Sujet pour l'épreuve finale d'ABD

5 mars 2015 – 9h30-11h00 – Amphi A

La durée de l'épreuve est estimée à 86 minutes. Le barème détaillé de chaque question est purement indicatif.
Document autorisé : **une** feuille manuscrite recto-verso format A4.
Sujet à compléter et à rendre à la fin de l'épreuve.
L'espace proposé pour chaque question correspond à la taille de la réponse attendue.
Barème indicatif : chaque question vaut environ le même nombre de points.

1. Que signifie l'expression "cinq neuf" ?

   Solution: Un système est disponible 99,999% du temps, ce qui représente environ 5 minutes d'indisponibilité par an.
   1 point pour l'équivalence en pourcentage. 1 point pour la relation avec la disponibilité. 1 point bonus pour l'équivalence en minute par an.

2. Définir ce que sont les langages SQL/PSM et PL/SQL et discuter comment ils se différent.

   Solution: Ces langages servent à écrire des procédures stockées. SQL/PSM est le langage faisant parti du standard SQL et supporté par MySQL. PL/SQL est l'alternative, plus puissante, utilisée par Oracle Database et PostgreSQL (sous forme de variante pour ce dernier).
   1 point par description. 1 point pour la différence.

3. Citer deux SGBDs propriétaires et les sociétés qui les éditent.

   Solution: Oracle Database (Oracle), DB2 (IBM) et SQL Server (Micorsoft).
   1 point par SGBD. 1 point par société correspondante.

4. Quelle est le numéro de version du logiciel PostgreSQL utilisé lors des séances TP en salle machine ?

   Solution: La version du logiciel PostgreSQL utilisé est 9.4.
   1 point pour le numéro de version majeur, 1 point pour le numéro de version mineur.

5. Définir chaque propriété ACID.

   Solution:
   Atomicité Chaque transaction est exécutée entièrement ou pas du tout.
   Cohérence Les contraintes sont toutes respectées au début et à la fin de chaque transaction.
   Isolation Le résultat de l'exécution concurrente de plusieurs transactions est équivalente à une exécution sérialisée de ces transactions.
   Durabilité Les modifications d'une transaction validée ne sont jamais perdues.
   1 point par propriété citée. 1 point par définition.

6. Dans quels cas les vues matérialisées améliorent-elles les performances ?

Solution: Dans les cas suivants :
- Lorsque la requête de sélection associée à la vue est complexe : la vue joue alors le rôle de cache pour les requêtes suivantes.
- Lorsque la vue est très sélective et qu'elle retourne peu de lignes : il n'y alors pas besoin de reparcourir toutes les lignes.
- Lorsque les tables de base de la vue sont peu modifiées : la vue n'aura pas beaucoup à être mise à jour fréquemment.
- Lorsque beaucoup de requêtes de sélection utilisent la vue : on factorise alors beaucoup de calcul.

1 point pour le compromis entre lecture et mise à jour. 1 point pour la sélectivité ou la complexité de la requéte.

7. Soit la table `PATIENT(numSecuPatient,nom,prenom,adresse,rhesus)`.
Limiter les états valides de cette table à l'aide d'une contrainte qui garantit que l'adresse est inexistante (`NULL`) si le nom ou le prénom ne sont pas saisis.

Solution:
```
ALTER TABLE PATIENT ADD CONSTRAINT adressenull
CHECK (adresse IS NULL OR nom IS NOT NULL AND prenom IS NOT NULL);
```

8. Soit le schéma relationnel suivant :
`SERVICE(codeService,nom,categorie)`
`MEDECIN(matriculeMedecin,nom,prenom,@codeService,grade,dateEmbauche)`
Les clés primaires sont soulignées et les clés étrangères sont précédées d'une arobase.
Limiter les états valides de la base de données à l'aide d'un déclencheur en insertion (de type `FOR EACH ROW`) sur la table `MEDECIN` qui contrôle qu'un interne (il s'agit du `grade` d'un médecin) ne peut pas intégrer le service de cancérologie (il s'agit du `nom` du service).

Solution:
```
CREATE TRIGGER interneCancerologie
AFTER INSERT ON MEDECIN
FOR EACH ROW
WHEN NEW.grade = 'interne' AND
(SELECT nom = 'cancérologie' FROM SERVICE S
WHERE S.codeService = NEW.codeService)
DELETE FROM MEDECIN M
WHERE M.matriculeMedecin = NEW.matriculeMedecin;
```

9. Soit le schéma relationnel suivant :
`SERVICE(codeService,nom,categorie)`
`MEDECIN(matriculeMedecin,nom,prenom,@codeService,grade,dateEmbauche)`
Les clés primaires sont soulignées et les clés étrangères sont précédées d'une arobase.
Écrire une fonction en PL/pgSQL qui prend en entrée le matricule d'un médecin et qui retourne en sortie son salaire. Le salaire dépend du grade du médecin : 1200 pour les externes et 1800 pour les internes.

