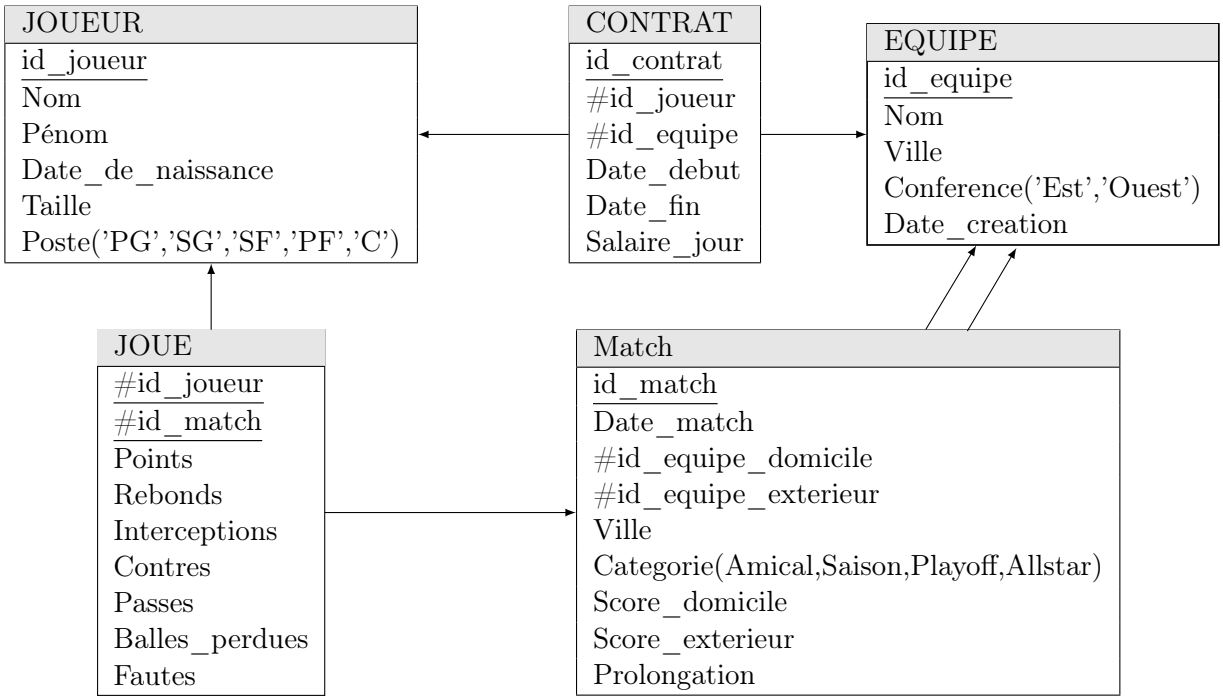
NOM :
PRENOM :
GROUPE :

Examen
90 minutes, 75 points
Aucun document autorisé

PL/SQL : NBA Statistiques

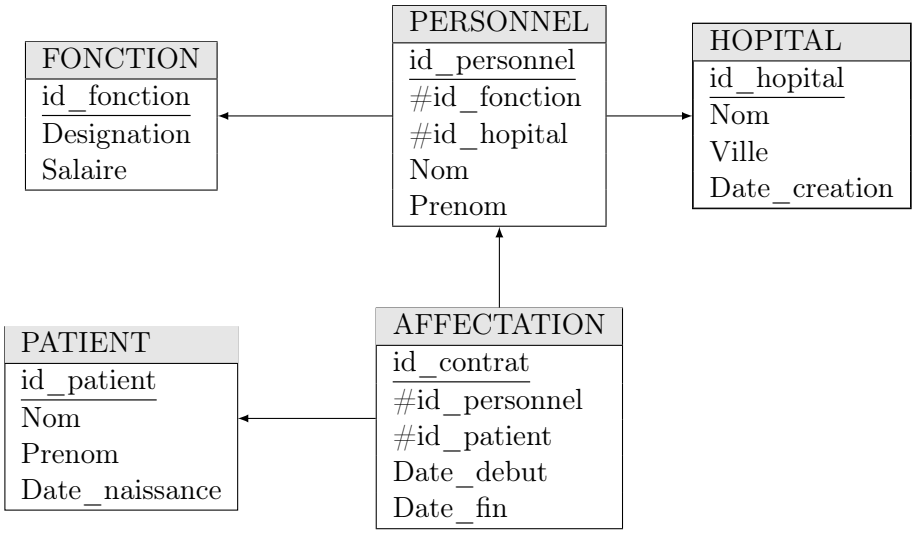
Ci-dessous un modèle de la base de données de statistiques de la NBA (National Basketball Association), où PG, SG, SF, PF et C sont les différents postes occupés par les joueurs sur le terrain. Un joueur ne peut pas être en activité dans deux équipes en même temps.



