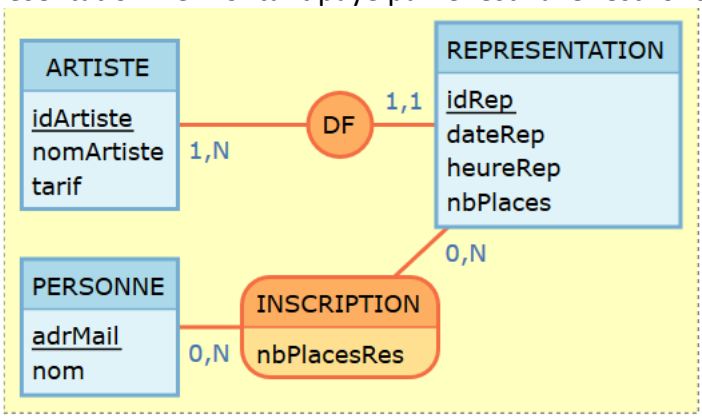
NOM : GROUPE :

DS BDD PL/SQL-PRO*C

Durée 1h30 – Aucun document n'est autorisé
NE PAS DEGRAFER LE SUJET
Ne pas utiliser de crayon de papier si possible
Vous pouvez écrire au verso des feuillets.
Des feuilles supplémentaires sont à la fin du sujet pour vous permettre de poursuivre la rédaction ou vous en servir de brouillon.
Un mémo est également en dernière page.

Application support pour les exercices:

Le festival de musique VARIATIONS accueille plusieurs artistes. Un artiste peut se produire à différents moments du festival et donc avoir plusieurs représentations. Toute personne participant au festival organise son planning comme elle le souhaite. Elle peut s'inscrire à une ou plusieurs représentations. Quand une personne s'inscrit à une représentation, elle peut réserver un certain nombre de places (nbPlacesRes) pour cette représentation. Pour chaque représentation il est défini un nombre maximum de places (nbPlaces). Une représentation concerne un artiste pour lequel il a été défini un tarif à appliquer à chaque représentation. Le montant payé par le festivalier est fonction des inscriptions faites.



ARTISTE (idArtiste, nomArtiste, tarif)
REPRESENTATION (idRep, dateRep, heureRep, nbPlaces, *idArtiste*)
PERSONNE (adrMail, nom)
INSCRIPTION (*adrMail*, *idRep*, nbPlacesRes)

Recensement des données stockées en tables:

Nom symbolique	Type	Longueur	Commentaire
idArtiste	*Caractères*	*5*	
nomArtiste	*Caractères*	*20*	*Longueur variable*
tarif	*Numérique*	*3.2*	*5 caractères dont 2 décimales*
idRep	*Caractères*	*5*	
dateRep	*date*		
heureRep	*Caractères*	*5*	*Format : 99H99*
nbPlaces	*Numérique*	*4*	*Peut être non renseigné*
nbPlacesRes	*Numérique*	*2*	
adrMail	*Caractères*	*30*	*Longueur variable*
nomArtiste	*Caractères*	*20*	*Longueur variable*

NOM : GROUPE :

Exercice 1: Langage PL/SQL (6 points)

IMPORTANT : on utilisera la table tresultat pour afficher les informations.

```
drop table tresultat purge;
create table tresultat(ligne varchar2(200));
```

La création n'est pas à écrire.

Ecrire le script PL/SQL permettant d'afficher, pour chaque représentation, le nombre total de places réservées et ceci par ordre décroissant. Seules seront affichées les représentations à venir ayant un nombre total d'inscriptions supérieur à un nombre de référence lu (vnbRef).

Indiquer le message (en utilisant une exception utilisateur): « nombre de référence trop élevé » si aucune réservation n'est trouvée.

Id représentation	**Nb total de places réservées**
-----	**-----**
....	

NOM : GROUPE :

NOM : GROUPE :

Exercice 2: Langage PRO*C (14 points)

IMPORTANT : Les fonctions connexion(), deconnexion(), sql_error(char* msg) ont été écrites et ne sont plus à écrire.

A) Question 1 (3 points)

Les représentations doivent obligatoirement avoir un nombre maximum de places. Aussi **avant de procéder aux inscriptions**, la direction souhaite mettre à jour les représentations pour lesquelles ce nombre n'est pas renseigné. La procédure devra affecter pour chacune de ces représentations un nombre maximal demandé à l'utilisateur (utilisation de la variable vnbMax).
Écrire le script Pro*C permettant de faire cette modification.

NOM : GROUPE :

NOM : GROUPE :

B) **Question 2 (5 points**)

Les inscriptions ont commencé (le script de la q°1 a donc été exécuté)**. Régulièrement la direction fait le point sur les places restantes.**
Ecrire le script Pro*C permettant d'afficher la liste des représentations, les places déjà réservées et le nombre de places disponibles pour les jours à venir. La liste est triée par dates de représentation puis heures de représentation (voir ci-après). Vous devez utiliser un curseur pour le traitement général et un tableau pour le détail de chaque représentation. Sachant que le nombre d'inscriptions par représentation n'excède pas 500.

Liste des représentations

Date Représentation *JJ/MM/YYYY*	**Heure Représentation**	**Id représentation**	**Nb places réservées/inscription**
----------	------	-----	

	Nombre de places disponibles :		----------
----------	------	-----	

	Nombre de places disponibles :		----------
…….			

NOM : GROUPE :

NOM : GROUPE :

C) **Question 3 (6 points)**

Une personne, d'identifiant vadrMail, inscrite à une représentation d'identifiant vidRepAnc, peut demander à changer cette inscription pour une nouvelle représentation d'identifiant vidRepNouv**.** Le nombre de places réservées reste le même.
Cette nouvelle inscription peut ne pas être pour le même artiste et donc ne pas être au même tarif !
Ecrire le script Pro*C permettant de réaliser cette modification**.** Les informations vadrMail, vidRepAnc et vidRepNouv sont considérées comme **déjà lues . Certains contrôles ont déjà été faits.**

Contrôles et Messages à prévoir qui arrêtent le traitement :

- Inscription déjà faite à ce nouveau choix de représentation
- Plus assez de place disponibles pour ce nouveau choix

Information:

- La modification a été correctement enregistrée. Vous devez: …… (ou on vous doit: …) euros

NOM : GROUPE :

NOM : GROUPE :

NOM : GROUPE :

NOM : GROUPE :

NOM : GROUPE :

NOM : GROUPE :

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);
}
```

NOM : GROUPE :

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 */
2020-2021 BD PLS/SQL
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
```