Solution:
```
CREATE FUNCTION calculSalaire(matr INTEGER)
RETURNS INT AS $$
DECLARE
gr TEXT;
BEGIN
SELECT categorie INTO gr FROM MEDECIN
```

```
WHERE MEDECIN.matriculeMedecin = matr;
CASE gr
WHEN 'Interne' THEN
RETURN 1800;
WHEN 'Externe' THEN
RETURN 1200;
END CASE;
END $$ language plpgsql;
```

10. Soit le schéma relationnel suivant :

    PATIENT(numSecuPatient,nom,prenom,adresse,rhesus)
    VISITE(codeVisite,@matriculeMedecin,@numSecuPatient,dateEtHeure,duree)
    PRESCRIPTION(@codeVisite,@codeMedicament,posologie,duree)

    Les clés primaires sont soulignées et les clés étrangères sont précédées d'une arobase.

    Mettre au point une requête de sélection qui donne la liste des patients (numéro de sécurité sociale seulement) ayant eu entre 1 et 5 prescriptions.

    Solution:
    ```
    SELECT pa.numSecuPatient
    FROM PATIENT pa, VISITE v, PRESCRIPTION pr
    WHERE pa.numSecuPatient=v.numSecuPatient
    AND v.codeVisite=pr.codeVisite
    GROUP BY pa.numSecuPatient
    HAVING COUNT(*) BETWEEN 1 AND 5;
    ```

11. On considère la création de tables SQL suivante :

    ```
    CREATE TABLE R(e INT PRIMARY KEY, f INT);
    CREATE TABLE S(c INT PRIMARY KEY REFERENCES R(e) ON UPDATE CASCADE, d INT);
    CREATE TABLE T(a INT PRIMARY KEY, b INT REFERENCES S(c) ON UPDATE CASCADE);
    ```

    On suppose que `R(e,f)` contient les quatre tuples suivants : (1,5), (2,14), (7,18) et (9,23). On suppose que `S(c,d)` contient les quatre tuples suivants : (2,6), (1,8), (9,6) et (2,2). On suppose que `T(a,b)` contient les quatre tuples suivants : (4,9), (9,2), (13,1) et (19,1). Les contraintes d'intégrité référentielle déclarées lors de la création des tables peuvent entraîner des mises à jour supplémentaires automatiques lorsque certaines lignes sont modifiées.

    Quelle mise à jour sur `R.e` laisse la table `T` dans un état tel que la somme de la colonne `b` vaut 15 ? Cette mise à jour ne doit modifier qu'une seule valeur.

    Solution: Pour chaque mise à jour possible de l'attribut `e` de la table `R`, on répercute la modification en cascade sur l'attribut `c` de la table `S`, puis sur l'attribut `b` de la table `T`. L'attribut `b` de la table `T` des tuples (4,9), (9,2), (13,1) et (19,1) référence l'attribut `c` de la table `S` des tuples (2,6), (1,8), (9,6) et (2,2) qui référence l'attribut `e` de la table `R` des tuples (1,5), (2,14), (7,18), and (9,23).

12. Soit trois tables vides `T1`, `T2` et `T3` contenant chacune un unique attribut `A` de type entier. 6 déclencheurs sont définis de la façon suivante (nous supposons que ces déclencheurs sont supportés) :

    ```
    CREATE TRIGGER D1_1
    BEFORE INSERT ON T1
    FOR EACH ROW
    WHEN (SELECT COUNT(*) FROM T2) < 1
    INSERT INTO T2 VALUES (1)

    CREATE TRIGGER D2_1
    BEFORE INSERT ON T2
    FOR EACH ROW
    WHEN (SELECT COUNT(*) FROM T3) < 1
    INSERT INTO T3 VALUES (6)

    CREATE TRIGGER D3_1
    BEFORE INSERT ON T3
    FOR EACH ROW
    WHEN (SELECT COUNT(*) FROM T1) > 10
    ```

```
INSERT INTO T1 VALUES (9)

CREATE TRIGGER D1_2
AFTER INSERT ON T1
FOR EACH ROW
WHEN (SELECT COUNT(*) FROM T2) < 2
INSERT INTO T3 VALUES (11)

CREATE TRIGGER D2_2
AFTER INSERT ON T2
FOR EACH ROW
WHEN (SELECT COUNT(*) FROM T1) < 10
INSERT INTO T1 VALUES (16)

CREATE TRIGGER D3_2
AFTER INSERT ON T3
FOR EACH ROW
WHEN (SELECT COUNT(*) FROM T3) < 2
INSERT INTO T2 VALUES (22)
```

On suppose que l'on exécute la requête `INSERT INTO T1 VALUES (0)`.
Décrire l'état des tables `T1`, `T2` et `T3` en donnant les valeurs dans leur ordre d'insertion (par exemple "`T1 : 11, 12`" si 11 est la première valeur insérée et 12 la seconde).

