Examen final

Master 1 Ingénierie Mathématique

Bases de données relationnelles

Benjamin Auder

26 mars 2014

Exercice 0 [Environ 0% des points]

PostgreSQL et PHP ont un point commun de taille. Quel est-il ?

Solution :
Ils ont tous les deux un éléphant sur leur logo.

Exercice 1 [Environ 25% des points]

Cet exercice a pour objectif la construction d'une base de données modélisant l'organisation d'une piscine, selon la description donnée ci-après.

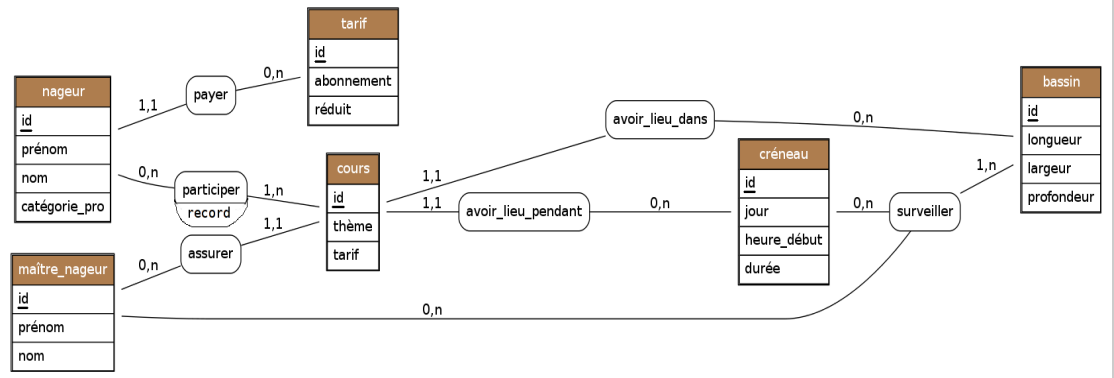
— Il y plusieurs bassins de tailles et profondeurs différentes.
— Un bassin est ouvert au public et/ou pour des cours pendant certains créneaux horaires ; les horaires d'ouverture varient au fil de la semaine.
— Un bassin ouvert est toujours surveillé par un maître-nageur.
— Un maître-nageur peut donner des cours, éventuellement à thème.
— Un cours − hebdomadaire − est donné dans un certain bassin à un certain créneau.
— Un nageur paye son entrée à un tarif variant en fonction de sa situation professionnelle, et pouvant être réduit par un abonnement de 10 séances.
— Un nageur participant à un cours thématique est susceptible d'avoir un record enregistré (par exemple 5min05 au 400m 4 nages).

Exemples de requêtes devant pouvoir être effectuées sur la base :
— nombre de nageurs inscrits au cours donné par Nathalie le mercredi à 18h ;
— nom du surveillant du bassin de 50m le dimanche matin ;
— dépenses totales pour les cours du nageur numéro 32.

(a) Dessinez et commentez un schéma entités/associations correspondant à cette situation. Ensuite, normalisez le schéma comme indiqué au second cours vers le slide 20 ; les étapes ne sont pas toutes cruciales ou applicables, mais vous devriez au moins :
— utiliser des noms communs pour les entités et leurs attributs, et des verbes (éventuellement au passif et/ou avec une préposition) pour les associations ;
— souligner les attributs identifiants ;
— indiquer les cardinalités.
Le schéma peut être dessiné sur la copie d'examen, ou conçu via un logiciel.
Suggestion 1 : http://code.google.com/p/merisier/wiki/Usage. Il est installé sur la machine 192.168.31.236, mais pas complètement au point : les diagrammes nécessitent quelques ajustements manuels.
Suggestion 2 : https://www.draw.io/# (moins spécialisé ; choisir "entity relationship" dans le menu à gauche).

