L3 MIAGE – 2017/2018

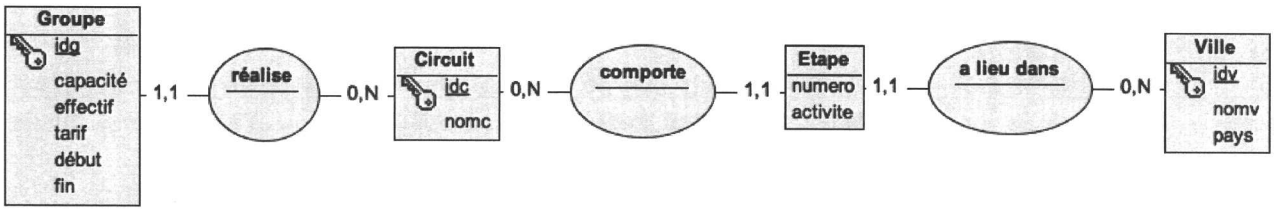
Bases de Données

le 28 mars 2018

Examen - 1ère session

durée 3h, documents autorisés, appareils mobiles de communication interdits.

Exercice 1 : Une agence de voyage décide d'informatiser son catalogue de circuits touristiques, ainsi que la gestion de ses clients. Un circuit touristique est décrit par une succession d'étapes, à raison d'une par jour. Pour simplifier cet exercice, on supposera qu'un jour se passe toujours dans une seule ville. Un circuit est effectué plusieurs fois par an par des groupes différents, à différentes périodes de l'année.
Voici un premier Modèle Conceptuel de Données pour représenter les circuits touristiques :



Et voici les tables que l'on a créées conformément à ce MCD :

1. table `Ville(idv, nomv, pays)` : l'identifiant, le nom de la ville, et le pays où elle se situe

2. table `Circuit(idc, nomc)` : l'identifiant et le nom du circuit.

3. table `Etape(idv, idc, numero, activite)` : l'identifiant de la ville où se déroule l'étape, l'identifiant du circuit dont fait partie l'étape, le numéro d'ordre de l'étape (1 si c'est le premier jour, etc) **au sein de ce circuit**, et l'activité qui est faite ce jour là. Une étape représente donc une journée d'un circuit touristique.

4. table `Groupe(idg, capacite, effectif, tarif, idc, debut, fin)` : un groupe de voyageurs qui réalise le circuit d'identifiant `idc`, entre la date `debut` et la date `fin`. Ce groupe peut accueillir `capacite` personnes, et a en réalité `effectif` inscrits (au départ, 0). Naturellement, l'effectif ne peut pas dépasser la capacité du groupe. `tarif` est le prix que coûte l'inscription à ce groupe. `idg` est l'identifiant du groupe.

Voici par exemple les données décrivant un circuit touristique en Hongrie, circuit qui va être réalisé par un groupe en avril et par un autre groupe en juin. Sur cet exemple il y a pour l'instant 3 personnes inscrites au groupe 3, et aucune au groupe 4.

```
 -- table Ville
IDV    NOMV                PAYS
----- ------------------ ------------------------------------
9      Budapest           Hongrie
10     Visegrad           Hongrie
11     Balatonfured       Hongrie
```

```
-- table Circuit
IDC    NOMC
------ ------------------------------------------------------
16     Découverte Hongrie

-- table Etape
IDV  IDC  NUMERO                ACTIVITE
---- ---- ------- ------------------------------------------------------------------------------------------
9    16     1    Envol à destination de Budapest. Accueil à l'aéroport et transfert à votre hôtel
9    16     2    Journée consacrée à la visite de Budapest
10   16     3    Excursion dans la région de la Boucle du Danube. A Visegrad, ascension au belvédère ...
11   16     4    Départ vers la région du lac Balaton. Promenade à Balatonfured, élégante station balnéair
9    16     5    Bains thermaux à Szechenyi. Croisière sur le Danube pour admirer la ville de Budapest.

-- table Groupe
IDG   CAPACITE EFFECTIF TARIF IDC    DEBUT    FIN
----- -------- -------- ----- ----- -------- --------
3       25       3      1050   16   09/04/18 13/04/18
4       20       0      1150   16   20/06/18 24/06/18
```

Question 1.1 : Donner les instructions SQL qui permettent de créer les tables `Etape` et `Groupe`, en supposant que `Circuit` et `Ville` existent déjà. Le MCD n'indique pas quelle est la clé primaire de `Etape` - ne pas oublier de la définir.

