Direction Générale
des Etudes Technologiques
********** DGET *********

Institut Supérieur des Etudes Technologiques du Kef

Support de TP « SGBD »

Réalisé par :

BOUKCHIM Mosaab
(AT-Iset de Kef)
HOSNI Anis
(AT-Iset de Kef)

Niveau : Quatrième niveau en Informatique
Option : Informatique de gestion, Réseaux informatique
Version : 1.0

Plan

TP N 1 (MCD MERISE) 2
TP N 2 (SQL) 5
TP N 3 (SQL) 9
TP N 4 (SQL) 12
TP N 5 (SQL) 13
TP N 1 PL/Sql 14
TP N 2 PL/SQL. 20
TP N 3 PL/SQL 21
TP N 4 PL/SQL 25

TP N 1 (MCD MERISE)

Tester vos connaissances

Exercice 1 : Centre médical

On vous donne un MCD (Modèle Conceptuel de Données) représentant des visites dans un centre médical. Répondez aux questions suivantes **:**

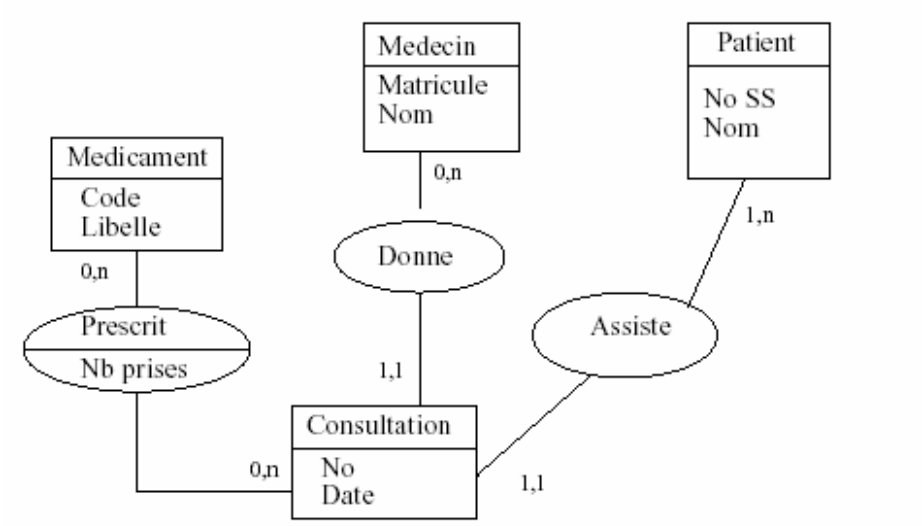


FIG. 1.1 – *Centre médical*

Q1 : Un patient peut-il effectuer plusieurs visites ?

Q2 : Un médecin peut-il recevoir plusieurs patients dans la même consultation ?

Q3 : Peut-on prescrire plusieurs médicaments dans une même consultation ?

Q4 : Deux médecins différents peuvent-ils prescrire le même médicament ?

Q5 : Donner le schéma relationnel pour ce MCD.

Exercice 2 : Tournoi de tennis

Le second MCD représente des rencontres dans un tournoi de tennis.

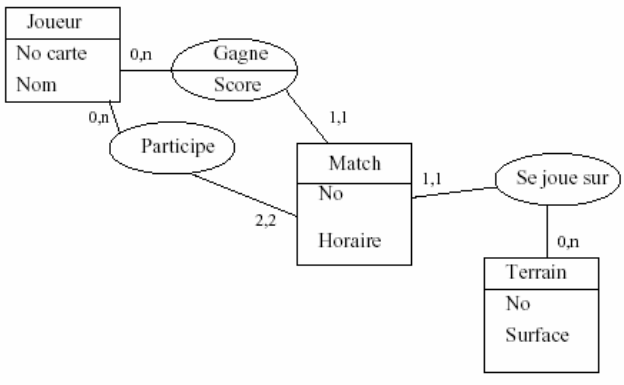


FIG. 1.2 – *Tournoi de tennis*

Q1 : Peut-on jouer des matchs de double ?

Q2 : Un joueur peut-il gagner un match sans y a voir participé ?

Q3 : Peut-il y avoir deux matchs sur le même terrain à la même heure ?

Q4 : Donner le schéma relationnel pour ce MCD.

Exercice 3 : Un journal

Voici le MCD du système d'information (très simplifié) d'un quotidien.

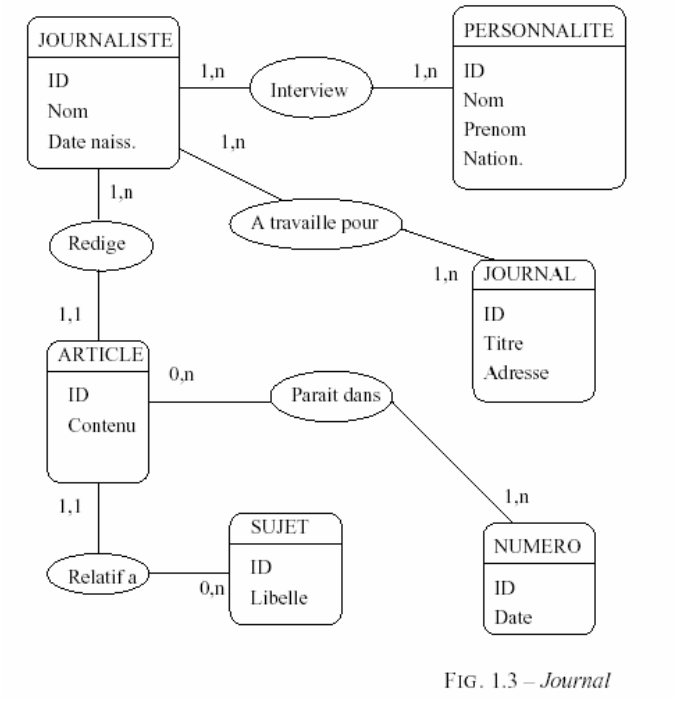


FIG. 1.3 – *Journal*

Q1 : Un article peut-il être rédigé par plusieurs journalistes ?

Q2 : Un article peut-il être publié plusieurs fois dans le même numéro ?

Q3 : Peut-il y avoir plusieurs articles sur le même sujet dans le même numéro ?

Q4 : Donner le schéma relationnel pour ce MCD.

