TD1 & TP1 : bibliothèque

Les relations de cette base de données sont accessibles en préfixant leur nom par biblio. Par exemple : select ... from biblio.leCatalogue.

1 Position du problème

Une association, constituée d'un ensemble d'*adhérents*, gère une bibliothèque de prêts : elle possède un certain nombre d'exemplaires (au moins un) de chaque *livre* de son *catalogue*. Une personne ne peut emprunter un livre que si toutes les règles suivantes sont respectées :

- La personne doit être adhérente de l'association.
- Le livre doit apparaître dans le catalogue et au moins un exemplaire doit être *disponible* en bibliothèque. Un livre est disponible s'il est dans le catalogue et si au moins un de ses exemplaires n'est pas en cours de prêt.
- Une personne ne peut emprunter qu'un seul exemplaire d'un même livre.
- Le nombre de livres empruntés par une personne à un instant donné ne peut pas dépasser une limite fixée qui est la même pour tous les adhérents.

La base de données que l'on considère ici gère le *catalogue* des livres et le *répertoire* des adhérents. Pour chaque livre, on dispose des informations suivantes : son titre, son auteur identifié par ses nom et prénom, et son année d'édition. Un livre est identifié par son titre. Le *fonds* de la bibliothèque est constitué d'un ensemble d'exemplaires de ces livres (au moins un exemplaire de chaque livre). Chaque exemplaire est identifié par sa cote. Pour chaque adhérent, on dispose des informations suivantes : le numéro qui l'identifie, ses nom, prénom, adresse, année de naissance et date d'adhésion. Chaque *emprunt* associe un exemplaire, un adhérent et une date de début de l'emprunt. Lorsqu'un exemplaire emprunté est rendu, l'emprunt correspondant est effacé de la base de données.

2 Schéma de relations

LeCatalogue (titre, nom, prénom, anEd)
/ <t, n, p, a> ∈ LeCatalogue ⟺ le livre de titre t a été écrit par l'auteur de nom n et de prénom p. Ce livre a été édité pour la première fois dans l'année a. */*
LeFonds (cote, titre)
/ <c, t> ∈ LeFonds ⟺ la bibliothèque possède un exemplaire identifié par c du livre de titre t. */*
LesEmprunts (cote, noAdh, datEmp)
/ <c, a, d> ∈ LesEmprunts ⟺ l'exemplaire identifié par c est emprunté par l'adhérent a depuis la date d. */*
LesAdhérents (noAdh, nom, prénom, adresse, anNais, datAdh)
/ <a, n, p, v, y, d> ∈ LesAdhérents ⟺ l'adhérent identifié par a, a pour nom n et prénom p, et habite dans la ville v. Son année de naissance est y. Il(elle) est adhérent(e) depuis la date d. */*

Remarque : sous Oracle, le nom des relations doit être préfixé par biblio..
Par exemple : select cote from biblio.LeFonds;

3 Compréhension et lecture

Question 1 :
Compléter le schéma donné dans l'énoncé par l'expression des contraintes d'intégrité mentionnées dans le texte.

Question 2 :
Les expressions suivantes sont-elles correctes ? Si non, pourquoi ? Si oui, pour chacune, donnez la spécification de la relation qu'elle construit (attribut, identifiant, prédicat).

1. select titre, nom, prénom from LeCatalogue where anEd=1959;
2. select noAdh from LesEmprunts where cote <> 10;
3. select distinct titre from LeFonds;
4. select titre, anEd from LeFonds;
5. select titre, nom, prénom from LeCatalogue;

4 Expression de requêtes

Question 3 :
Pour chaque requête ci-dessous, spécifier la relation construite par le résultat attendu, et donner son expression en SQL. Les requêtes devront construire des résultats sans répétition de valeurs. La clause distinct sera utilisée uniquement lorsque nécessaire. Les produits de relation seront exprimés dans la clause from des requêtes.

1. Titre et année d'édition des livres du catalogue.
2. Numéro, nom, prénom et adresse des adhérents nés après 1960.
3. Nom et prénom des auteurs dont la bibliothèque possède au moins un exemplaire.
4. Numéro des adhérents qui n'ont aucun emprunt.
5. Numéro des adhérents qui n'empruntent aucun des exemplaires dont la cote est 2, 3, ou 8.
6. Cote et titre des exemplaires de livres édités en 1980 ou après.
7. Numéro, nom et prénom des adhérents qui empruntent au moins un exemplaire d'un livre écrit par Franck Herbert.
8. Nom et prénom des adhérents qui sont auteurs d'au moins un livre détenu par la bibliothèque.
9. Couple de titre de livres d'auteurs différents et édités la même année.

TP1 : découvrir Oracle

L'accès au Système de Gestion de Bases de Données (SGBD) Oracle n'est possible que sur le serveur im2ag-oracle.