Solution :

FIGURE 1 – Modèle conceptuel des données



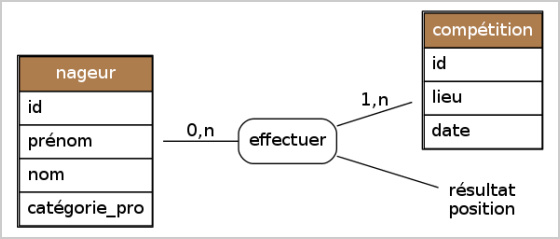
Un cours est assuré par exactement un maître-nageur, et a lieu pendant un créneau donné dans un bassin donné : d'où les cardinalités 1,1 partant de l'entité "Cours". Une surveillance de bassin met en association exactement un maître-nageur, un créneau et un bassin : l'association "surveiller" est donc ternaire. Un nageur paye un seul tarif, mais ce dernier peut être payé par de nombreux nageurs (par exemple, tous les étudiants) ou par personne (s'il n'y a pas de chômeurs par exemple). D'où les cardinalités 1,1 et 0,n.

(b) Comment étendre la modélisation pour tenir compte d'éventuelles compétitions ? Celles-ci se définissent par leurs lieu et date, et on veut mémoriser les résultats des membres du club ; avec les éventuelles médailles le cas échéant. On ne prendra pas en compte cette extension pour la question suivante.

Solution :

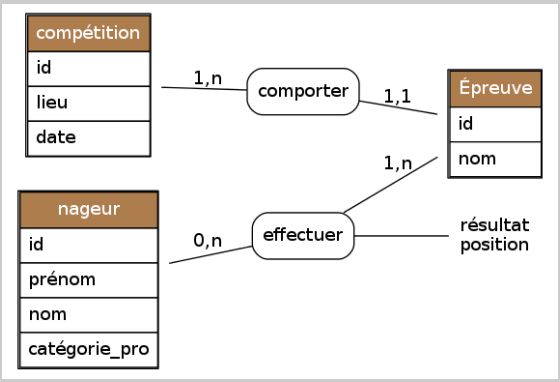
La manière la plus simple de prendre les compétitions en compte est d'ajouter une entité Compétition, et de la connecter aux nageurs via une association "effectuer". Ensuite, si on considère qu'une compétition consiste en une unique épreuve (par exemple 100m nage libre) il suffit d'ajouter le temps et le classement en attributs d'association comme suit.

FIGURE 2 – Une seule épreuve par compétition



Si en revanche on considère qu'un nageur participe à plusieurs épreuves par compétition, alors on peut garder le même schéma en renommant Compétition en Épreuve. cette dernière entité est en association avec Compétition (qui peut donc comporter plusieurs épreuves).