Exercice 4:

On trouve dans un SGBD relationnel les relations ci-dessous. Les clés primaires sont soulignées, mais pas les clés étrangères.

IMMEUBLE (**Adresse**, Nb-étages, Date-construction, Nom-Gérant)

APPART (**Adresse, Numéro**, Type, Superficie, Etage)

PERSONNE (**Nom**, Age, Code-Profession)

OCCUPANT (**Adresse, Numéro-Appart, Nom-Occupant**, Date-arrivée, Date-départ)

PROPRIÉTÉ (**Adresse, Nom-Propriétaire**, Quote-part)

TYPE-APPART (**Code**, Libellé)

PROFESSION (**Code**, Libellé)

Q1 : Identifier les clés étrangères dans chaque relation.

Q2 : Reconstruire le MCD.

TP N 2 (SQL)

Employé Département

I. Schéma

Les exemples sont bâtis sur une base de données composée des deux relations suivantes :

emp (nom, num, fonction, n_sup, embauche, salaire, comm, n_dept)

dept(n_dept, nom,lieu)

Salgrade(grade, salmin,salmax)

II. Requêtes

Exprimer les requêtes suivantes en SQL.

Requête 1 : Liste des employés ayant une commission (non nulle), classée par commission décroissante.

Requête 2 : Nom des personnes embauchées depuis janvier 1991.

Requête 3 : Liste des employés dont la commission est inférieure à 10% du salaire.

Requête 4 : Donner les noms des ingénieurs embauchés avant le 1 janvier 1990.

Requête 5 : Quelles sont les fonctions ne donnant pas lieu à commission.

Requête 6 : Donner les noms et fonctions de chacun. Classer le résultat par fonction puis par nom.

Requête 7 : Donner les revenus annuels de chacun.

Requête 8 : Liste des employés travaillant à OPERATIONS.

Requête 9 : Liste des subordonnés de King

Requête 10 : Liste des employés ayant un subordonné ne se trouvant pas dans le même département.

Requête 11 : Employés ayant même directeur que king.

Requête 12 : Donner pour chaque directeur son lieu de travail.

Requête 13 : Liste des employés embauchés avant leur supérieur hiérarchique.

Requête 14 : Donner le nom du supérieur hiérarchique de JAMES.

Requête 15 : Donner, pour chaque lieu les différentes fonctions des employés y travaillant.

Requête 16 : Quelles sont les personnes travaillant à BOSTON et embauchées avant le 1er janvier 1990.

Requête 17 : Extraire les 10 premiers employés de l'entreprise.

Requête 18 : Extraire les 10 plus chers employés.

Requête 19 : Nous ne voulons maintenant afficher que les 4 plus chers employés

Requête 20 : Extraire les 5 premiers employés de l'entreprise ayant un salaire compris entre 1000 et 1300.

III. Astuces

TO_CHAR (date, format)

Format indique le format sous lequel sera affichée date. C'est une combinaison de codes ; en voici quelques uns :

YYYY année

YY deux derniers chiffres de l'année

WW numéro de la semaine dans l'année

MM numéro du mois

DDD numéro du jour dans l'année

DD numéro du jour dans le mois

D numéro du jour dans la semaine

HH ou **HH12** heure (sur 12 heures)

HH24 heure (sur 24 heures)

MI minutes

Tout caractère spécial inséré dans le format sera reproduit dans la chaîne de caractère résultat. On peut également insérer dans le format une chaîne de caractères quelconque, à condition de la placer entre guillemets.

Chaque ligne résultat du requête exécuté a un numéro sauvegardé dans **ROWNUM** ainsi pour obtenir les 10 premières lignes d'une requête il suffit d'écrire la requête suivante :

```
SELECT   *
FROM   EMP
WHERE   ROWNUM <= 10;
```

Remarque:

Tout les TPS sont Notés.

Correction du
TP 2(Employé Département)

Requête 1 : Liste des employés ayant une commission (non nulle), classée par commission décroissante.

```
select * from emp where comm is not null order by comm desc;
```

Requête 2 : Nom des personnes embauchées depuis janvier 1991.

```
select * from emp where embauche> '1/01/1991';
```

Requête 3 : Liste des employés dont la commission est inférieure à 10% du salaire.

```
select * from emp where comm < 0.1 * sal;
```

Requête 4 : Donner les noms des ingénieurs embauchés avant le 1 janvier 1990.

```
select * from emp where fonction = 'ingenieur' and embauche < '1/1/1990';
```

Requête 5 : Quelles sont les fonctions ne donnant pas lieu à commission.

```
select distinct(fonction) from emp where fonction not in (select fonction from
emp where comm is not null);
```

Requête 6 : Donner les noms et fonctions de chacun. Classer le résultat par fonction puis par nom.

```
select nom,fonction from emp order by fonction;
select nom,fonction from emp order by nom;
```

Requête 7 : Donner les revenus annuels de chacun.

```
select nom, (sal+ nvl(comm,0))*12 from emp;
```

Requête 8 : Liste des employés travaillant à OPERATIONS.

```
select * from emp,dept where dept.deptno = emp.deptno and dname =
OPERATIONS';
```

Requête 9 : Liste des subordonnés de King

```
select * from emp where n_sup = (select empno from emp where nom =
'KING');
```

Requête 10 : Liste des employés ayant un subordonné ne se trouvant pas dans le même département.

```
select ename from emp e1 where e1.empno in(select mgr from emp where
e1.deptno <> deptno);
```

Requête 11 : Employés ayant même directeur que king.

```
select * from emp where n_sup = (select n_sup from emp where nom =
'KING');
```

Requête 12 : Donner pour chaque directeur son lieu de travail.

```
select nom,lieu from emp,dept where fonction='DIRRECTEUR' and
emp.deptno=dept.deptno;
```

Requête 13 : Liste des employés embauchés avant leur supérieur hiérarchique.

```
select e1.nom from emp e1,emp e2 where e2.embauche<e1.embauche and
e2.empno=e1.n_sup;
```

Requête 14 : Donner le nom du supérieur hiérarchique de JAMES.

```
select nom from emp where empno = (select n_sup from emp where nom =
'JAMES');
```

Requête 15 : Donner, pour chaque lieu les différentes fonctions des employés y travaillant.

```
select fonction,lieu from emp,dept where emp.deptno = dept.deptno group by
fonction,lieu order by lieu;
```