Exercice 1 (NBA, PLSQL)**.** (30 points) Le MLD ci-dessus stocke les données des matchs NBA.

1. (15 points) Écrire un code PL/SQL qui, pour un `id_joueur` saisi par l'utilisateur, liste dans une table `TLIGNES` son maximum de points lors d'un match de sa carrière. La solution proposée doit mettre en place deux mécanismes un avec et un sans `EXCEPTION` pour traiter les cas d'erreurs suivants :
   — `'joueur inconnu'`, il n'y a pas le joueur dans la table `JOUEUR`,
   — `'joueur n a pas fait de match'`, il n'apparait pas dans la table `JOUE`.
2. (15 points) Écrire un code PL/SQL qui liste dans une table `TLIGNES` les joueurs en activité (ayant une date de fin de contrat non nulle) du plus vieux au plus jeune.

```
Nom      Prenom    Date de naissance      Nom Equipe Actuelle
XXXXXX   XXXXXXX   XX/XX/XXXX             XXXXXXXX
XXXXXX   XXXXXXX   XX/XX/XXXX             XXXXXXXX
XXXXXX   XXXXXXX   XX/XX/XXXX             XXXXXXXX
...
```



Exercice 2 (Pro C)**.** (45 points)

1. (30 points) En utilisant les curseurs, écrire un programme Pro C qui permet d'afficher le nom et prénom des membres du personnel par hopital en indiquant pour chacun le nombre de patients suivis et par ordre alphabétique des prénoms ayant la forme suivante :

```
Personnels de l'hopital XXXXX :
   1) Nom : XXXXXX Prenom : XXXXX  Salaire: XXXXX  Nombre de patients : ZZ
   2) Nom : XXXXXX Prenom : XXXXX  Salaire: XXXXX  Nombre de patients : ZZ
   3) Nom : XXXXXX Prenom : XXXXX  Salaire: XXXXX  Nombre de patients : ZZ
   ...
Nombre de patients total sur l'hopital : ZZ1
Nombre de personnels total sur l'hopital : ZZ2

Personnels de l'hopital XXXXX :
   1) Nom : XXXXXX Prenom : XXXXX  Salaire: XXXXX  Nombre de patients : ZZ
   2) Nom : XXXXXX Prenom : XXXXX  Salaire: XXXXX  Nombre de patients : ZZ
   3) Nom : XXXXXX Prenom : XXXXX  Salaire: XXXXX  Nombre de patients : ZZ
   ...
Nombre de patients total sur l'hopital : ZZ1
Nombre de personnels total sur l'hopital : ZZ2
...
```

   Sachant que la valeur de `salaire` peut ne pas être renseignée, il faudra donc gérer ce cas.
2. (15 points) Écrire un programme Pro C qui permet d'insérer un nouveau membre du personnel. Les valeurs saisies par l'utilisateurs seront insérées dans la table `PERSONNEL`.
   — Vérifier que la fonction existe déjà ainsi que l'hopital.
   — Vérifier aussi que l'`id_personnel` n'existe pas déjà, s'il existe alors afficher les caractéristiques du personnel existant.