FIGURE 3 – Plusieurs épreuves par compétition

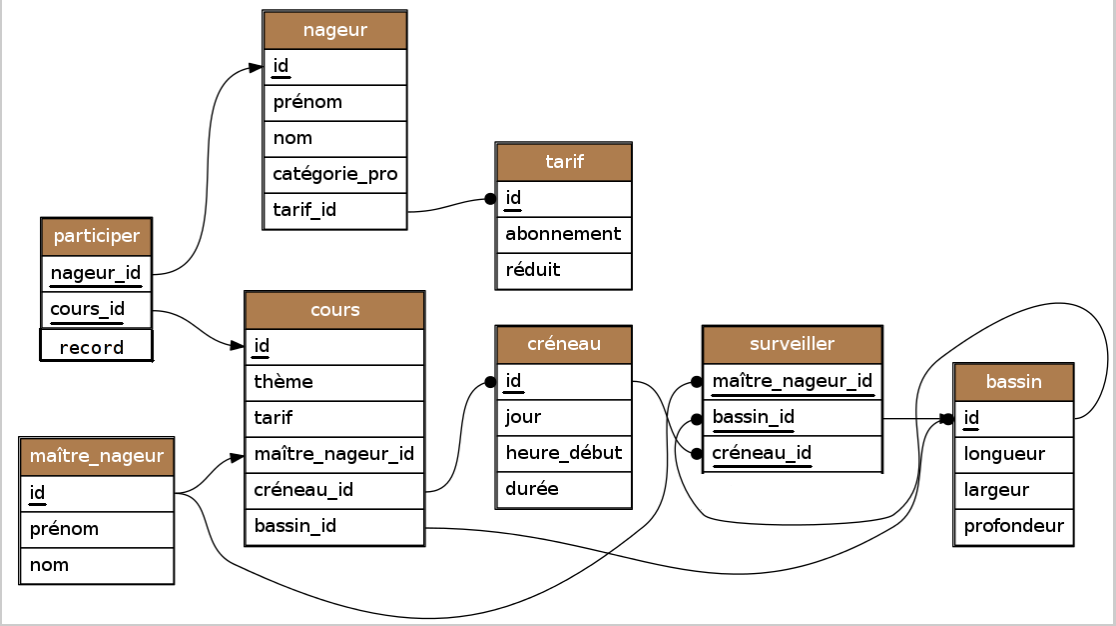


(c) Indiquez et justifiez les transformations à effectuer pour obtenir un schéma relationnel. Vous pouvez éventuellement détailler certaines parties du schéma si cela semble pertinent. Expliquez ensuite ensuite comment obtenir une instruction CREATE TABLE à partir d'une table dans le schéma relationnel. Appliquez cet "algorithme" de transformation à une table qui en référence d'autres.

Solution :

L'association ternaire devient une table "Surveillance" référençant les trois clés primaires des entités en associations. Le tarif est de cardinalité 1,1, donc intégré comme clé étrangère dans l'entité nageur. Les cardinalités de Créneau, Bassin et Maître-nageur étant 1,1 par rapport au Cours, la table correspondant à cette dernière entité référence les clés des trois entités en association.

FIGURE 4 – Modèle physique des données



Pour passer du schéma relationnel à l'implémentation dans PostgreSQL (les instructions CREATE TABLE) il suffit de déterminer le type de chaque attribut, et de ne pas oublier les mots-clés REFERENCES table(colonne) pour les clés étrangères

et PRIMARY KEY pour les clés primaires.

```
-- Instructions CREATE TABLE pour les tables Nageur et Cours
CREATE TABLE nageur (
    id serial PRIMARY KEY,
    prénom varchar(32) not null,
    nom varchar(32) not null,
    catégorie_pro integer not null default 0,
    tarif_id integer not null REFERENCES tarif(id)
);
CREATE TABLE cours (
    id serial PRIMARY KEY,
    thème varchar(32),
    tarif real not null,
    maître_nageur_id integer not null REFERENCES maître_nageur(id),
    créneau_id integer not null REFERENCES créneau(id),
    bassin_id integer not null REFERENCES bassin(id)
);
```

Exercice 2 [Environ 25% des points]

On considère les schémas de tables ci-dessous, représentant des pages web interconnectées appartenant à des sites. Les attributs auteur dans Page et webmaster dans Site référencent l'identifiant de la relation Rédacteur. Les attributs page1 et page2 dans la relation Lien référencent l'attribut chemin dans une Page, et site1 site2 sont les sites de départ et d'arrivée.

L'url (complète) d'une page est constituée du site auquel elle appartient, et d'un suffixe indiquant le chemin vers la page à partir du site (l'attribut "chemin" dans la relation Page). Par exemple (url du) site = "http ://tex.stackexchange.com" et chemin (vers la page) = "/questions/13173/how-to-change-the-position-of-underset". Enfin, il peut coexister différentes versions d'une page à des dates différentes (dans un wiki par exemple) ; c'est pourquoi date_màj fait partie de la clé de la relation Page.

— Rédacteur (id, nom, email)
— Site (url, #webmaster, thème)
— Page (#site, chemin, date_màj, #auteur, contenu)
— Lien (#site1, #page1, #site2, #page2)

Remarque : le modèle relationnel n'est pas idéal pour représenter un graphe. Une recherche google avec "graph database" conduit à d'autres types de SGBD plus adaptés.