Requête 16 : Quelles sont les personnes travaillant à BOSTON et embauchées avant le 1er janvier 1990.

```
select * from emp,dept where emp.deptno = dept.deptno and lieu='BOSTON'
and embauche < '1/1/1990';
```

Requête 17 : Extraire les 10 premiers employés de l'entreprise.

```
select * from emp where rownum <= 10;
```

Requête 18 : Extraire les 10 plus chers employés.

```
select * from emp where rownum <= 10 order by sal desc;
```

Requête 19 : Nous ne voulons maintenant afficher que les 4 plus chers employés

```
select * from emp where rownum <= 4 order by sal desc;
```

Requête 20 : Extraire les 5 premiers employés de l'entreprise ayant un salaire compris entre 1000 et 1300.

```
select * from emp where rownum <= 10 and sal >=1000 and sal <=1300;
```

TP N 3 (SQL)

Tour de France de cyclisme

Soit le modèle relationnel suivant relatif à la gestion simplifiée des étapes du Tour de France 2006, dont une des étapes de type "contre la montre individuel" se déroula à Saint-Etienne :

EQUIPE(CodeEquipe, NomEquipe, DirecteurSportif)
COUREUR(NuméroCoureur, NomCoureur, CodeEquipe*, CodePays*)
PAYS(CodePays, NomPays)
TYPE_ETAPE(CodeType, LibelléType)
ETAPE(NuméroEtape, DateEtape, VilleDép, VilleArr, NbKm, CodeType*)
PARTICIPER(NuméroCoureur*, NuméroEtape*, TempsRéalisé)
ATTRIBUER_BONIFICATION(NuméroEtape*, km, Rang, NbSecondes, NuméroCoureur*)

Remarque : les clés primaires sont soulignées et les clés étrangères sont marquées par *

Notes :

- La Tour de France de cyclisme est une compétition internationale annuelle.
- Plusieurs équipes y participent.
- Une équipe est formée de coureurs de différentes nationalités.
- Un Tour est composé de plusieurs étapes.
- Une étape est un parcours entre deux villes.
- Une étape a un type. Exp : « contre la montre », « haute montagne »,..etc.
- Dans une étape,il y a des kilomètres de bonification qui améliore le temps réalisé.
- Le classement est par étape et par Tour
- Le classement est par coureur et par équipe.

Questions :

1 - Quelle est la composition de l'équipe « Festina » (Numéro, nom et pays des coureurs) ?
2 - Quel est le nombre de kilomètres total du Tour de France 2006 ?
3 - Quel est le nombre de kilomètres total des étapes de type "Haute Montagne"?
4 - Quels sont les noms des coureurs qui n'ont pas obtenu de bonifications ?
5 - Quels sont les noms des coureurs qui ont participé à toutes les étapes ?

6 - Quel est le classement général des coureurs (nom, code équipe, code pays et temps des coureurs) à l'issue des 13 premières étapes sachant que les bonifications ont été intégrées dans les temps réalisés à chaque étape ?
7 - Quel est le classement par équipe à l'issue des 13 premières étapes (nom et temps des équipes) ?

Correction de TP N 3

1 - Quelle est la composition de l'équipe FESTINA (Numéro, nom et pays des coureurs) ?

```
SELECT NuméroCoureur, NomCoureur, NomPays
FROM EQUIPE A, COUREUR B, PAYS C
WHERE A.CodeEquipe=B.CodeEquipe And B.CodePays=C.CodePays
And NomEquipe="FESTINA" ;
```

2 - Quel est le nombre de kilomètres total du Tour de France 97 ?

```
SELECT SUM(Nbkm) FROM ETAPE ;
```

3 - Quel est le nombre de kilomètres total des étapes de type HAUTE MONTAGNE ?

```
SELECT SUM(Nbkm) FROM ETAPE A, TYPE_ETAPE B
WHERE A.CodeType=B.CodeType And LibelléType="HAUTE
MONTAGNE" ;
```

4 - Quels sont les noms des coureurs qui n'ont pas obtenu de bonifications ?

```
SELECT NomCoureur FROM COUREUR
WHERE NuméroCoureur NOT IN (SELECT NuméroCoureur FROM
ATTRIBUER_BONIFICATION) ;
```

5 - Quels sont les noms des coureurs qui ont participé à toutes les étapes ?

```
SELECT NomCoureur FROM PARTICIPER A, COUREUR B
WHERE A.NuméroCoureur=B.NuméroCoureur
GROUP BY NuméroCoureur, NomCoureur
HAVING COUNT(*)=(SELECT COUNT(*) FROM ETAPE) ;
```

6 - Quel est le classement général des coureurs (nom, code équipe, code pays et temps des coureurs) à l'issue des 13 premières étapes sachant que les bonifications ont été intégrées dans les temps réalisés à chaque étape ?

```
SELECT NomCoureur, CodeEquipe, CodePays, SUM(TempsRéalisé) AS Total
FROM PARTICIPER A, COUREUR B
WHERE A.NuméroCoureur=B.NuméroCoureur and NuméroEtape<=13
GROUP BY A.NuméroCoureur, NomCoureur, CodeEquipe, CodePays
ORDER BY Total;
```

7 - Quel est le classement par équipe à l'issue des 13 premières étapes (nom et temps des équipes) ?

```
SELECT NomEquipe, SUM(TempsRéalisé) AS Total
FROM PARTICIPER A, COUREUR B, EQUIPE C
WHERE A.NuméroCoureur=B.NuméroCoureur And
B.CodeEquipe=C.CodeEquipe
And NuméroEtape<=13
GROUP BY B.CodeEquipe, NomEquipe
ORDER BY Total;
```

TP N 4 (SQL)

Soit la base de données **CINEMA** suivante :

FILM (NUM-F, TITRE, DATE, LONGUEUR, BUDGET, REALISATEUR, SALAIRE-R)
GENERIQUE (FILM, ACTEUR, ROLE, SALAIRE)
PERSONNE (NUM-P, PRENOM, NOM, DATENAIS, SEXE, NATIONALITE, ADRESSE, TELEPHONE)
CINEMA (NUM-C, NOM, ADRESSE, TELEPHONE, COMPAGNIE)
PASSE (NUM-F, CINEMA, HORAIRE, DATE-DEB, DATE-FIN, SALLE, PRIX)
SALLE (NUM-S, CINEMA , TAILLE-ECRAN, PLACES)

NUM-F, NUM-P, NUM-C, NUM-S sont des identifiants uniques (clés primaires) pour respectivement : FILM, PERSONNE, CINEMA, SALLE. Tout nom de relation utilisé comme attribut est une clé étrangère qui renvoie à l'identifiant (clé primaire) de la relation correspondante, par exemple dans GENERIQUE, FILM correspond à NUM-F de FILM et est défini sur le même domaine. REALISATEUR, dans FILM, correspond à NUM-P.

