture aux offres d'emploi qui correspondent à son profil (même qualification, et expérience demandée inférieure ou égale à la sienne).
Il a accès à ses candidatures. C'est lui qui enregistre toute nouvelle demande d'entretien pour une offre qui lui a été proposée. Une fois sa demande enregistrée, il ne peut la retirer. Par contre, c'est lui qui met à jour les informations concernant les étapes de sa recherche d'emploi.

Un conseiller Pôle Emploi : C'est lui qui enregistre les informations des offres d'emploi déposées. Il peut consulter et modifier ces informations (en particulier, lorsqu'une offre n'est plus disponible, il passe l'attribut *disponible* à `false`) mais pas les supprimer.
C'est également lui qui termine l'inscription des nouveaux demandeurs d'emploi (il affecte son propre login au champ *conseiller* du n-uplet correspondant) et qui enregistre leur qualification et expérience professionnelle lors du premier entretien.
Pour simplifier, il peut consulter et modifier les informations concernant tous les demandeurs et il peut consulter leurs demarches de recherche d'emploi (qu'il suive ou non le demandeur en question).

Ecrire des commandes SQL pour répondre aux questions suivantes :

1. Créer un utilisateur générique *invite* avec le mot de passe *invite* pour la pré-inscription. Créer ensuite deux rôles *inscrits* et *conseillers* qui vont contenir les utilisateurs correspondant respectivement à un demandeur d'emploi inscrit et à un conseiller Pôle Emploi.

2. Définir les droits de l'utilisateur *invite.*

3. Définir les droits du rôle conseillers, puis accorder ces droits à l'utilisateur *joseph*. Créer un nouvel utilisateur *henry* de telle sorte qu'il possède immédiatement les droits accordés aux conseillers.

4. On exécute la commande **`REVOKE select ON offre FROM public;`**. Les conseillers *joseph* et *henry* peuvent-ils encore consulter les offres?

5. Définir les droits des demandeurs d'emploi inscrits.