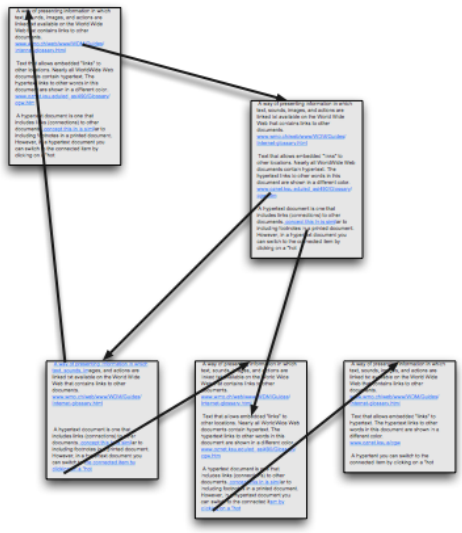


FIGURE 5 – Illustration

Écrivez les requêtes suivantes en SQL. Exprimez les trois premières également en algèbre relationnelle (AR). Si vous choisissez le format informatique vous pouvez écrire "sél[ection]", "proj[ection]", "join[ture]" ...etc au lieu de rechercher les symboles ; du moment qu'il n'y a pas d'ambiguité ça me va.

(a) Les chemins de pages rédigées par l'auteur d'identifiant 32 dans le site www.starwars.com

Solution :

```
SELECT chemin
FROM Page
WHERE auteur = 32 AND site = 'www.starwars.com';
```

AR :
σ[auteur = 32, site = 7](π[chemin](Page))

(b) Les sites comportant des liens vers une page de www.disney.fr

Solution :

```
SELECT DISTINCT site1
FROM Lien
WHERE site2 = 'www.disney.fr';
```

AR :
π[site1](σ[site2 = www.disney.fr](Lien))

(c) Les noms des auteurs ayant mis à jour une page en 2014.

Solution :

```
-- Avec une sous-requête
SELECT nom
FROM Rédacteur
WHERE id IN
  (SELECT auteur
   FROM Page
   WHERE date_màj >= '2014-01-01');
```

AR : avec une jointure, pour varier
π[R.nom](σ[date_màj $\geq$ '2014-01-01'](Rédacteur $\bowtie$[R.id = P.auteur] Page))

(d) i. Les sites ayant au moins un lien vers une page rédigée par Luke,

Solution :

```
SELECT distinct site1
FROM Lien
JOIN
  (SELECT site, chemin
   FROM Page
   WHERE auteur IN
     (SELECT id
      FROM rédacteur
      WHERE nom = 'Luke')) x
  ON site2 = x.site AND page2 = x.chemin;
```

ii. rangés par nombre total de liens (ne vérifiant pas nécessairement la condition) décroissant. Vous pouvez réutiliser la requête précédente avec WITH.

Solution :

```
-- Notant R le résultat de la requête précédente (WITH R as ...)
SELECT NL.site1 as site, NL.nbLiens
FROM R
JOIN
     -- On compte le nombre de liens par site
    (SELECT site1, COUNT(*) AS nbLiens
     FROM Lien
     GROUP BY site1) NL
  ON R.site1 = NL.site1
ORDER BY NL.nbLiens DESC;
```

(e) Toutes les URLs (site + chemin) pouvant être atteintes à partir du site www.disney.fr,

i. en une étape, rangées par site puis par chemin ;

Solution :

```
SELECT site2 || page2 as url
```

```
FROM Lien
WHERE site1 = 'www.disney.fr' AND site2 <> site1
ORDER BY site2, page2;
```

ii. en un nombre quelconque d'étapes ;

Solution :

```
WITH RECURSIVE liensAtteints(site, page) AS (
    -- Initialisation avec les pages de www.disney.fr
    SELECT site, chemin
    FROM Page
    WHERE site = 'www.disney.fr'
  UNION
    SELECT L.site2, L.page2
    FROM Lien L
    JOIN liensAtteints LA
      ON L.site1 = LA.site AND L.page1 = LA.page)
SELECT site || page
FROM liensAtteints
WHERE site <> 'www.disney.fr';
```

en excluant les pages du site de départ dans les résultats de recherche.