Solution: La table `T1` contient (16, 16, 0). La table `T2` contient (22, 1). La table `T3` contient (6, 11).
1 point par table correcte (vrai ou faux).

13. On considère la table `T(A,B,C)` dont le propriétaire est Amy et on suppose que les actions suivantes sont exécutées (chaque commande est numérotée et préfixée par l'utilisateur qui l'exécute) :

```
1. Amy: Grant Select on T to Bob With Grant Option
2. Bob: Grant Select on T to Carol With Grant Option
3. Carol: Grant Select(A,C) on T to David With Grant Option
4. Carol: Grant Select(A,B) on T to Eve With Grant Option
5. Amy: Grant Select on T to Eve
6. Amy: Grant Select(C) on T to Frank
7. David: Grant Select(A,C) on T to Frank With Grant Option
8. Eve: Grant Select(A,C) on T to Frank
9. David: Grant Select(A) on T to Gary
10. Eve: Grant Select(A) on T to Gary
11. Amy: Revoke Select on T From Eve Restrict
12. David: Revoke Select(A) on T From Eve
13. Bob: Revoke Select on T From Carol Cascade
14. Amy: Revoke Select on T From Bob Restrict
```

   1. Quelle commande `grant` et/ou `revoke` est interdite (ou sont interdites, s'il y en a plusieurs) ?
   2. À l'issue des ces commandes (en ignorant la ou les commandes interdites), quel(s) droit(s) possède(nt) l'utilisateur Frank sur la table `T` ?

Solution: Commandes interdites : 8 et 12. Droit de Frank : Select(C).

14. On considère la table `Notation(nom, note)` et les trois transactions suivantes :
   — T1 :
   ```
   Begin Transaction
   S1: update notation set note = note + 2;
   S2: update notation set note = note - 1;
   Commit
   ```
   — T2 :
   ```
   Begin Transaction
   S3: insert into notation values ("Frédéric", 2);
   Commit
   ```
   — T3 :
   ```
   Begin Transaction
   S4: delete from notation where note < 8;
   Commit
   ```

Chaque instruction `S1`, `S2`, `S3` et `S4` s'exécute atomiquement. La table contient trois tuples avant que les transactions ne démarrent : `("Julien", 18)`, `("François", 10)` et `("Maxime", 5)`. Chaque transaction s'exécute avec le niveau d'isolation `READ UNCOMMITTED` et termine correctement.

À l'issue de ces transactions, on calcule la moyenne avec `select avg(note) from notation`.

Quelles sont les moyennes finales possibles ?

Solution: On cherche d'abord l'ensemble des exécutions sérialisés possibles :
- T1, T2, T3 : 15
- T1, T3, T2 : 10.7
- T2, T1, T3 : 15
- T2, T3, T1 : 15
- T3, T1, T2 : 10.7
- T3, T2, T1 : 11

On a trois valeurs possibles : 15, 10.7 et 11.

On cherche désormais les exécutions non-sérialisées :
- S1, S3, S2, S4 : 15
- S1, S3, S4, S2 : 15
- S3, S1, S4, S2 : 15
- S1, S4, S2, S3 : 10.7
- S1, S4, S3, S2 : 10.3
- S4, S1, S3, S2 : 10.3

Une seule valeur diffère : 10.3.

15. On considère la table `T(K, V)` et la définition de vue suivante (on considère qu'elle est automatiquement modifiable malgré la sous-requête correlée) :

```
CREATE VIEW Vue AS
SELECT * FROM T AS T1 WHERE V NOT IN (SELECT V FROM T AS T2 WHERE T1.K <> T2.K)
OR K * V > 21 OR K > 5
WITH CHECK OPTION;
```

La table `T` est vide initialement et on exécute les requêtes suivantes :

```
1. INSERT INTO Vue VALUES (7, 4);
2. INSERT INTO Vue VALUES (6, 0);
3. INSERT INTO Vue VALUES (1, 4);
4. INSERT INTO Vue VALUES (5, 12);
5. INSERT INTO Vue VALUES (9, 0);
6. INSERT INTO Vue VALUES (1, 17);
7. INSERT INTO Vue VALUES (8, 12);
8. INSERT INTO Vue VALUES (5, 17);
```

Décrire le résultat de ces requêtes.

Solution: L'insertion 3 échoue.