1. Donner les titres des films réalisés par Roman Polanski.
2. Donner les films qui ne passent dans aucun cinéma de la compagnie FOX.
3. Donner le prénom, le nom et le numéro des acteurs qui ont joué dans tous les films de Lelouch.
4 Trouver le nom et le prénom des acteurs qui ont eu un salaire total plus important, dans un film
particulier, que le salaire du réalisateur du même film.
5. Pour chaque film de Bergman, trouver le nom et le prénom de l'acteur qui a eu le plus gros
salaire pour un rôle dans ce film.
6. Lister les cinémas dont la taille moyenne d'écran est supérieure à 40 mètres carré.

TP N 5 (SQL)

Soit la base de données « tennis »:

JOUEUR (numjoueur, nom, prenom, ann_naiss, nationalité)
GAIN (nujoueur, lieutournoi, annee, prime, sponsor)
RENCONTRE (numgagant, numpardant, lieutournoi, annee, score)

a) Numéro et tournoi d'engagement (défini par le lieu et l'année) des joueurs sponsorisés par Peugeot entre 1990 et 1994 ;

b) Nom et année de naissance des joueurs ayant participé à Roland Garros en 1994 ;

c) Nom et nationalité des joueurs ayant participé à la fois au tournoi de Roland Garros et à celui de Wimbledon, en 1992 ;

d) Nom et nationalité des joueurs ayant été sponsorisés par Peugeot et ayant gagné à Roland Garros au moins un match (avec un sponsor quelconque);

e) Nom des joueurs ayant toutes leurs primes à Roland Garros supérieures à 1MF ;

f) Numéros des joueurs qui ont toujours perdu à Wimbledon, et toujours gagné à Roland Garros ;

g) Liste des vainqueurs de tournoi, mentionnant le nom du joueur avec le lieu et l'année du tournoi qu'il a gagné ;

m) Moyenne des primes gagnées par année.

n) Valeur de la plus forte prime attribuée lors d'un tournoi en 1992, et noms des joueurs qui l'ont touchée.
o) Somme gagnée en 1992 par chaque joueur, pour l'ensemble des tournois auxquels il a participé (présentation par ordre de gain décroissant).

p) Noms et prénoms des vainqueurs du Simple Homme et du Simple Dame du tournoi de Roland Garros en 1992.

q) Nom des joueurs ayant participé à tous les tournois de Roland Garros.

r) Pour chaque joueur, noms des adversaires qu'il a toujours battus.

s) Noms des sponsors représentés à tous les tournois.

t) Noms des pays qui ont eu un vainqueur de tournoi chaque année.

TP N 1 PL/Sql

- Affiche RI41 à l'écran

```
DECLARE
   W_TEMP VARCHAR2(10)

BEGIN
   W_TEMP:='RI41';
   DBMS_OUTPUT.PUTLINE('W_TEMP');
END;
```

A noter : Set Serveroutput on; permet de visualiser les messages utilisateur

-- Regroupez les deux chaînes « RI » et « 41 »

```
DECLARE
   W_TEMP VARCHAR2(10);
   W_TEMP2 VARCHAR2(10);

BEGIN
   W_TEMP:='RI';
   W_TEMP2:='41';
   W_TEMP=W_TEMP||W_TEMP2;
   DBMS_OUTPUT.PUTLINE(W_TEMP);
END ;
```

-- Affiche les nombres de 1 à 15

```
DECLARE
   W_TEMP VARCHAR2(20) ;

BEGIN
   FOR I IN 1..15 LOOP
      W_TEMP:=i ;
      DBMS_OUTPUT.PUT_LINE(W_TEMP);
   END LOOP;
END;
```

-- Lecture table emp et affichage du premier enregistrement trouvé

```
DECLARE
CURSOR CTP IS SELECT ename FROM emp;
W_LIBELLE VARCHAR(30);
```

```
BEGIN
   OPEN CTP;
   FETCH CTP INTO W_LIBELLE;
   CLOSE CTP;
   DBMS_OUTPUT.PUT_LINE(W_LIBELLE);
END;
```

A noter : ne prend en compte que le premier enregistrement trouvé

-- Lecture table emp et affichage des 5 premiers noms

```
DECLARE
CURSOR CTP IS SELECT ename FROM emp;
W_LIBELLE VARCHAR(30);

BEGIN
   FOR I IN 1..5 LOOP
      FETCH CTP INTO W_LIBELLE;
      DBMS_OUTPUT.PUT_LINE(W_LIBELLE);
   END LOOP;
CLOSE CTP1;
END;
```

-- Lecture table emp et affichage de tous les enregistrements

```
DECLARE
CURSOR CTP IS SELECT ename FROM emp;
W_LIBELLE VARCHAR2(30);
BEGIN
open CTP;
LOOP
   FETCH CTP INTO W_LIBELLE;
   EXIT WHEN CTP%NOTFOUND;
   DBMS_OUTPUT.PUT_LINE(W_LIBELLE);
END LOOP;
close CTP;
END;
```

-- Lecture table emp et affichage du dernier Numéro et nom

```
DECLARE
CURSOR CTP IS SELECT EMPNO, ename FROM emp ;
W_LIBELLE VARCHAR2(30) ;
W_NUM NUMBER(7,0) ;

BEGIN
   OPEN CTP ;
   LOOP
      FETCH CTP INTO W_NUM,W_LIBELLE;
```

```
      EXIT WHEN CTP %NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(W_NUM||W_LIBELLE) ;
   END LOOP ;
CLOSE CTP ;
END ;
```

A noter : RPAD(W_LIBELLE,30,' ') permet de cadrer à droite et de remplir la chaîne d'espaces

A noter :

W_LIBELLE VARCHAR2(30) ; -- Formule statique
W_LIBELLE emp.enam%TYPE – Dynamique (évite les maintenances)

Premières Fonctions et procédures :

Exercice 1 :