(f) la dernière activité connue des auteurs des pages du site www.nintendo.fr (pas forcément sur ce même site). Retournez les URLs des pages, les noms des auteurs et les dates de mise à jour.

Solution :

```
WITH lastUpdates as
  -- il y a un petit piège ici : si on filtre les auteurs de pages
  -- de www.nintendo.fr, on ne se retrouve pas avec les bonnes
  -- dates de mises à jour.
  (SELECT distinct R.nom,
     max(P.date_màj) OVER (PARTITION BY R.nom) as lastMàj
   FROM Rédacteur R join Page P on R.id = P.auteur)
select R.nom, P.site || P.chemin as url, LU.lastMàj
FROM Page P
JOIN Rédacteur R on P.auteur = R.id
JOIN lastUpdates LU ON LU.nom = R.nom
WHERE P.date_màj = LU.lastMàj
  -- on ne filtre donc qu'à la fin de la requête
  AND R.id IN (
    SELECT auteur FROM Page WHERE site = 'www.nintendo.fr');
```

(g) Les auteurs d'une page sur le thème "annonces" contenant un numéro de téléphone, qui sont aussi webmasters d'un site sur un thème différent.

Solution :

```
SELECT R.nom
FROM Page P
JOIN Rédacteur R on R.id = P.auteur
-- Note : l'expression régulière ci-dessous n'est pas parfaite ;
-- elle ne vérifie pas que le numéro est isolé. Par exemple
-- 1230123456789999 sera reconnu comme un numéro.
WHERE substring(P.contenu from '0[0-9](?:(?:.|-| )?[0-9]{2}){4}')
        IS NOT NULL
  AND P.site IN (SELECT url FROM Site WHERE thème = 'annonces')
  AND R.id IN (SELECT webmaster FROM Site WHERE thème <> 'annonces');
```

Toutes les requêtes SQL peuvent être testées en se connectant à la base "web" présente sur 192.168.31.236. Vous disposez de deux alternatives pour les tests.

Connexion sur la machine des TPs :

1. ouvrir un terminal
2. ssh initiales@192.168.31.236 #mot de passe par défaut : tpsql
3. psql -d web #mot de passe par défaut : tpsql

Passage par l'interface web :

1. récupération de votre mot de passe si vous ne le connaissez pas :
   ssh initiales@192.168.31.236 #mot de passe par défaut : tpsql
   cat .web_password
2. naviguez sur http ://192.168.31.236
   login = vos initiales
   mot de passe = la sortie de cat .web_password ci-dessus.
3. choisissez "TP2 : écrire des requêtes SQL" puis la base de données "web"

Exercice 3 [Environ 25% des points]

On considère dans cet exercice la relation Cours (C, P, H, S, E, N) dont les attributs représentent respectivement un cours, un professeur, une heure, une salle, un étudiant et une note. Les dépendances fonctionnelles de cette relation sont

$$C \to P$$
$$HS \to C$$
$$HP \to S$$
$$HE \to S$$
$$CE \to N$$

(a) Sans effectuer de calculs, juste en observant les dépendances fonctionnelles, déterminez le cardinal minimal d'une clé de la relation Cours. Justifiez votre réponse.

Solution :
H et E n'apparaissent jamais en partie droite. Donc ils appartiennent nécessairement à une potentielle clé. Donc une clé contient au moins 2 attributs.

(b) Cherchez alors une clé minimale pour la relation Cours. Utilisez-la pour les questions suivantes.