Rappels

Les fonctions `connexion`, `deconnexion` et `sql_error` sont rappellées ci-dessous. Elles ne sont pas à réécrire sur vos copies, mais à utiliser dans les exercices si besoin.

```
void connexion()
```

```
{ VARCHAR uid[50];
char login[20];
char passwd[20];
printf("Donner votre login : ");
scanf("%s",login);
printf("\nDonnez votre mot de passe Oracle : ");
scanf("%s",passwd);
printf("\n");
strcpy(uid.arr,login);
strcat(uid.arr,"/");
strcat(uid.arr,passwd);
strcat(uid.arr,"@kirov");
uid.len=strlen(uid.arr);

EXEC SQL CONNECT :uid;
if (sqlca.sqlcode==0)
printf(" Connexion réussie avec succès.\n\n");
else
{
printf ("Problème à la connexion.\n\n");
exit(1);
}}

void deconnexion(int validation)
{
if (validation == 1)
{  EXEC SQL COMMIT WORK RELEASE;
} else
{  EXEC SQL ROLLBACK WORK RELEASE;}
printf("Déconnexion sans problème.\n");
}

void sql_error(char *msg)
{ char err_msg[128];
long buf_len, msg_len;

EXEC SQL WHENEVER SQLERROR CONTINUE;

printf("%s\n", msg);
buf_len = sizeof (err_msg);
sqlglm(err_msg, &buf_len, &msg_len);

if (msg_len > buf_len)
msg_len = buf_len;

printf("%.*s\n", msg_len, err_msg);
deconnexion(0);
exit(1);
}
```

MEMO

Langage PL/SQL

```
DECLARE
    variable_PL/SQL   {type SQL  |  nom_table.nom_colonne%TYPE  |  nom_table%ROWTYPE} ;
    CURSOR curseur IS SELECT ... ;
    nom_exception EXCEPTION;
BEGIN
    OPEN curseur ;
    FETCH curseur INTO  liste de variables ;
    curseur%FOUND
    CLOSE curseur ;
    RAISE nom_exception ;
    SELECT   liste_de_sélection  INTO liste de variables  FROM ... WHERE ... ORDER BY ... ;
    Variable PL/SQL  :=   ‘&variable_ SQLPlus’  ...
    IF  ... THEN ...  [ELSE ...]  END IF ;
    WHILE  ...   LOOP  ...   END LOOP ;
EXCEPTION
    WHEN nom_exception THEN ...;
    WHEN NO_DATA_FOUND THEN ... ;
    WHEN OTHERS THEN  ... :
END ;
```

Langage PRO*C

```
struct {long sqlcode;/* code resultant de l’exécution
                         =0 -> ok,
                         >0 -> ok avec un code d’état,
                         <0 -> erreur */
        struct {
                unsigned short sqlerrml;/*longueur du message*/
                char sqlerrmc[70];/*message d’erreur*/
        } sqlerrm;
        long sqlerrd[6];/* seul sqlerrd[2] est utilisé -> donne le nombre de lignes modifiées
                           UPDATE ou rajoutées par INSERT ou ramenées par un SELECT*/
        char sqlwarn[8];/*sqlwarn[0] ’W’ -> warning*/
        sqlwarn[0] = ’’ /*-> pas de warning*/
        sqlwarn[1] = ’W’/*-> troncation numérique ou char*/
        sqlwarn[2] = ’W’/*-> valeur Null est ignore */
        sqlwarn[3] = ’W’/*-> plus de champs dans SELECT que de variables pour recevoir*/
        sqlwarn[4] = ’W’/*-> toutes les lignes d’une table sont touchées (par DELETE ou
                             UPDATE par exemple)*/
        sqlwarn[5]      /* inutilisé */
        sqlwarn[6] = ’W’/*-> Oracle a dû exécuter un rollback */
        sqlwarn[7] = ’W’/*-> la donnée ramenée par un FETCH a été modifié
        depuis que la clause SELECT a été executé */
} sqlca;
:var_hote INDICATOR :indicateur
:var_hote :indicateur
```