Réaliser une fonction Calcul_Revenue_Annuel qui renvoie le revenu annuel d'un employé donné par son numéro.

```
CREATE OR REPLACE FUNCTION F_MTT (E_NUM NUMBER) RETURN NUMBER IS
CURSOR C IS
SELECT (SAL*12 + NVL(COMM,0) ) FROM EMP WHERE EMPNO=E_NUM;
W_MT NUMBER:=0;

BEGIN
   OPEN C;
   FETCH C INTO W_MT;
   CLOSE C;
   RETURN W_MT;
END F_MTT;
```

Utilisation de la fonction :

```
-- Revenue annual de claque employé
SELECT F_MTT (EMPNO) FROM EMP;

-- Revenue annual de l'employé numero 7698
SELECT F_MTT (7698) FROM dual;
```

Remarque: Essayer : SELECT F_MTT (7698) FROM emp;

Inetrpretez???

```
-- Somme de toutes les revenues
SELECT SUM(F_MTT (EMPNO)) FROM EMP;

-- Avec un DECLARE
```

```
DECLARE
   W_TEMP NUMBER :=0 ;

BEGIN
   W_TEMP:=F_MTT (7698);
   DBMS_OUTPUT.PUT_LINE(W_TEMP);
END;
```

Exercice 4 :

Recherchez le nom et le job d'un employée passé en paramètre

```
CREATE OR REPLACE PROCEDURE P_INFOS
(P_Num in number, P_Nom out varchar2, P_job out varchar2) is

CURSOR C IS
SELECT ename, job FROM emp WHERE EMPNO =P_NUM;

BEGIN

   OPEN C;
   FETCH C  INTO P_Nom, P_job;
   CLOSE C;

END P_INFOS;
```

Exercice 7 :

Fonction ajout de deux nombres

```
CREATE OR REPLACE FUNCTION F_AJOUT (P1 NUMBER, P2 NUMBER) RETURN
NUMBER IS
CURSOR C_SOMME IS SELECT P1+P2 FROM DUAL;

W_SOMME NUMBER;

BEGIN
   OPEN C_SOMME;
   FETCH C_SOMME INTO W_SOMME;
   DBMS_OUTPUT.PUT_LINE(W_SOMME);
   CLOSE C_SOMME;
   RETURN W_SOMME;
END;
```

Utilisation :

```
SELECT F_AJOUT(3,4) FROM DUAL;
```

Ou

```
DECLARE
W_SOMME NUMBER:=0;

BEGIN
   W_SOMME:=F_AJOUT(3,5);
   DBMS_OUTPUT.PUT_LINE(W_SOMME) ;
END;
```

Exercice 8 :

Concaténez deux chaînes de caractères

```
CREATE OR REPLACE FUNCTION F_AJOUT (P1 VARCHAR2, P2 VARCHAR2)
RETURN VARCHAR2 IS
P3 VARCHAR2(10)

BEGIN
   P3:=P1||P2;
   DBMS_OUTPUT.PUT_LINE(P3);
   RETURN P3;
END;
```

Utilisation :

```
SELECT F_AJOUT('Fred','Eric') FROM DUAL;
```

A noter : Dans le cas d'une création de fonction ou de procédure, c'est le nom qui importe. Le nom doit être unique. Sinon, il faut utiliser des packages afin de regrouper les fonctions et procédures et avoir la possibilité d'utiliser le même nom.

Exercice 3 :

Ecrire la procédure qui donne la liste de tous les employées (triés par empno).

```
Create or replace Procedure P_Affiche is

Cursor C is
   Select empno, ename from emp
   order by empno;

W_Num emp.empno %Type ;
W_Nom varchar2(30);
```

```
BEGIN
   Open C;
   LOOP
      FETCH C into W_Num,W_Nom ;
      Exit when C%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE( To_char (W_Num) || W_Nom) ;
   END LOOP;
   CLOSE C;

End P_Affiche;
```

Utilisation :

```
Execute P_Affiche;
```

TP N 2 PL/SQL.

Exemples de script PL/SQL.

Les extensions de SQL qui s'appellent PL/SQL permettent de réaliser des traitements procéduraux et de travailler avec des enregistrements tampons que l'on appelle curseur ou cursor. Ce sont les curseurs qui permettent des traitements de type boucle (loop, for…). Le PL/SQL propose également toutes une série de fonctions : fonctions d'entrées-sorties, fonction de formatage de données (to_date pour mettre en forme les dates, to_char pour une présentation alphanumérique…).

Voici quelques exemples :

Tout d'abord une boucle (loop..end loop) qui affiche 10 message à l'écran de type voici le numéro de passage.

Notez :

le signe d'affectation :=
la fonction d'affichage put_line qui fait partie du package (ensemble de procédures et fonctions regroupées) dbms_output ;
le signe de concaténation || qui permet de construire une ligne composée de plusieurs parties ;
; matérialise la fin d'une instruction PL/SQL et on indique la fin du traitement par /

```
declare
i number:=0;
begin
loop
i:=i+1;
dbms_output.put_line('voici le'|| i || ' ieme passage...');
exit when i=10;
end loop;
end;
/
```

```
SQL> start p10
voici le1 ieme passage...
voici le2 ieme passage...
voici le3 ieme passage...
voici le4 ieme passage...
voici le5 ieme passage...
voici le6 ieme passage...
voici le7 ieme passage...
voici le8 ieme passage...
voici le9 ieme passage...
voici le10 ieme passage...
PL/SQL procedure successfully completed.
```

TP 3: Que font les procédures et fonctions suivantes ?

```
DECLARE
v_salaire  NUMBER(7)  --variable locale
BEGIN
select sal into v_salaire from emp
where ename='SCOTT' for update of sal ;
IF v_salaire <= 1000 THEN
UPDATE emp set sal=sal*1.1 where ename='SCOTT' ;
ELSIF v_salaire BETWEEN (1000 and 5000) THEN
UPDATE emp set sal=sal*1.05 where ename='SCOTT' ;
ELSE UPDATE emp set comm=comm+100 where ename='SCOTT' ;
END IF ;
COMMIT ;
END ;
```

```
DECLARE
V_cinq  CONSTANT :=5 ;
V_resultat   NUMBER ;
i  NUMBER :=1 ;
BEGIN
WHILE i <=9
LOOP
V_resultat := V_cinq * i ;
INSERT INTO table5 VALUES (V_cinq || '*' || i || '=' || V_resultat) ;
I :=i+1 ;
END LOOP ;
END ;
/
```

```
CREATE OR REPLACE FUNCTION f_euro (v_francs IN NUMBER)
RETURN REAL IS
V_euro REAL ;
BEGIN
v_euro :=v_francs / 6.55957 ;
return v_euro ;
END ;
/
```

```
DECLARE
v_francs REAL :='1000' ;
v_euro REAL ;
BEGIN
dbms_output.put_line('Valeur en francs :'|| to_char(v_francs)) ;
v_euro := f_euro (v_francs) ;
dbms_output.put_line('Valeur en euros :'|| to_char(v_euro)) ;
END ;
/
```