Solution :
$HE \to S$ et $HS \to C$, donc la connaissance de H et E implique celle de C et S. De plus C détermine P et $CE \to N$, ce qui signifie que l'on peut tout déterminer à partir de H et E. H, E est donc l'unique clé minimale.

(c) Y a-t-il des dépendances fonctionnelles redondantes ? (C'est-à-dire des DF pouvant être déduites à partir d'autres par application des axiomes d'Amstrong ; parcourez éventuellement cette lecture complémentaire intéressante, mais ne vous y perdez pas, vous n'avez pas besoin de tout ça). Justifiez votre réponse. Si oui, continuez l'exercice sans elles.

Solution :
On voit vite que la dépendance contenant HE en partie gauche est nécessaire, car sans elle on ne peut rien déterminer à partir de la clé choisie. N n'est présent en partie droite que dans une seule dépendance : elle n'est donc pas supprimable. Le même argument fonctionne pour l'attribut C, présent en partie droite dans une seule dépendance. Si on enlève $C \to P$, alors il n'y a plus aucun moyen de déterminer P : elle est donc nécessaire elle aussi.

Finalement, $HP \to S$ est la seule dépendance dont la suppression n'empêcherait pas de tout déterminer à partir de la clé. Si on la supprimait alors le seul moyen de déduire $HP \to S$ consisterait à utiliser $HE \to S$, car ce serait la seule dépendance restante faisant intervenir S. Il faudrait alors que P puisse (par application des axiomes d'Amstrong) déterminer E. On se convainc rapidement que c'est impossible : P n'est présent nul part en partie gauche d'une DF. Donc il n'y a aucune dépendance redondante.

(d) Quelles sont les formes normales vérifiée par la relation Cours ? Justifiez la réponse.

Solution :
On est en 2NF car tout dépend pleinement de la clé ; H tout seul ne détermine rien, et E non plus. La 3NF n'est en revanche pas vérifiée car $C \to P$ et C n'est pas un attribut clé.

(e) Décomposez la relation Cours en un ensemble de relations en 3NF en utilisant les théorèmes du paragraphe 3.3.2. Ces relations sont-elles aussi en BCNF ?

Solution :
$HE \to S$, donc on peut choisir $X = HE$, $Y = S$ et $Z = CPN$ puis appliquer le théorème vu en cours. On obtient d'une part $R(H, E, S)$ dont la seule DF non triviale consiste en $HE \to S$ (on ne peut donc pas aller plus loin), et d'autre part $R(H, E, C, P, N)$ qui comporte la dépendance $C \to P$. C ne faisant pas partie de la clé, on doit poursuivre en choisissant par exemple $X = C$, $Y = P$ et $Z = HEN$. On obtient alors (par application du même théorème) $R(C, P)$ d'une part, et $R(C, H, E, N)$ d'autre part. Cette dernière comporte encore la DF $CE \to N$, et doit donc être décomposée. En choisissant $X = CE$, $Y = N$ et $Z = H$ on obtient finalement $R(C, E, N)$ et $R(C, E, H)$, qui ne comportent plus d'anomalies par rapport à la 3NF. On conclut en soulignant les clés :

$$Cours(C, P, H, S, E, N) = R(\underline{H, E}, S) \bowtie R(\underline{H, E}, C) \bowtie R(\underline{C}, P) \bowtie R(\underline{C, E}, N)$$

Ces relations sont en BCNF (aucun attribut clé ne dépend d'un attribut non clé)

(f) Interprétez la décomposition obtenue : vous semble-t-elle cohérente par rapport à la sémantique des attributs ? Que remarquez-vous ?

Solution :
Un cours détermine un professeur : cohérent, signifie qu'un cours est donné par un unique professeur. Un horaire et
- une salle déterminent un cours : cohérent, il se déroule un seul cours dans une salle donnée à une heure fixée ;
- un professeur déterminent une salle : cohérent, on ne peut pas être dans deux endroits à la fois ;
- un étudiant déterminent une salle : cohérent pour la même raison.