1 Connexion à Oracle

Se connecter à la machine im2ag-oracle. Pour cela, par exemple, à partir d'un autre serveur taper :

```
ssh -X im2ag-oracle
```

Vous devez fournir votre nom d'utilisateur et votre mot de passe (ceux que vous utilisez habituellement sur le réseau de l'université). Vous êtes sous **Unix**, il s'affiche :

```
{1}im2ag-oracle
```

Ensuite, appeler le logiciel `Oracle` en utilisant :

```
{1}im2ag-oracle sqlplus
```

Vous devez fournir votre nom d'utilisateur (le même que précédemment) et le mot de passe associé (celui que vous utilisez habituellement sur le réseau de l'université).

Vous êtes sous **Oracle**, il s'affiche :

```
SQL>
```

Pour quitter l'interprète SQL d'Oracle, utiliser `exit`, qui ramène sous Unix. Ensuite, pour sortir d'Unix, utiliser à nouveau `exit`.

Si besoin, pour travailler à la fois avec Oracle et avec un éditeur, ouvrir deux fenêtres sous im2ag-oracle; dans l'une, appeler Oracle par `sqlplus`, et, dans l'autre, appeler un éditeur par `nedit` (par exemple).

La commande `host` permet d'exécuter un shell depuis `sqlplus`. Par exemple :

```
SQL> host
> gedit &
> exit
SQL>
```

2 Exécution d'une requête en ligne

Dans l'environnement Oracle, taper la ligne suivante et analyser le résultat :

```
SQL> select titre from biblio.LeCatalogue;
```

Ensuite, taper la ligne suivante et analyser le résultat :

```
SQL> select titre from LeCatalogue;
```

Pour avoir une description des atttributs de la relation `biblio.LeCatalogue`, taper :

```
SQL> desc biblio.LeCatalogue
```

Les relations de la bibliothèque ont été créées dans le schéma `biblio`. Pour basculer vers ce schéma pendant la durée de la session `sql` en cours, il suffit de soumettre la requête : `alter session set current_schema=biblio`. La durée de la bascule est liée à celle de la session `sql`.

3 Exécution d'une requête à partir d'un fichier

La plupart du temps, on prépare les requêtes dans un fichier qu'on demande ensuite à Oracle d'exécuter, en utilisant la commande `start`. Créer un fichier de suffixe `.sql`, par exemple `Req1.sql`, puis dans ce fichier, taper le texte suivant :

```
-- le titre des livres de la bibliothèque
select titre
from biblio.LeCatalogue;
```

La ligne commençant par "`--`" est une ligne de commentaires, les autres lignes forment une requête `SQL`. Il ne faut pas placer de commentaires à l'intérieur d'une requête.

Pour exécuter cette requête, taper sous `Oracle` :

```
SQL> start Req1
```

Le système cherche alors le fichier `Req1.sql` et l'exécute. Le résultat doit être le même que lorsqu'on tape la requête en ligne. Essayer différentes variantes dans le fichier `Req1.sql`, les faire exécuter et analyser le résultat. Si `Oracle` indique des erreurs, les corriger.

Essayer aussi d'écrire plusieurs requêtes l'une après l'autre dans le même fichier.

Il est aussi possible de procéder par *copier-coller* : copier la requête rédigée avec l'éditeur puis la coller dans la fenêtre où Oracle est exécuté.

4 Stockage des résultats des requêtes dans un fichier

Pour stocker automatiquement les résultats des requêtes dans un fichier, utiliser `spool`, en insérant dans le fichier `Req1.sql` une ligne en haut et une ligne en bas :

```
spool ResReq1
... (ici vos requêtes) ...
spool off;
```

Comme précédemment, taper sous `Oracle` :

```
SQL> start Req1
```

Un fichier est créé automatiquement, de nom `ResReq1.lst` (le nom que vous avez donné, suivi du suffixe `.lst`). Ouvrir le fichier `ResReq1.lst` pour voir son contenu.

Remarque : on peut remplacer `spool ResReq1` par `spool Req1`, le système ne confond pas les fichiers `Req1.lst` et `Req1.sql` puisqu'ils n'ont pas le même suffixe.

On obtient le même comportement par copier-coller de la requête.

5 Requêtes paramétrées

Dans un fichier de nom par exemple `Req2.sql`, taper le texte suivant (où `accept` et `prompt` sont des mots-clés d'`Oracle`) :

```
-- Numéro, nom, et prénom des adhérents nés après 1960
-- et qui habitent à Grenoble.
accept lannee prompt 'Donner une annee de naissance : '
accept laville prompt 'Donner un nom de ville : '
select noAdh, nom, prenom
from biblio.LesAdherents
where adresse = '&laville' and anNais > &lannee;
```

Attention à l'utilisation des quotes ! Il faut en mettre pour `'&laville'` (chaîne de caractères) mais pas pour `&leprix` (nombre).

Taper `start Req2`. Le système imprime le texte `Donner une annee de naissance :` (sans quotes). Répondre en donnant un entier. Puis le système imprime le texte `Donner un nom de ville :` (sans quotes). Répondre en donnant une ville (sans quotes). Alors le système exécute la requête.

TD2 & TP2 : la grande bouffe

Les relations de cette base de données sont accessibles en préfixant leur nom par repas. Par exemple : select ... from repas.lesRepas.