Dans cet exemple, on déclare un curseur (mémoire intermédiaire de travail). Le curseur est rempli par un ordre SQL (select empno,ename,dname from emp,dept where dept.deptno=emp.deptno and dname='SALES' order by ename;) et on traite successivement toutes les lignes du curseur grâce à la variable ligne (qui comporte les champs empno, ename et dname) :on affiche juste le nom de l'employé et le nombre total d'employés à la fin du département SALES (pour cette opération simple un simple ordre SQL aurait suffit). Le script est mis dans le fichier p10.sql ce qui permet des modifications et de réutiliser les lignes de code SQL.

```
DECLARE
numemp number(10):=0;
cursor C1 is
select empno,ename,dname from emp,dept
where dept.deptno=emp.deptno
and dname='SALES'
order by ename;
begin
FOR ligne in C1
loop
dbms_output.put_line(ligne.ename);
numemp:=numemp+1;
end loop;
dbms_output.put_line(numemp);
end;
/
```

```
SQL> start p10
PL/SQL procedure successfully completed.
```

Attention : on ne voit rien si on active pas l'affichage à l'écran pas la commande set serveroutput on. Cette commande se tape en début de session sqlplus et reste valable pendant toute la durée de la session.

```
SQL> set serveroutput on
SQL> start p10
ALLEN
BLAKE
JAMES
MARTIN
TURNER
WARD
PL/SQL procedure successfully completed.
```

Cet exemple ressemble au précédent mais une autre table est impactée (insertion de lignes dans cette table). Mais dans cette exemple la variable ligne a la même structure qu'une ligne de la table emp (c%rowtype), le curseur doit être ouvert quand on en a besoin par la commande open (pour des raisons d'économie de place mémoire). On transfère chaque ligne du curseur une par une dans la variable ligne avec la commande fetch into.
La condition d'arrêt est c%notfound c'est à dire que la boucle s'arrête quand il n'y a plus de lignes à traiter.

Il faut définir une autre table que l'on appellera table_temporaire :

```
SQL> create table table_temporaire
  2  as select empno,ename,sal from emp where 1=2;
Table created.
```

```
declare
cursor c is select * from emp
where sal<2000;
ligne c%rowtype;
begin
open c;
loop
fetch c into ligne;
exit when c%notfound;
if ligne.sal<2000 then
insert into table_temporaire values
(ligne.empno,ligne.ename,ligne.sal);
end if;
end loop;
close c;
commit;
end;
/
```

Le résultat se voit dans la table table_temporaire :

```
SQL> select * from table_temporaire;
Appuyer sur ENTREE pour continuer...
 EMPNO ENAME          SAL
------ ---------- --------
  7369 SMITH             800
  7499 ALLEN             1600
  7521 WARD             1250
  7654 MARTIN           1250
  7844 TURNER            1500
  7876 ADAMS            1100
  7900 JAMES            950
  7934 MILLER           1300
8 rows selected.
```

Les triggers.

Les triggers sont des procédures prédéfinies qui seront déclenchées (lancées) de façon automatique en plus lors de la réalisation sur la base de l'opération SQL mentionnée dans le trigger ou déclencheur. Dans notre exemple, on souhaite à chaque INSERT ou UPDATE sur la table emp, vérifier que le salaire est compris dans une fourchette de salaire (les fourchettes de salaire sont définies pour chaque métier dans la table sal_guide). Il n'y a pas de contrôle pour le président !

```
SQL> select * from sal_guide;
Appuyer sur ENTREE pour continuer...
  MINSAL   MAXSAL     JOB
-------- -------- ---------
     500     9500             SALESMAN
    1000     4000             CLERK
    1000     4000             ANALYST
    2000     7000             MANAGER
```

Remarquez :

Create or replace trigger pour remplacer l'ancienne version du trigger s'il existe déjà, :NEW qui est un opérateur qui permet de préciser la nouvelle valeur d'un champ (il existe aussi :OLD) ; ainsi on peut faire une distinction entre l'ancienne et la nouvelle valeur d'une rubrique qui va être changée.
FOR EACH ROW : toutes les lignes de la tables sont contrôlées.
La fonction RAISE_APPLICATION_ERROR qui permet d'envoyer un message en bas de l'écran de l'utilisateur.

```
create or replace trigger verif_salaire
before INSERT or UPDATE of sal,job on emp
for EACH ROW
when (NEW.job <> 'PRESIDENT')
DECLARE
vminsal number;
vmaxsal number;
BEGIN
select minsal,maxsal INTO vminsal,vmaxsal from sal_guide
where job= :NEW.job;
if :NEW.sal < vminsal or :NEW.sal > vmaxsal
then RAISE_APPLICATION_ERROR (-20601, 'Salaire' || :NEW.sal ||
'hors tranche pour la profession' || :NEW.job ||
'de l employe' || :NEW.ename);
end if;
end;
/
```

```
SQL> start p9
Trigger created.
```

TP N 4(PL/SQL)

Exercice 1 :

Soit le schéma suivant :

AVION (AvNum, AvNom, Capacite, Localisation)
PILOTE (PlNum, PlNom, PlPrenom, Ville, Salaire)
VOL (VolNum, PlNum#, AvNum#, VilleDep, VilleArr, HeureDep, HeureArr)

Les performances des avions de marque Airbus évoluent, aussi souhaite t-on faire des mises à jour
de la table VOL. Les temps de vol des avions de type A300 (avions n° 1 et 4) doivent être réduits de 10 % et ceux des avions de type A310 (avions n° 2 et 8) de 15 %. Il s'agit de définir un programme PL/SQL permettant ces modifications.
1. Recopier les tables PILOTE, AVION et VOL du compte DARMONT sur le vôtre.
2. Dans un bloc PL/SQL anonyme, déclarer un curseur permettant de lire les données suivantes : numéro de vol, numéro d'avion, heure de départ et heure d'arrivée des vols pour lesquels l'avion utilisé est le n° 1, 2, 4 ou 8. Pour chaque vol lu par le curseur, calculer le temps de vol, le réduire dans la proportion voulue selon l'avion utilisé, puis mettre à jour l'attribut HEUREARR de ce vol dans la table VOL.
3. Tester !