Question 1.2 : Quand on ajoute un groupe, on veut être certain que les dates `debut` et `fin` sont cohérentes avec le nombre de jours que dure le circuit. Définir un trigger `calcul_fin` qui calcule la date de fin en fonction de la date de début quand on insère ou modifie une ligne de la table `Groupe`[1]. Par exemple, le groupe 4 de la table `Groupe` doit pouvoir être inséré grâce à l'instruction :

```
insert into groupe(idg, capacite, tarif, idc, debut) values(4,20,1150,16,'20/06/2018');
```

On souhaite maintenant gérer les inscriptions des clients aux groupes de voyageurs. On définit donc une table `CLIENT` et une table `INSCRIPTION` :

- table `Client(idcl, nom, prenom, solde)` : `idcl` est l'identifiant du client. Le solde est utilisé pour le paiement/remboursement des inscriptions : par exemple un solde de -1500€ signifie que le client doit à l'agence 1500 €. A l'inverse, un solde de $+1500$€ signifie que l'agence doit rembourser le client de 1500 €.
- table `Inscription(idcl, idg)` : cette table traduit une association entre `Client` et `Groupe` et une ligne signifie que le client `idcl` s'est inscrit au groupe `idg`.

Question 1.3 : Ecrivez les requêtes SQL permettant d'obtenir :

1. les groupes qui ne sont pas complets
   SCHEMA :`(idg, idc, debut, capacite, effectif)`

[1]En SQL, si d est une date et k est un nombre, $d + k$ est la date k jour(s) après d. Par exemple si nous sommes le 28 mars 2018, $sysdate + 1$ vaut la date du 29 mars 2018.

2. les inscriptions avec pour chacune le nom et prénom du client, la date de début du voyage et le nom du circuit
   SCHEMA :`(nom, prenom, debut, nomc)`
3. les circuits dont au moins une étape a lieu en Espagne
   SCHEMA :`(idc, nomc)`
4. les circuits dont toutes les étapes sont en Espagne
   SCHEMA :`(idc, nomc)`
5. Les clients avec leur nombre d'inscriptions (0 si aucune inscription)
   SCHEMA :`(idcl, nom, prenom, nb_inscriptions)`
6. Les clients avec d'une part leur nombre d'inscriptions (0 si aucune inscription) à des groupes partis jusqu'à aujourd'hui inclus et d'autre part leur nombre d'inscriptions pour des départs qui auront lieu à partir de demain.
   SCHEMA :`(idcl, nom, prenom, nb_inscriptions_passees, nb_inscriptions_a_venir)`

   Pour cette dernière requête, vous pouvez utiliser la fonction `decode` vue en TP

   ```
   decode({valeur de test}, {valeur de comparaison}, {valeur retournée en cas de correspondance},
          [{valeur de comparaison}, {valeur retournée en cas de correspondance}, [ ... ]],
          {valeur retournée si aucune correspondance n'est trouvée}
          )
   ```

   et la fonction `sign(expr)` qui renvoie 1 (resp 0, -1) si `expr` est < 0 (resp. $= 0$, > 0),

On définit un paquetage pour gérer les inscriptions :

```
create or replace
package paq_inscription as

  PARAMETRE_INDEFINI Exception ;
  CLIENT_INCONNU Exception ;
  CLIENT_NON_INSCRIT Exception ;
  GROUPE_INCONNU Exception ;
  GROUPE_COMPLET Exception ;
  DEJA_INSCRIT Exception ;

  pragma Exception_init(PARAMETRE_INDEFINI, -20000);
  pragma Exception_init(CLIENT_INCONNU, -20001);
  pragma Exception_init(CLIENT_NON_INSCRIT, -20002);
  pragma Exception_init(GROUPE_INCONNU, -20003);
  pragma Exception_init(GROUPE_COMPLET, -20004);
  pragma Exception_init(DEJA_INSCRIT, -20005);

  procedure inscrire(le_client client.idcl%type, le_groupe groupe.idg%type);

  procedure desinscrire(le_client client.idcl%type, le_groupe groupe.idg%type);

  procedure annuler_groupe(le_groupe groupe.idg%type);

end paq_inscription;
```

Les questions suivantes concernent l'implémentation des procédures de ce paquetage. Voici, pour information, quelques codes d'erreurs SQL :

- -1 pour un problème d'unicité ; par exemple violation de clé primaire.
- -2290 pour violation de contrainte check
- -2291 pour violation de clé étrangère : par exemple si on veut mettre dans la table `Etape` une valeur de `idv` qui ne correspond à aucune clé primaire de la table `Ville`.
- -2292 si on tente de supprimer une clé référencée ; par exemple si on veut supprimer une ville alors qu'il existe une étape qui fait référence à cette ville et que l'on n'a pas prévu de clause `ON DELETE ...`

Question 1.4 : Ecrire la procedure `inscrire`, qui permet d'inscrire le client d'identifiant `le_client` au groupe d'identifiant `le_groupe`. Cette procédure ajoute une ligne dans la table `Inscription`, modifie le solde du client puisque celui-ci doit la somme correspondant au tarif du groupe, modifie le groupe en augmentant son effectif. Cette procédure déclenche `PARAMETRE_INDEFINI` si l'un des paramètre vaut `null`. Elle déclenche `CLIENT_INCONNU` (resp. `GROUPE_INCONNU`) si l'identifiant du client (resp. du groupe) n'est pas dans la base. Elle déclenche `DEJA_INSCRIT` si le client est déjà inscrit à ce groupe. Elle déclenche `GROUPE_COMPLET` si l'effectif a déjà atteint la capacité du groupe.