1 Position du problème et schéma des relations

Une maîtresse de maison a constitué une base de données relationnelle sur les *amis* qu'elle invite. Elle représente, pour chaque *repas* identifié par une date, l'ensemble des *plats* qui ont été servis et pour chaque plat le *vin* qui l'accompagnait. Pour un certain nombre d'*amis*, pas forcément invités, elle connaît leurs *plats préférés*. Pour chaque plat (servi ou apprécié des amis) elle associe son type.

Cette maîtresse de maison fort avisée définit le schéma de relations suivant (les identifiants des relations sont les attributs soulignés) :

LesRepas (dateR, nomI)
/ (d, i) ∈ LesRepas ⟺ la personne de nom i, a été invitée au repas identifié par la date d */*

LeMenu (dateR, nomP, nomV)
/ (d, p, v) ∈ LeMenu ⟺ le plat p accompagné par le vin v a été servi au repas d */*

LesPréférences (nomA, nomP)
/ (n, p) ∈ LesPréférences ⟺ la personne de nom n aime le plat de nom p */*

LesPlats (nomP, typeP)
/ (p, t) ∈ LesPlats ⟺ le plat p est du type t */*

domaine (dateR) = date
domaine (nomI) = domaine (nomP) = domaine (nomV) =
domaine (typeP) = domaine (nomA) = chaînes de caractères

2 Compréhension et lecture

Question 1 :
Compléter le schéma donné dans l'énoncé par l'expression des contraintes d'intégrité référentielle.

Question 2 :
Les expressions suivantes sont-elles correctes ? Si non, pourquoi ? Si oui, pour chacune, donnez la spécification de la relation qu'elle construit (attribut, identifiant, prédicat).

1. select nomI from LesPréférences where nomP = 'mousse';
2. select distinct nomI from LesRepas where to_char(dateR, 'DD-MM-YYYY') = '31-12-2004';
3. select * from LesRepas R join LeMenu M on (R.dateR=M.dateR) where nomP = 'Ile flottante';
4. select * from LesRepas natural join LeMenu where nomP = 'Ile flottante';

2.1 Expression de requêtes

Question 3 :
Pour chaque requête ci-dessous, spécifier la relation construite par le résultat attendu, et donner son expression en SQL. Les requêtes devront construire des résultats sans répétition de valeurs. La clause distinct sera utilisée uniquement lorsque nécessaire. Les produits de relation seront exprimés dans la clause from des requêtes.

Sous Oracle, le nom des relations doit être préfixé par repas.. Par exemple : select * from repas.LesPlats;

1. Donner l'ensemble des vins qui ont été servis avec un médaillon de langouste.

2. Donner les plats (avec le vin qui les accompagne) qui ont été servis le 21 octobre 2003.

3. Donner le nom des amis qui n'ont jamais été invités.

4. Donner la liste des repas (date, plats et vins servis) auxquels Jacques et Thomas ont été invités (ensemble).

5. Donner la liste des amis qui ont eu, au moins une fois, un plat de leurs préférences.

6. Donner les noms des personnes qui ont été invitées au moins 2 fois.
   Indication : ne pas utiliser les opérateurs de partition et d'agrégation de SQL.

7. Donner les noms des personnes qui ont été invitées exactement 2 fois.
   Indication : ne pas utiliser les opérateurs de partition et d'agrégation de SQL.

8. Donner l'ensemble des invités qui n'aiment que les desserts.

9. Donner l'ensemble des amis qui n'ont jamais eu un de leurs plats préférés.

10. Donner l'ensemble des invités qui ont mangé du Foie gras, alors que ce plat ne fait pas partie de leurs préférences.

11. Donner le menu et les invités du repas le plus récent.
    Indication : ne pas utiliser les opérateurs de partition et d'agrégation de SQL.

12. Donner les invités à tous les repas
    Indication : ne pas utiliser les opérateurs de partition et d'agrégation de SQL.

TD3 & TP3 : la grande bouffe – exercices de synthèse

1 Position du problème et schéma des relations

On reprend le sujet "Repas" déjà étudié.

2 Expression de requêtes en SQL

1. Pour la personne invitée le plus souvent donner son nom ainsi que ses plats préférés.

2. A chaque date où un repas est organisé, donner l'invité qui apprécie le plus grand nombre de plats.

3. A chaque date où un repas est organisé donner le nombre d'invités chanceux. Un invité chanceux est un invité à qui est servi au moins un des plats de ses préférences.

4. Donner les noms des amis qui aiment tous les types de plats, c'est-à-dire au moins un plat de chaque type.

5. Donner pour chaque mois où au moins repas a été organisé, le nombre d'invités.

6. Donner pour chaque plat, le vin qui l'accompagne le plus souvent, et le nombre de fois que ce vin accompagne ce plat.

7. Donner pour chaque personne (amie et/ou invitée), son nom, le nombre de fois qu'elle a été invitée, le nombre de plats qu'elle apprécie, et le nombre de types de plats qu'elle apprécie.