On souhaite gérer les résultats d'examens de la Faculté de Sciences Économiques et de Gestion. Il s'agit de définir un programme PL/SQL permettant l'insertion automatique d'informations dans les relations « résultats » RESULTAT et CLASSEMENT, à partir des données des relations « sources » ETUDIANT, NOTATION et MATIERE, qui contiennent respectivement des renseignements sur les étudiants, les notes obtenues par les étudiants et les coefficients affectés aux matières.
Pour définir ce programme, suivre les étapes suivantes.

1) Définir en SQL la structure des relations RESULTAT et CLASSEMENT :
• la relation RESULTAT a pour attributs un numéro d'étudiant, un nom d'étudiant, un code matière, ainsi qu'un attribut note globale pour cet étudiant ;
• la relation CLASSEMENT a pour attributs un numéro d'étudiant, un nom d'étudiant, une moyenne générale et un rang (place au classement).
Ne pas inclure de contrainte d'intégrité dans la définition de ces deux relations, qui sont temporaires et ne servent qu'au stockage des résultats.
2) Définir un bloc PL/SQL anonyme permettant d'insérer dans RESULTAT tous les n-uplets constitués du numéro d'un étudiant, de son nom, du code d'une matière et de la note obtenue par cet étudiant dans cette matière. Le calcul de cette note doit tenir compte des coefficients de contrôle continu et d'examen définis pour la matière en question, ainsi que de la possibilité d'avoir des valeurs nulles pour les notes des étudiants, qui sont alors assimilées à 0 (utiliser la fonction NVL). Les n-uplets considérés doivent être extraits des tables ETUDIANT, NOTATION et MATIERE de manière itérative, grâce à un curseur adapté.

3) Terminer le traitement en réalisant l'insertion dans la relation CLASSEMENT des n-uplets constitués du numéro d'un étudiant, de son nom, de son prénom, de la moyenne générale obtenue dans toutes les matières par cet étudiant (ces informations doivent être extraites de la table RESULTAT) et de son rang (place), qui doit être calculé. Pour simplifier, on considère que toutes les matières sont équivalentes en termes de notes. Utiliser un curseur dans lequel les enregistrements sont triés.
Questions complémentaires
Modifier le programme afin de prendre en compte :
• le cas où plusieurs étudiants ont le même rang (ex æquo) ;
• le cas d'étudiants n'ayant aucune note (par défaut, leur moyenne générale est 0) ;
• la possibilité de n'avoir aucun n-uplet dans la relation ETUDIANT. Dans ce cas, un n-uplet de valeur (0, 'Aucun étudiant', NULL, NULL) doit être inséré dans la relation RESULTAT et le traitement doit s'arrêter.

Exercice 2 : Curseur simple et curseur dynamique imbriqués

1. Modifier le bloc PL/SQL de l'exercice 1 de manière à afficher le schéma de toutes les tables et vues présentes dans votre catalogue système (vue TAB (TNAME, …)).
2. Ajouter à la description du schéma de chaque table son nombre de n-uplets. Dans le cas où la table est vide (zéro n-uplet), on affichera de préférence la chaîne « Vide ».
3. Transformer le bloc PL/SQL anonyme en procédure stockée nommée « schema ». Comment généraliseriez-vous cette procédure à tout utilisateur ?

Exercice 3 : Contraintes de domaine et contraintes dynamiques dans un déclencheur

Soit le schéma relationnel d'une agence bancaire régionale.

CLIENT (NumCl, Nom, Prenom, Adr, CP, Ville, Salaire, NumConjoint#)
DETENTEUR (NumCl#, NumCpt#)
COMPTE (NumCpt, DateOuvr, Solde)

On souhaite mettre en oeuvre un déclencheur avant insertion ou mise à jour permettant de contrôler les contraintes suivantes :
− le département dans lequel habite le client doit être 01, 07, 26, 38, 42, 69, 73, ou 74 (région Rhône-Alpes) ;
− le nom du conjoint doit être le même que celui du client.
1. À l'aide du langage SQL, créer la structure simplifiée de la table CLIENT (NumCl, Nom, CP, NumConjoint#).
2. Écrire le code du déclencheur, puis le créer.

Rappel : Définition d'un déclencheur
CREATE [OR REPLACE] TRIGGER Nom_Déclencheur
BEFORE | AFTER
INSERT | DELETE | UPDATE | [INSERT] [[OR] DELETE] [[OR] UPDATE]
ON Nom_Table
[FOR EACH ROW]
-- Bloc PL/SQL contenant le traitement à effectuer

3. Exécuter le déclencheur plusieurs fois en insérant des n-uplets dans la table CLIENT.
4. Vérifier son bon fonctionnement en affichant le contenu de la table CLIENT.

Exercice 4 : Numérotation automatique de clé primaire à l'aide d'un déclencheur

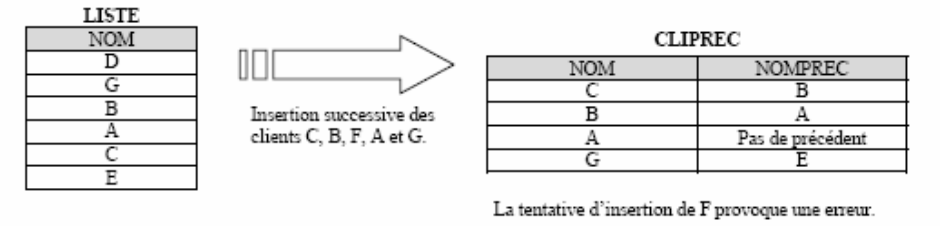
Soit une table quelconque TABL, dont la clé primaire CLENUM est numérique. Définir un déclencheur avant insertion permettant d'implémenter une numérotation automatique de la clé. Le premier numéro doit être 1.

1. Créer à l'aide de SQL la structure de la table TABL (Clenum).
2. Saisir le code du déclencheur adéquat dans un fichier, puis le créer.
3. Exécuter le déclencheur plusieurs fois en insérant des n-uplets dans la table TABL, puis supprimer un n-uplet et en insérer un dernier.
4. Vérifier son bon fonctionnement en affichant le contenu de la table TABL.