Enfin, un cours et un étudiant déterminent une note, ce qui est normal si l'on considère une seule note par matière.

Note : description complémentaire trouvée dans une copie :
- H,E,C = liste d'appel,
- H,E,S = emploi du temps élève,
- C,E,N = relevé de notes,
- C,P = liste des professeurs.

On remarque que la plupart des DF ont donné lieu à une unique relation.

(g) La décomposition est-elle sans perte de dépendances ? Pouvez-vous la transformer intuitivement en une décomposition vérifiant ce critère ? (Un algorithme général est présenté ici, mais vous n'en avez pas vraiment besoin).

Solution :
On a perdu les dépendances $HP \to S$ et $HS \to C$ au profil de $HE \to C$. Intuitivement, conformément à la dernière remarque de la question précédente il

suffirait de remplacer $R(\underline{H,E},C)$ par $R(\underline{H,S},C)$ et $R(\underline{H,P},S)$. L'inconvénient cependant est que l'on ne peut plus réaliser les jointures systématiquement sur les clés des relations : on perd en efficacité.

Exercice 4 [Environ 25% des points]

On considère les schémas de relations suivants, modélisant les commandes passées dans un restaurant. Celles-ci peuvent être consommées sur place ou emportées ; dans ce dernier cas on effectue une remise de 10% sur le prix total. Certaines parties de la commande peuvent être offertes (les apéritifs par exemple). Les attributs commande et produit référencent respectivement l'identifiant d'une commande et d'un produit.

— Produit (id, type, nom, prix_unitaire, stock)
— Entête_commande (id, prix, à_emporter, date)
— Détail_commande (#commande, #produit, quantité, offert)

Les commandes SQL générant les tables correspondantes sont accessibles en lecture dans le répertoire /home/tpbd/m1im sur la machine 192.168.31.236. Chargez-les dans votre base en tapant "\i /home/tpbd/m1im/restaurant.sql" une fois dans psql ou "psql -f /home/tpbd/m1im/restaurant.sql" depuis un terminal. Cela permettra de tester vos réponses au fur et à mesure.

Rappel :
L'interaction avec PostgreSQL est possible par connexion sur la machine des TPs :

1. ouvrir un terminal
2. ssh initiales@192.168.31.236
3. psql

Ou bien en passant par l'interface web :

1. naviguez sur http ://192.168.31.236
2. choisissez "TP3 : connexion à votre base"

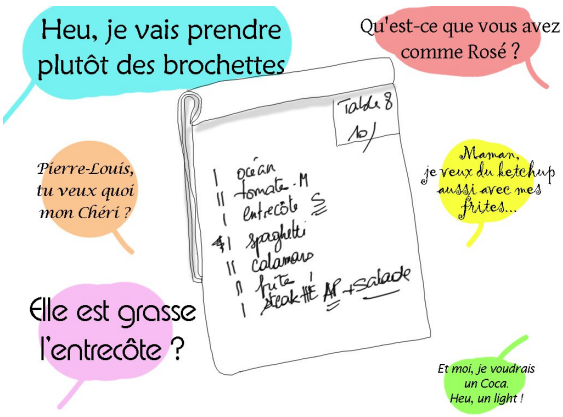


FIGURE 6 – Illustration

Implémentez les fonctions suivantes.

(a) Trigger verifieMajDetail() qui vérifie les disponibilités et met à jour les stocks lors de l'ajout, modification ou suppression d'un détail de commande.