TD4 & TP4, TP5 : agence de voyages

Les relations de cette base de données sont accessibles en préfixant leur nom par agence. Par exemple : select ... from agence.lesCircuits.

1 Position du problème

Une agence de voyage a informatisé la gestion des voyages qu'elle propose (itinéraires, monuments visités, réservations, etc.). La base de données a été construite à partir du cahier des charges suivant :

les circuits :

Un circuit est identifié par un numéro, il est décrit par une ville de départ, une ville d'arrivée et une séquence d'étapes. Une étape se déroule pendant un nombre donné de jours, dans une ville donnée. Au cours de chaque étape, tous les monuments de la ville, lorsqu'il y en a, sont visités. Les villes de départ et d'arrivée n'étant pas considérées comme des étapes, leurs monuments ne sont pas visités.

Un même circuit ne repasse jamais plusieurs fois dans la même ville étape, mais il peut arriver qu'une ville départ ou arrivée figure aussi dans l'ensemble des villes étapes. Ceci permet de prendre en compte les situations où les villes de départ et/ou d'arrivée font l'objet d'une visite. Les villes sont identifiées par leur nom. Les monuments sont identifiés par leur nom, dans la ville où ils sont situés.

Un circuit peut être programmé plusieurs fois, à des dates différentes. A chacune de ces programmations, on associe un nombre de places proposées (c'est-à-dire le nombre maximal de places qu'il est possible de réserver). Deux programmations d'un même circuit peuvent avoir des nombres différents de places proposées. Par contre, le prix d'un circuit est fixé, toujours le même quelque soit sa programmation. Un circuit dure un nombre de jours égal à la somme des durées de chacune de ses étapes.

les réservations :

Une réservation, identifiée par un numéro, est effectuée pour le compte d'un client (identifié par son nom) et concerne une programmation d'un circuit. Plusieurs places pour la même programmation du même circuit peuvent être réservées en une seule fois.

Pour chaque programmation, le nombre total de places réservées calculé sur toutes les réservations de cette programmation est inférieur ou égal au nombre de places offertes de la programmation.

2 Schéma des relations

Le schéma de la base de données est donné ci-dessous (les identifiants sont soulignés) :

LesVilles (nomV, pays)
/ (n, p) ∈ LesVilles ⟺ la ville dont le nom est n, est située dans le pays p. Le nom de la ville est un identifiant (clef de la relation). */*
LesMonuments (nomM, nomV, prix)
/ (nm, nv, p) ∈ LesMonuments ⟺ le monument de nom nm est situé dans la ville nv. Son prix de visite est p euros. */*
LesCircuits (numC, vDep, vArr, prix)
/ (n, vd, va, pr) ∈ LesCircuits ⟺ le circuit touristique identifié par le numéro n, part de la ville vd et se termine dans la ville va. Son prix est de pr, qui ne prend pas en compte le prix des monuments visités. La ville de départ représente le point de rendez-vous avec les accompagnateurs. */*
LesEtapes (numC, rang, vEtape, nbJours)

/ (n, r, ve, nbj) ∈ LesEtapes ⟺ la r^e étape du circuit n se déroule dans la ville ve, où le séjour est de nbj jours. On fait comme hypothèse que lorsqu'une ville est dans un circuit, tous ses monuments sont visités. r ≥ 1. Les villes de départ et d'arrivée (vArr et vDép de LesCircuits) sont dans LesEtapes lorsqu'elles sont visitées. */*

LesProgrammations (numC, dateDep, nbPlaces)

/ (n, d, nbl) ∈ LesProgrammations ⟺ le circuit identifié par le numéro n, programmé à la date d est offert avec nbl places. Le même circuit peut être programmé à différentes dates. */*

LesReservations (numR, nomC, numC, dateDep, nbRes)

/ (nr, no, nc, d, nbr) ∈ LesRéservations ⟺ le client de nom no, a effectué une réservation identifiée par nr, sur le circuit nc programmé à la date d. Il a réservé nbr places. */*

Les domaines associés sont :

domaine (nomC) = {'Bonemine', 'Corto', etc.}
domaine (vEtape) = domaine (vDep) = domaine (vArr)
= domaine (nomV) = {'Paris', 'Florence', 'Briançon', etc.}
domaine (pays) = {'Italie', 'Finlande', 'France', etc.}
domaine (numC) = domaine (numR) = domaine (nbRes) = domaine (rang)
= domaine (nbJours) = domaine (prix) = domaine (nbPlaces) = entiers > 0
domaine (dateDep) = instant (unité jour)

Les contraintes d'intégrité référentielle sont :

LesProgrammations[numC] = LesCircuits[numC] LesEtapes[numC] ⊆ LesCircuits[numC]
LesEtapes[vEtape] ⊆ LesVilles[nomV] LesCircuits[vDep] ⊆ LesVilles[nomV]
LesCircuits[vArr] ⊆ LesVilles[nomV] LesMonuments[nomV] ⊆ LesVilles[nomV]
LesReservations[numC, dateDep] ⊆ LesProgrammations[numC, dateDep]

Remarques :

- On dit qu'un circuit passe par une ville v, lorsque v est une de ses étapes, ou/et la ville arrivée, ou/et la ville de départ ;
- On dit qu'un circuit visite une ville lorsque celle-ci est une étape de ce circuit.
- On généralise ces définitions aux pays : un circuit passe dans un pays p lorsque l'une de ses étapes, et/ou sa ville arrivée, ou/et sa ville de départ sont situées dans p ; un circuit visite un pays lorsque celui-ci a une fois au moins une étape dans ce pays.