Exercice 5 : Utilisation de curseur dans un déclencheur

Soit LISTE une table de noms non triée, contenant un seul attribut NOM de type VARCHAR(20). On souhaite insérer dans une table CLIPREC des noms (de client) et, pour chacun, le nom du client qui le précède dans la LISTE (suivant l'ordre alphabétique). La table CLIPREC ne contient que deux champs : NOM et NOMPREC, tous deux de type VARCHAR(20).

1. Définir à l'aide de SQL la structure des tables LISTE et CLIPREC. Peupler la table liste comme ci-dessous.
2. Définir un déclencheur avant insertion sur CLIPREC permettant la recherche et l'insertion automatique du nom du client précédent lors de l'insertion d'un nom de client. Gérer le cas où il n'existe pas de précédent (premier de la liste). Gérer le cas d'erreur où le nom saisi n'existe pas dans la table LISTE. Il est interdit de modifier la table LISTE.



Exercice 2 : Statistiques d'utilisation automatiques

On souhaite conserver des statistiques concernant les mises à jour (insertions, modifications, suppressions) sur la table EMP1.

1. Créer à l'aide de SQL la structure de la table STATS et la peupler comme suit.

STATS (TypeMaj, NbMaj, Date_derniere_Maj)

TypeMaj	NbMaj	Date_derniere_Maj
INSERT	0	NULL
UPDATE	0	NULL
DELETE	0	NULL

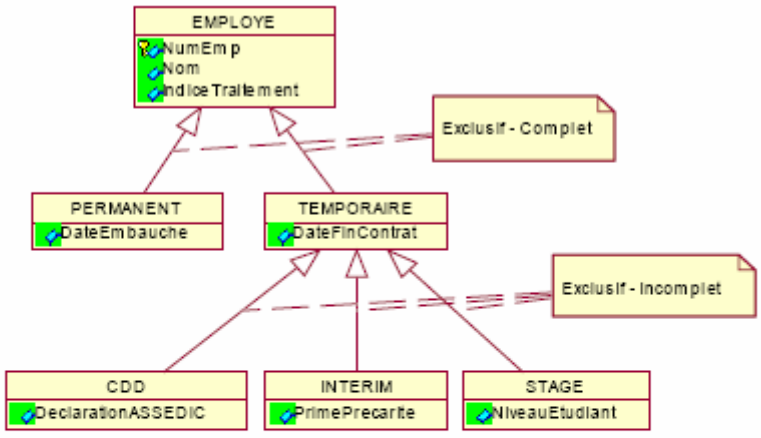
2. Définir un déclencheur après insertion/modification/suppression de la table EMP permettant de mettre à jour automatiquement la table STATS. Tester son utilisation en effectuant diverses mises à jour sur la table EMP.

Indications :
• Détermination du type de mise à jour :
IF INSERTING THEN -- insertion
IF UPDATING THEN -- modification
IF DELETING THEN -- suppression
• Date système : SYSDATE
3. Tester l'effet de la présence et de l'absence de la clause FOR EACH ROW sur le comportement du déclencheur en utilisant une requête qui modifie plusieurs n-uplets (ex. UPDATE EMP SET SAL = SAL * 1.05;)

Exercice 4 : Contraintes d'exclusivité / complétude dans un schéma objet

On souhaite implanter sous Oracle la hiérarchie de généralisation-spécialisation représentée sous forme de schéma conceptuel UML dans la figure ci-contre.
Sa traduction en schéma logique relationnel est la suivante.
Employe (NumEmp, Nom, IndiceTraitement, Type) Type appartient à { PERM, TEMP }
Employe_Perm (NumEmp#, DateEmbauche)
Employe_Temp (NumEmp#, DateFinContrat, Type) Type = CDD | INTERIM | STAGE | …
Employe_CDD (NumEmp#3, DeclarationASSEDIC)
Employe_Interim (NumEmp#3, PrimePrecarite)
Employe_Stage (NumEmp#3, NiveauEtudiant)

Cette structure est également décrite par des métadonnées stockées dans la table « Meta » dont l'extension est fournie ci-contre.
Créer le schéma physique Oracle de cette base de données (table de métadonnées comprise) en garantissant le respect des contraintes d'intégrité induites par la hiérarchie de généralisation/spécialisation.
Pour cela, exploiter :



META

IdClasse	NomTable	IdSuperclasse
0	Employe	NULL
1	Employe_Perm	0
2	Employe_Temp	0
3	Employe_CDD	2
4	Employe_Interim	2
5	Employe_Stage	2

• les contraintes d'intégrité intégrées dans Oracle :

o clés primaires,
o clés étrangères (que l'on pourra spécifier en incluant la clause ON DELETE CASCADE4 afin que lors de la suppression d'une instance de superclasse, les éventuelles instances correspondantes dans les sous-classes soient supprimées automatiquement),
o contrainte de domaine pour l'attribut Type de la table Employe ;
• un déclencheur par table correspondant à une sous-classe pour les contraintes d'exclusivité.
Les cinq déclencheurs à mettre en oeuvre sont pratiquement identiques. Ils servent à s'assurer que, alors de l'insertion ou de la mise à jour d'un n-uplet dans une table correspondant à une sous-classe, ce n-uplet n'est pas déjà présent dans une autre sous-classe de la superclasse. Deux stratégies différentes peuvent être employées pour ce faire :
1. rechercher la table correspondant à la superclasse de la sous-classe traitée, rechercher dans la table-superclasse le type du n-uplet inséré ou modifié dans la table-sousclasse (à l'aide d'une requête dynamique), vérifier que le type est correct (par exemple, PERM pour Employe_Perm) ;
2. rechercher les tables correspondant aux autres sous-classes de la superclasse de la sousclasse traitée (curseur), pour chacune d'entre elles, compter le nombre de n-uplets d'identifiant identique à celui qui est inséré ou modifié dans la table traitée (à l'aide d'une requête dynamique), vérifier que le total de ces comptes est égal à zéro.

Implémenter chaque stratégie au moins une fois. Quelle est la moins coûteuse ? La plus sûre ? Établir un jeu d'essais permettant de tester toutes les contraintes mises en œuvre ainsi que la suppression de n-uplets en cascade.