On suppose que la procédure `desinscrire` qui permet de désinscrire le client d'identifiant `le_client` du groupe d'identifiant `le_groupe` est implémentée. Logiquement cette procédure "défait" ce qu'a fait la procédure d'inscription : elle supprime l'inscription de la table `Inscription`, modifie le solde du client, modifie le groupe en diminuant son effectif. Cette procédure déclenche `PARAMETRE_INDEFINI` si l'un des paramètres vaut `null`. Elle déclenche `CLIENT_INCONNU` (resp. `GROUPE_INCONNU`) si l'identifiant du client (resp. du groupe) n'est pas dans la base. Elle déclenche `CLIENT_NON_INSCRIT` si le client et le groupe existent mais pas l'inscription du client à ce groupe.

Parfois, il est possible d'annuler un groupe quand on n'a pas assez d'inscrits. Dans ce cas il faut supprimer toutes les inscriptions des clients qui s'étaient inscrits au groupe.
Voici une première version de la procédure `annuler_groupe`.

```
procedure annuler_groupe(le_groupe groupe.idg%type) as
    cursor les_inscrits is
    select * from inscription
    where idg=le_groupe ;
  begin
    for un_inscrit in les_inscrits loop
      desinscrire(un_inscrit.idcl, le_groupe) ;
    end loop ;
  end annuler_groupe ;
```

Cette procédure fonctionne correctement, mais pose quelques problèmes d'efficacité. Les questions suivantes vont nous permettre de l'améliorer, en partant du constat que - à résultat équivalent - une requête SQL qui concerne k lignes est plus efficace que k requêtes qui concernent 1 ligne.

Question 1.5 : Sachant que la désinscription est très semblable à l'inscription en nombre de requêtes, estimez combien de requêtes SQL sont faites par la procédure `annuler_groupe`, si cette annulation concerne 10 clients ? Expliquez vos calculs en précisant combien de lignes sont concernées par chaque requête.

Question 1.6 : Proposez une implémentation sans curseur de la procédure `annuler_groupe`. Cette procédure a exactement la même spécification que la première version avec curseur (i.e. déclenchement des exceptions, modification en base).

Question 1.7 : Combien de requêtes SQL sont faites par votre procédure `annuler_groupe`, si cette annulation concerne 10 clients ? Comparez avec la solution avec curseur.

Exercice 2 : Dans cet exercice, on s'intéresse au scrutin proportionnel. Le principe est le suivant : chaque parti présente une liste de candidats aux électeurs, les électeurs votent pour un parti. Puis les sièges sont attribués aux différents partis proportionnellement au nombre de voix qu'ils ont obtenues (les candidats élus sont pris dans chacune des listes dans leur ordre d'apparition).
Pour expliquer comment se passe la répartition des sièges, nous prendrons pour exemple une élection où 6 sièges sont à pourvoir, sept listes en présence et où 100 000 suffrages ont été exprimés. :

1. **La barre des 5 %** :
   On élimine les listes qui n'ont pas obtenu un certain seuil de représentativité (ici, 5%).

liste A	32000 voix	32%
liste B	25000 voix	25%
liste C	16000 voix	16%
liste D	12000 voix	12%
liste E	8000 voix	8%
liste F	4500 voix	4,5%
liste G	2500 voix	2,5%

   Les deux dernières listes n'ayant pas atteint 5 % des voix n'obtiennent aucun siège (donc 0 siège pour F et G). Les **suffrages exprimés utiles** sont les voix des listes dépassant 5 % : $A + B + C + D + E = 93000$.

2. Le calcul du **quotient électoral** pour répartir les premiers sièges :
   On calcule le quotient électoral en divisant[2] le nombre de suffrages exprimés utiles par le nombre de sièges à pourvoir : $93000/6 = 15500$. On attribue ensuite à chaque liste autant de sièges que son nombre de voix contient de fois le quotient.

Liste A	2 sièges
Liste B	1 siège
Liste C	1 siège
Listes D	0 siège
Liste E	0 siège

[2]On applique la division entière.

3. La répartition des derniers sièges :
   Il reste des sièges à pourvoir (dans cet exemple 2 sièges). On attribue d'abord fictivement un siège supplémentaire à chacune des listes. On divise ensuite le nombre de suffrages recueillis par chaque liste par le nombre de sièges déjà attribués + 1. Celle qui a le plus fort résultat obtient un siège (en cas d'ex-aequo, on choisit arbitrairement une liste parmi celles qui ont le plus fort résultat).