3 Expressions de requêtes

Les requêtes devront construire des résultats ordonnés et sans répétition de valeurs, la clause distinct ne sera utilisée que lorsque nécessaire. Les produits de relation seront exprimés dans la clause from des requêtes.

- Sous Oracle, les relations sont accessibles via le préfixe agence.
- Pour comparer des valeurs de type date, on utilisera les opérateurs de comparaison habituels (<, >, =, <>, ≤, ≥). On dispose de plus des fonctions to_date et to_char fournies par Oracle :

format: {'DD/MM/YYYY', 'DD-Month-YYYY', 'DD-Month-YYYY HH24:MI', 'HH24:MI:SS', etc. }
/ Voir la liste des formats supportés par Oracle. */*
to_date: une chaîne de caractères, un format ⟶ une date
/ to_date (c, f) est la date déduite de la forme externe c, donnée sous forme de chaîne de caractères, selon le format f. */*
to_char: une date, un format ⟶ une chaîne de caractères
/ to_char (d, f) est la forme externe de la date d selon le format f. */*

- Afin de faciliter la lecture des résultats utiliser les instructions illustrée ci-dessous. La portée de chacune de ces instructions est limitée à la session.

```
SQL> set pagesize 200                    /* affichage 200 lignes avant de répéter les entêtes. */
SQL> set linesize 260                    /* 260 caractères par ligne, au delà afficher sur plusieurs lignes. */
SQL> col nomV format a20
   /* consacrer 20 caractères pour l'affichage des valeurs de l'attribut nomV (de type chaîne). */
SQL> col numC format 999
   /* consacrer 3 caractères pour l'affichage de des valeurs de l'attribut numC (de type numérique). */
```

1. Donner le numéro, les villes de départ et d'arrivée des circuits qui démarrent après une date donnée.

2. Donner le nom des clients qui ne visitent aucun monument.

3. Donner le numéro, le nombre total de jours, et le prix de base des circuits qui n'ont aucune réservation.

4. Donner les noms des villes visitées par Milou (c'est-à-dire les villes étapes).

5. Donner les noms des villes et des monuments visités par Milou.

6. Donner les noms des villes visitées par Milou avec le nombre de monuments dans chaque ville.

7. Pour chacune des villes visitées par Milou, donner le prix total pour visiter tous les monuments de cette ville.

8. Donner le nom des clients qui ont réservé au moins un circuit qui passe par un pays donné.

9. Donner le nom des pays, des villes et des monuments visités par un client donné par son nom.

10. Donner le numéro des circuits qui passent dans toutes les villes d'un pays donné. **Instruction pour la requête 10 :** la requête ne doit pas utiliser l'opérateur in ni sa négation not in

11. Donner le numéro, le prix de base (sans tenir compte du prix des monuments visités), la date de départ et le nombre de places disponibles des programmations de circuits qui ont encore des places disponibles et dont le nombre de jours est inférieur ou égal à un entier donné.

12. Pour chaque circuit restant dans le même pays, donner ce pays et le numéro du circuit. **Instruction pour la requête 12 :** la requête ne doit pas utiliser l'opérateur in ni sa négation not in

13. Pour chaque programmation de circuit, retrouver les pays dans lesquels passe le circuit et la date à laquelle le circuit arrive dans ce pays.

    Il s'agit d'écrire une requête qui construit la relation R (numC, dateDep, pays, dateEntree) telle que : $<c, d, p, e> \in R \iff$ *la programmation de la date d du circuit c arrive dans le pays p à la date e.*

    Par exemple, le résultat attendu est (limité au circuit de numéro 13 programmé le 31 décembre 2009, et au circuit 14) :

```
      NUMC DATEDEP   PAYS                 DATE_ENTR
---------- --------- -------------------- ---------
...
        13 31-DEC-09 Danemark             31-DEC-09
        13 31-DEC-09 France               31-DEC-09
        13 31-DEC-09 Groenland            04-JAN-10
        13 31-DEC-09 France               18-JAN-10
...
        14 06-JUL-10 Islande              06-JUL-10
        14 26-JUL-10 Islande              26-JUL-10

...
```

14. Etablir un récapitulatif faisant apparaître, pour chaque circuit : son numéro, son coût total et son taux moyen de remplissage.

    Remarques :

    - Le taux moyen de remplissage d'un circuit est la moyenne des taux de remplissage de ses programmations.
    - Le taux de remplissage d'une programmation est le rapport du nombre total de places réservées pour cette programmation, par le nombre de places offertes. Ce taux est 0 pour les programmations sans réservation.

