M1 Informatique – Année 2025-2026
Bases de données avancées
Épreuve écrite
J. Darmont (http://eric.univ-lyon2.fr/jdarmont/), 07/11/2025

Durée : 1h30 – Documents autorisés – Barème fourni à titre indicatif

Questions générales (3 points)

1. Soit la table PRODUIT(id, nom, description, categorie, prix). Pour charger en mémoire le résultat de la requête SELECT * FROM PRODUIT WHERE categorie = 'Téléphone', quel mécanisme faut-il utiliser : SELECT INTO ou bien un curseur ? Justifier.

2. Un curseur paramétré est-il implicite (non lié) ou explicite (lié) ? Justifier.

3. Quelle caractéristique permet aux données semi-structurées de ne pas avoir de schéma ?

PL/pgSQL (10 points)

Soit la base de données « Los Surfers Muertos » dont le schéma relationnel est donné ci-dessous.

SURFEUR (NumSurfeur, Surnom, DateNaiss, Sexe)
SPOT (CodeSpot, NomSpot, Localisation)
COMPET (Surfeur#, Spot#, Annee, Points, Gain)

1. Créer un type composite (enregistrement) de nom tResultat, constitué de deux champs : libelle (chaîne de caractères) et valeur (nombre).

2. Écrire une fonction de nom gainsHF() qui retourne la moyenne des gains des surfeurs (H) et des surfeuses (F), respectivement.

3. Créer un type composite de nom tCompet, constitué de trois champs : nom de spot, année et points.

4. Écrire une fonction de nom listeCompet() qui retourne le nom du spot, l'année et le nombre de points.

5. Écrire un déclencheur de nom trigCompet, ainsi que sa fonction associée, qui vérifie que les points et le gain ont bien une valeur. Dans le cas contraire, déclencher un ou deux *warnings*, selon que ce sont les points, le gain ou les deux qui sont sans valeur.

XQuery (7 points)

Soit le document XML *gestion.xml*. La structure de ce document est donnée ci-dessous.

```
gestion
    clients
        client*
            @id
            nom
            prenom
    produits
        produit*
            @id
            designation
            prix
            stock
    commandes
        commande*
            @idcli
            @idprog
            quantite
```

Formuler à l’aide du langage XQuery les requêtes suivantes (utiliser la syntaxe FLWOR pour plus de lisibilité).

1. Nombres de clients.

2. Noms et prénoms des clients triés par noms et prénoms.

3. Désignation et prix des produits qui coûtent plus de 10.

4. Désignation des produits dont le stock est inférieur ou égal à 100 et dont le prix est inférieur ou égal à 5, triés par désignation.

5. Clients et quantités des commandes.

6. Somme de tous les stocks de produits.

7. Clients qui ont plus de 2 commandes.

Correction Questions générales

1. La requête pouvant retourner plusieurs n-uplets, il faut utiliser un **curseur**.

2. Il est possible de définir des **curseurs paramétrés explicites**, mais il est bien plus simple d'utiliser des curseurs paramétrés implicites.

3. Les données semi-structurées ne nécessitent pas obligatoirement de schéma car elles sont **autodescriptives**.

Correction PL/pgSQL

```
-- 1
create type tResultat as (
     libelle varchar,
     valeur numeric
);


-- 2
create or replace function gainsHF() returns setof tResultat as $$
     declare
          res tResultat;
     begin
          -- Surfeurs
          res.libelle := 'Surfeurs';
          select avg(gain) into res.valeur from compet, surfeur
               where compet.numSurfeur = surfeur.numSurfeur and sexe = 'H';
          return next res;
          -- Surfeuses
          res.libelle := 'Surfeuses';
          select avg(gain) into res.valeur from compet, surfeur
               where compet.numSurfeur = surfeur.numSurfeur and sexe = 'F';
          return next res;
          return;
     end
$$ language plpgsql;


-- 3
create type tCompet as (
     spot varchar,
     annee numeric,
     points numeric
);


-- 4
create or replace function listeCompet() returns setof tCompet as $$
     declare
          c tCompet;
     begin
          for c in select nomSpot, Annee, Points from compet, surfeur, spot
               where compet.numSurfeur = surfeur.numSurfeur
               and compet.codeSpot = spot.codeSpot
          loop
               return next c;
          end loop;
          return;
     end
$$ language plpgsql;
```

```
-- 5
-- Fonction du déclencheur
create or replace function trigCompet() returns trigger as $$
     begin
          if new.points is null then
               raise warning 'Il manque une valeur de points.';
          end if;
          if new.gains is null then
               raise warning 'Il manque une valeur de gain.';
          end if;
          return new;
     end
$$ language plpgsql;

-- Déclencheur
create trigger trigCompet
before insert or update
on compet
for each statement
execute procedure trigCompet();
```

Correction XQuery

```
(: 1 :)
let $s := count(//client)
return <res>{$s}</res>

(: 2 :)
for $cli in //client
order by $cli/nom, $cli/prenom
return <res>{$cli/nom}{$cli/prenom}</res>

(: 3 :)
for $pro in //produit
where $pro/prix > 10
return <res>{$pro/designation}{$pro/prix}</res>

(: 4 :)
for $pro in //produit
order by $pro/designation
where $pro/stock <= 100 and $pro/prix <= 5
return <res>{$pro/designation}</res>

(: 5 :)
for $cli in //client,
    $com in //commande
where $cli/@id = $com/@idcli
return <res>{$cli/nom}{$com/quantite}</res>

(: 6:)
let $a := avg(
  for $cli in //client,
      $pro in //produit,
      $com in //commande
  where $cli/@id = $com/@idcli
  and $pro/@id = $com/@idprod
  return <res1>{$pro/stock}</res1>
)
return <res2>{$a}</res2>
```

```
(: 7 :)
for $cli in //client,
    $com in //commande
where $cli/@id = $com/@idcli
group by $g := $cli/nom
let $c := count($com/quantite)
where $c > 2
return <res>{$cli/nom}<nb-commandes>{$c}</nb-commandes></res>
```