Liste A	32000 voix	3 (2+1) sièges	32000/3 = 10666
Liste B	25000 voix	2 (1+1) sièges	25000/2 = 12500
Liste C	16000 voix	2 (1+1) sièges	16000/2 = 8000
Liste D	12000 voix	1 (0+1) siège	12000/1 = 12000
Liste E	8000 voix	1 (0+1) siège	8000/1 = 8000

   Le premier siège restant va à la liste B qui obtient le meilleur résultat. **On recommence l'opération** pour l'attribution du dernier siège qui ira à la liste D.

On dispose d'une base de données contenant la table `ELECTION(liste, nbVoix, nbSieges)`. Au départ, les deux premières colonnes sont remplies, ce qui permet de connaître les voix obtenues pour chaque liste. La colonne `nbSieges` est initialisée à 0. L'objectif de l'exercice est d'écrire un programme JDBC qui calcule la dernière colonne, c'est-à-dire le nombre de sièges de chaque liste. Voici la classe java qui va réaliser ce calcul :

```
import java.sql.* ;
import java.io.*;

public class Election {
  private Connection db ;
  ... différents statements ...

  private int barre = 5 ; // barre des 5%

  /* création objet Election, connexion à la base */
  public Election(String user, String password)
    throws SQLException {
    this.db
    = DriverManager.getConnection("jdbc:oracle:thin:@oracle.fil.univ-lille1.fr:1521:filora",user,password);
    ... Initialisation des statements ...
  }

  /* fin de connexion */
  public void fin() {
      try{       db.close() ;
      }catch(SQLException e){System.err.println("pb déconnexion : "+e.getMessage());}
  }

  /* calcul de la répartition des sièges,
     met à jour la colonne "nbSieges" de la table "Election" */
  public void calculSieges(int nbSiegesAPourvoir) throws SQLException{
    int nbSuffrages= this.getNbSuffrages() ;
    int seuil=(nbSuffrages*this.barre)/100 ;
    int nbSuffragesUtiles=this.getNbSuffragesUtiles(seuil) ;
    int quotientElectoral=nbSuffragesUtiles/nbSiegesAPourvoir ;
    this.repartitionSieges(seuil,quotientElectoral) ;
    this.repartitionResteSieges(seuil, nbSiegesAPourvoir) ;
  }
```

```
/* calcul du nombre de suffrages exprimés */
private int getNbSuffrages() throws SQLException{ ... }

/* calcul du nombre de suffrages utiles */
private int getNbSuffragesUtiles(int seuil) throws SQLException{ ... }

/* première répartition des sièges en fonction du quotient électoral */
private void repartitionSieges(int seuil, int quotientElectoral) throws SQLException{ ... }

/* répartition finale des sièges encore à pourvoir */
private void repartitionResteSieges(int seuil, int nbSiegesAPourvoir) throws SQLException{ ... }

public static void main(String args[]) {
    Election election = null ;
  try {
    election=new Election("castafiore","traviata") ; // login et mot de passe
    election.calculSieges(6) ;  //  6 sièges sont à pourvoir
  }catch(SQLException e){ ... // on traite l'exception
  }finally{ if (election != null) election.fin() ;} // quoiqu'il arrive on se déconnecte
}}
```

Pour chaque question qui suit, si vous avez besoin d'un `Statement` ou d'un `PreparedStatement`, vous indiquerez son initialisation en dehors de la méthode demandée puisque cette instruction d'initialisation figurera dans le constructeur de la classe.

Question 2.1 : Ecrire la fonction `getNbSuffrages` qui renvoie le nombre total de voix.

Question 2.2 : Ecrire la méthode `repartitionSieges` qui répartit les premiers sièges en fonction du quotient électoral. Cette méthode modifie donc la colonne `nbSieges` de la table `Election`. Le paramètre `seuil` est le nombre minimum de voix qu'il faut pour avoir au moins un siège, ce qui correspond au seuil de représentativité. Le second paramètre est le quotient électoral.

Question 2.3 : Ecrire la méthode `repartitionResteSieges` qui répartit les sièges non attribués par la méthode `repartitionSieges`. Cette méthode modifie la colonne `nbSieges` de la table `Election`. Le paramètre `seuil` est toujours le nombre minimum de voix qu'il faut pour avoir au moins un siège. Le paramètre `nbSiegesAPourvoir` est le nombre total de sièges à pourvoir (6 sur notre exemple).
Comme cette répartition se fait itérativement (1 siège à la fois), les mêmes requêtes sont appelées plusieurs fois. Vous utiliserez dans ce cas des `PreparedStatement`.