4 Instructions pour le compte-rendu du TP Agence

Le TP1 doit être réalisé en binôme, rendre un compte-rendu par binôme. Le compte-rendu est à rendre avant le 18/3/2017 à 17H (http://imag-moodle.e.ujf-grenoble.fr) délai de rigueur (aucun compte-rendu ne sera accepté après cette date). Il faut utiliser pour celà la plate-forme moodle accessible à l'url https://im2ag-moodle.e.ujf-grenoble.fr. Une inscription sur ce site est requise. Utiliser le bouton Créer un compte pour créer un compte. L'application envoie un mail à l'adresse précisée lors de l'inscription, les informations de validation sont indiquées dans ce mail. Il suffit ensuite de choisir le cours INF245 du L2 Informatique du DLST.

Les seules requêtes considérées sont les requêtes 10, 11, 13 et 14 parmi celles étudiées dans la section 3.

Le compte-rendu comprendra, pour chaque requête, les 3 points suivants :

A - La spécification de la relation construite par la requête.

B - La requête en SQL.

C - Des tests de la requête afin d'illustrer son exécution sur des cas limites.

Si besoin, la requête peut être décomposée en sous-requêtes, alors tous ces points seront traités pour chacune des sous-requêtes. De même, si les tests utilisent des requêtes auxiliaires, alors tous ces points seront traités pour chacune des requêtes auxiliaires.

Remarques :

- Afin de ne programmer aucune constante les requêtes devront être paramétrées.
  Par exemple, la requête ci-dessous est paramétrée par le numéro du circuit et la ville de départ :

  ```
  select * from agence.LesCircuits where numC = &num and vdep = '&villed' ;
     /* noter les ' autour de la variable de type chaîne de caractères. */
  ```

  L'usage des requêtes paramétrées peut être facilité en utilisant les instructions prompt et accept proposée par SQL (voir la documentation en ligne : help prompt et help accept).
- Pour la présentation des jeux d'essai, les formats d'affichage seront correctement définis (instructions col, set linesize, set pagesize, etc.).

TD5 & TD6, TP6 : à propos de transports

Les relations de ce sujet sont accessibles sur Oracle en préfixant leur nom par Camions.

1 Position du problème

Une entreprise de transports possède une flotte de camions. Les camions sont utilisés pour des transports décrits par des contrats passés avec des clients.

Le personnel est composé de personnes qui sont chacune, soit secrétaire soit chauffeur. Une personne est caractérisée par un identifiant, un nom, un prénom et un salaire. Une personne secrétaire est responsable des contrats d'un ensemble de clients (au moins un). Les chauffeurs sont affectés à la conduite de camions pour les transports associés aux contrats. Il peut exister des chauffeurs affectés à aucun contrat, mais chaque secrétaire est responsable d'au moins un contrat. On prend pour hypothèse simplificatrice qu'à l'issue d'un contrat, un chauffeur est immédiatement disponible pour un autre contrat. Un chauffeur ne peut pas effectuer plus d'un contrat à la même date. Pour chaque chauffeur on stocke le nombre total de kilomètres qu'il a effectués, ou qu'il est prévu qu'il effectue.

Les clients sont caractérisés chacun par un identifiant, un nom et une adresse. Tous les contrats d'un même client sont sous la responsabilité de la même personne, secrétaire de l'entreprise. Un contrat modélise le transport de marchandises d'une ville à une autre, avec une date de départ et une date d'arrivée prévues. On prend pour hypothèse que les marchandises concernées par un contrat peuvent toutes être embarquées à bord d'un seul camion, et que les dates prévues sont respectées. On garde dans la base de données des clients, même s'ils ne sont associés à aucun contrat.

Les camions sont identifiés par leur immatriculation. On modélise de plus la marque et la date d'achat de chaque camion. On considère qu'à l'issue d'un transport, un camion est immédiatement disponible pour un autre transport. Il peut exister des camions qui ne sont affectés à aucun transport. Un camion ne peut pas être affecté à plus d'un contrat à la même date.

1.1 Schéma de relations

Le schéma retenu pour la base de données est constitué des relations suivantes (les clefs primaires sont soulignées) :

LesPersonnes (noP, nom, prénom, salaire)
/ <p, no, pr, s> ∈ LesPersonnes ⟺ la personne identifiée par p a pour nom no, et prénom pr. Elle perçoit un salaire s. */*
LesClients (noC, nom, adresse)
/ <c, n, a> ∈ LesClients ⟺ le client identifié par c a pour nom n et réside à l'adresse a. */*
LesCamions (immat, marque, dateAchat)
/ <i, m, d> ∈ LesCamions ⟺ le camion d'immatriculation i est de marque m. Il a été acheté par l'entreprise à la date d. */*
LesContrats (noTr, dateDep, dateArr, vDep, vArr, noC, immat, noP)
/ <t, dd, da, vd, va, c, i, p> ∈ LesContrats ⟺ le contrat identifié par t est établi pour le compte du client c. Il s'agit du transport depuis la ville vd vers la ville va. Les dates de départ et d'arrivée prévues sont respectivement da et dd. Le camion affecté à ce contrat est celui d'immatriculation i, conduit par le chauffeur p. */*
LesSecrétaires (noP, noC)
/ <p, c> ∈ LesSecrétaires ⟺ la personne secrétaire d'identifiant p est responsable des contrats du client d'identifiant c. */*
LesChauffeurs (noP, nbK)