Solution :

```
create function verifieMajDetail() returns trigger as $$
declare
  deltaStock integer := 0;
  stock integer;
  idProduit integer;
begin
  -- compute deltaStock
  if tg_op = 'INSERT' THEN
    deltaStock := 0;
  else
    deltaStock := old.quantité;
  end if;
  if tg_op <> 'DELETE' THEN
    deltaStock := deltaStock - new.quantité;
  end if;
  -- get stock
  if tg_op = 'INSERT' then
    idProduit = new.produit;
  else
    idProduit = old.produit;
  end if;
  select Produit.stock from Produit where id = idProduit into stock;
  -- check if operation is feasible
  if stock + deltaStock < 0 then
```

```
    raise exception
      'Unable to complete the transaction: not enough stock';
    return null;
  end if;
  if tg_op = 'DELETE' then
    return old;
  end if;
return new;
end;
$$ language plpgsql;

create trigger verifieMajDetail_trigger
  BEFORE INSERT or update or delete
  ON Detail_commande
  FOR EACH ROW
  EXECUTE PROCEDURE verifieMajDetail();
```

(b) Fonction finaliseCommande() prenant en argument un identifiant de commande : enregistre la date, calcule le prix total et le met à jour. Cherchez aussi un moyen d'empêcher toute future modification ou suppression de la commande. Vous aurez probablement besoin d'un trigger.

Solution :

```
create function finaliseCommande(idCommande integer)
  returns void as $$
declare
  r Detail_commande%rowtype;
  prixFinal real := 0.0;
  à_emporter boolean;
begin
  -- calcul du prix initial (avec produits offerts déduits)
  FOR r in
    (select * from Detail_commande where commande = idCommande)
  loop
    if (NOT r.offert) then
      prixFinal := prixFinal + r.quantité *
        (select prix_unitaire from Produit where id = r.produit);
    end if;
  end loop;
  -- déduction des 10% sur vente à emporter
  select Entete_commande.à_emporter
  from Entete_commande
  where id = idCommande
  into à_emporter;
```

```
  if à_emporter then
    prixFinal := 0.9 * prixFinal;
  end if;

  -- mises à jour finalisant la commande
  update Entete_commande
  set prix = prixFinal
  where id = idCommande;

  update Entete_commande
  set date = NOW()
  where id = idCommande;
end;
$$ language plpgsql;

create or replace function preventCommandeModif()
  returns trigger as $$
declare
  idCommande integer;
  prix real;
  dateCommande date;
begin
  idCommande := old.id;
  select Entete_commande.prix
  from Entete_commande
  where id = idCommande
  into prix;

  select date
  from Entete_commande
  where id = idCommande
  into dateCommande;

  if prix >= 0 and dateCommande IS NOT NULL then
    -- la commande est finalisée
    raise warning 'opération impossible';
    return null;
  end if;
  return new;
end;
$$ language plpgsql;

create trigger preventCommandeModif_trigger
```

```
  before update or delete
  on Entete_commande
  for each row
  execute procedure preventCommandeModif();
```

(c) Fonction imprimeAddition() prenant en argumant un identifiant de commande, et retournant une table correspondant à l'addition : trois colonnes, "quantité", "nom du produit" et "prix". La dernière ligne ne contient rien dans la première colonne, puis "Total" dans la seconde et le prix total (supposé calculé) dans la troisième.

Solution :

```
create type LigneAddition as (
  quantité integer, description varchar(64), prix real);

-- affichage non optimal (total avec réduction indiqué en premier),
-- mais c'est pour montrer qu'une fonction SQL peut suffire ici
create function imprimeAddition(idCommande integer)
  returns setof LigneAddition as $$
with DetailAddition AS (
  select D.quantité, P.description,
         cast(P.prix_unitaire * D.quantité as real) as prix
  from Produit P
  join Detail_commande D
    on P.id = D.produit
  WHERE D.commande = idCommande AND NOT D.offert
  union
  select D.quantité, P.description || ' (offert)', 0.0
  from Produit P
  join Detail_commande D
    on P.id = D.produit
  WHERE D.commande = idCommande AND D.offert)
select *
from DetailAddition
union values (
  NULL::integer, 'Total (initial)',
  (select sum(prix) from DetailAddition))
union values (
  NULL::integer, 'Total (après réductions)',
  (select prix from Entete_commande WHERE id = idCommande))
order by prix; -- tri pour visualiser les totaux en dernier
$$ language sql;
```