/ <p, k> ∈ LesChauffeurs ⟺ il est prévu que le chauffeur identifié par p effectue k kms pour le compte de l'entreprise. k vaut 0 pour les chauffeurs affectés à aucun contrat. */*
LesDistances (vDép, vArr, nbK)
/ <d, a, k> ∈ LesDistances ⟺ k est la distance en km entre la ville d et la ville a. */*
Les domaines sont :
domaine (noP) = domaine (noTr) = domaine (noC) = entiers > 0
domaine (nom) = domaine (prénom) = domaine (immat) = domaine (marque) = chaînes de caractères
domaine (dateDep) = domaine (dateArr) = domaine (dateAchat) = date (à l'unité Jour)
domaine (adresse) = domaine (vDep) = domaine (vArr) = {"Grenoble", "Paris", "Pau", etc.}
domaine (nbK) = entiers ≥ 0, domaine (salaire) = réels > 0

Les contraintes d'intégrité :

LesPersonnes[noP] = LesSecrétaires[noP] ∪ LesChauffeurs[noP]
LesSecrétaires[noP] ∩ LesChauffeurs[noP] = ∅
LesSecrétaires[noC] ⊆ LesClients[noC] LesContrats[noC] ⊆ LesClients[noC]
LesContrats[immat] ⊆ LesCamions[immat] LesContrats[noP] ⊆ LesChauffeurs[noP]
LesContrats[vdep, varr] ⊆ LesDistances[vdep, varr]

Puis d'autres :

- La relation LesDistances est symétrique, c'est-à-dire :
  <v1, v2> ∈ LesDistances ⟺<v2, v1> ∈ LesDistances.
- Pour chaque chauffeur, le nombre de kilomètres qui lui est associé dans la relation LesChauffeurs est égal à la somme des distances effectuées sur tous ses contrats (passés et prévus). Ce nombre est 0 pour les chauffeurs affectés à aucun contrat.
- La date de départ d'un transport est inférieure ou égale à la date d'arrivée.
- La date d'achat d'un camion est inférieure ou égale aux dates de départ des transports auxquels il est affecté.
- Chaque jour un camion est affecté à au plus un contrat.
- Chaque jour un chauffeur est affecté à au plus un contrat.

2 Expression de requêtes

Question 1 :
Donner en SQL l'expression des requêtes suivantes.

Les requêtes devront construire des résultats sans répétition de valeurs. La clause distinct ne sera utilisée que lorsque nécessaire. Toute expression intermédiaire construite par une sous-requête devra être spécifiée avec précision. Les produits de relation seront exprimés dans la clause from des requêtes.

1. Donner pour chaque chauffeur, son numéro et les numéros des contrats auxquels il est affecté.

2. Compléter la requête ci-dessus afin de fournir aussi l'immatriculation et la marque du camion affecté au contrat.

3. Pour chaque secrétaire, donner son numéro et le nombre de clients dont elle s'occupe.

4. Compléter la requête ci-dessus pour retourner aussi le nom et le prénom des secrétaires.

5. Pour chaque chauffeur, donner son numéro, son nom, son prénom, son salaire et le nombre de camions qu'il a conduits ou qu'il est prévu qu'il conduise.

6. Ecrire la requête qui permet de retrouver les nuplets de LesChauffeurs qui ne respectent pas la contrainte suivante : *pour chaque chauffeur, le nombre de kilomètres qui lui est associé dans la relation LesChauffeurs est égal à la somme des distances effectuées sur tous ses contrats (passés et prévus).*

7. Donner le numéro, le nom et le prénom des chauffeurs qui ont conduit tous les camions.

8. Donner le numéro, le nom et le prénom des chauffeurs qui ont été affectés à tous les contrats du client de numéro 4.

9. Exprimer en SQL la requête qui permet de tester si des données violent la contrainte d'intégrité : *chaque jour un camion est affecté à au plus un contrat.*

10. Pour chaque chauffeur donner son numéro et l'immatriculation du camion avec lequel il a effectué (ou va effectuer) le plus de kilomètres.

11. Pour chaque chauffeur, donner son numéro, le nombre de kilomètres qu'il a effectués ou qu'il est prévu qu'il effectue, pour le compte de l'entreprise, et pour chaque contrat auquel il est affecté jusqu'à aujourd'hui, la durée du voyage et la valeur cumulée des durées de tous les voyages précédents (la durée d'un voyage est le nombre de jour séparant la date de départ de la date d'arrivée). Le résultat peut être modélisé par la relation R (X, Y, Z, T, U) telle que $<a, b, c, d, e> \in R \iff$ *le chauffeur de numéro a a effectué b kilomètres, d est la durée d'un contrat c auquel il a été affecté, e est la durée totale de tous les contrats précédents c (c exclus) et auxquels a était affecté.*

    Indication : en SQL, la différence de deux dates est une durée en jours.

TP7 : quelques exercices de programmation HTML/PhP

Le serveur `goedel.e.ujf-grenoble.fr` donne accès aux programmes PHP. Par exemple, l'accès à `http://goedel.e.ujf-grenoble.fr/~totoche` (avec n'importe quel navigateur comme `firefox`) exécutera le script `index.php` présent dans le répertoire `public_html` de l'utilisateur `totoche`. Si ce répertoire ne contient aucun programme PHP appelé `index.php`, le contenu du répertoire est alors affiché.

Question 1 :
Créer deux fichiers, index.php et index_action.php tels que (voir Figure 1) :

- index.php contient un formulaire demandant un login et un mot de passe, et envoie les données saisies à index_action.php.
- index_action.php affiche les données saisies dans le formulaire précédent

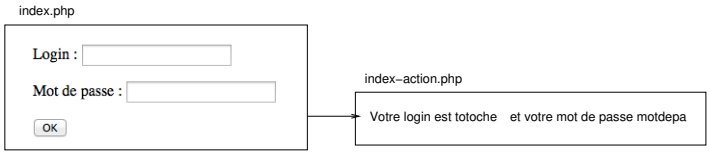

Figure 1: Un exemple de formulaire

Question 2 :
Que se passe-t-il si on valide le formulaire sans rien saisir ? Ou si on accède directement à index_action.php sans passer par le formulaire ? Trouver une façon de corriger ce problème (en affichant un message d'erreur par exemple).

Question 3 :
Nous avons récupéré le login et le mot de passe d'une personne, nous souhaitons maintenant demander son adresse email. Ajoutez un formulaire dans index_action.php, qui demande son adresse email à l'utilisateur. Ces informations sont envoyées dans index_action2.php, qui doit afficher le login, le mot de passe et l'adresse email (voir Figure 2).

Attention : on ne doit pas redemander la saisie du login ou du mot de passe dans index_action.php !

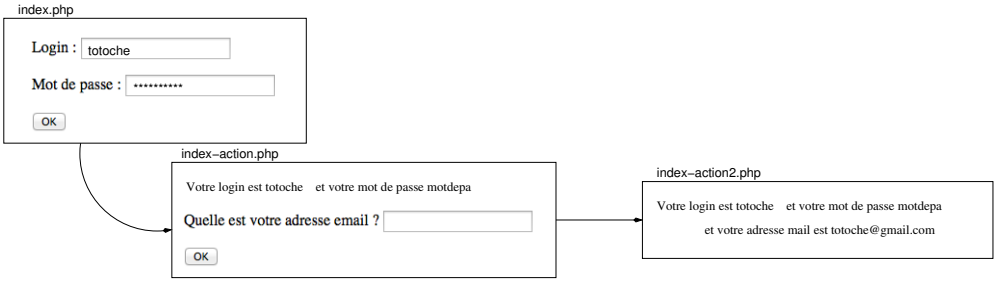

Figure 2: Un exemple de session

Question 4 :
Créer un fichier index2.php, dont le code est strictement identique à index.php.

Modifiez index.php et index2.php de manière à ce que (voir Figure 3) :

- index.php envoie ses données en post à index_action.php
- index2.php envoie ses données en get à index_action.php

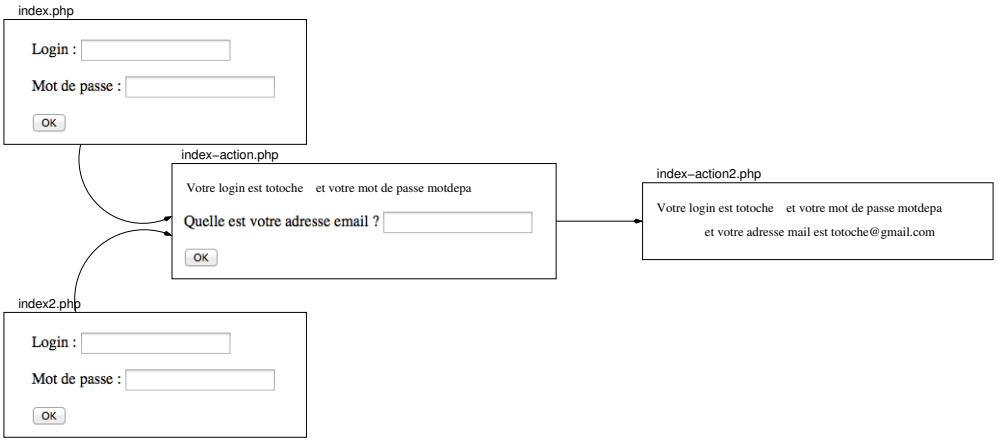

Figure 3: Méthodes get et post

Faites en sorte que index_action.php puisse récupérer les données dans deux cas, et modifiez vos programmes de manière à ce que index_action2.php affiche par quel formulaire vous êtes passé pour y arriver (index ou index2).