DISVE
Licence

Année Universitaire 2007/2008
Session 2 de Printemps

**PARCOURS :** CSB4 & CSB6
**UE :** INF 159, Bases de données
**Épreuve :** INF 159 EX
**Date :** juin 2008
**Heure :** **Durée :** 1 heure 30
**Documents :** non autorisés
Épreuve de M. Alain Griffault

# SUJET + CORRIGE

## Avertissement

- La plupart des questions sont indépendantes.
- Le barème de chaque question est (approximativement) proportionnel à sa difficulté.
- L'espace pour répondre est suffisant (sauf si vous l'utilisez comme brouillon, ce qui est fortement déconseillé).

## Le sujet

Le sujet porte sur une gestion simplifiée d'adresses électroniques et de numéros de téléphone d'employés. Soit `Adresses(Nom, Entreprise, Bureau, Tel_Bureau, Portable_Travail, Portable_Perso, Email_Travail, Email_Perso)` une relation. Un exemple est donné dans la table suivante, où vu le nombre important d'attributs, les colonnes et lignes sont inversées.

| Attributs | tuple 1 | tuple 2 | tuple 3 | tuple 4 |
|---|---|---|---|---|
| Nom text | Bonichon | Griffault | Griffault | Wacrenier |
| Prenom text | Nicolas | Alain | Odile | Pierre-Andre |
| Entreprise text | LaBRI | LaBRI | SSII | LaBRI |
| Bureau text | 304 | 262 | 13 | 118 A29bis |
| Tel_Bureau integer | 540006931 | 540006915 | 999999999 | 524574099 |
| Portable_Travail integer | 0 | 0 | 0 | 0 |
| Email_Travail text | inf159@LaBRI.fr | griffaul@labri.fr | ogriffault@ssii.com | inf159@laBRI.fr |
| Tel_Maison integer | 0 | 566666666 | 566666666 | 0 |
| Email_Maison text | INCONNU | griffault@free.fr | griffault@free.fr | INCONNU |

Pour une gestion correcte, la relation est munie des contraintes d'intégrité suivantes :

C1 : Tous les attributs sont déclarés `NON NULL`.

C2 : `PRIMARY KEY (Nom, Prenom)` qui indique la clef primaire.

C3 : `CONSTRAINT PortableTravail CHECK(Portable_Travail > 100000000 OR Portable_Travail = 0)` qui indique que le numéro est valide ou bien que l'employé n'a pas de portable au travail.

C4 : `CONSTRAINT TelMaison CHECK(Tel_Maison > 100000000 OR Tel_Maison = 0)` qui indique que le numéro est valide ou bien que l'employé n'a pas de téléphone personnel.

C5 : `CONSTRAINT EmailTravail CHECK(Email_Travail LIKE '%@%')` qui indique que l'adresse est valide.

C6 : `CONSTRAINT EmailMaison CHECK(Email_Maison LIKE '%@%' OR Email_Maison LIKE 'INCONNU')` qui indique que l'adresse est valide ou bien que l'employé n'a pas d'adresse électronique personnelle.

**Interprétation :** Les attributs `Portable_Travail, Tel_Maison` et `Email_Maison` sont *facultatifs*. Afin d'avoir une relation 1NF, le concepteur de la base veut éviter les valeurs `NULL` permises par certains SGBD. Pour cela il impose que les attributs facultatifs aient une valeur `0` lorsqu'il s'agit d'un entier (`Portable_Travail, Tel_Maison`), et une valeur "`INCONNU`" lorsqu'il s'agit d'un texte (`Email_Maison`). Il faut également noter que plusieurs personnes peuvent partager un même bureau, et par conséquent, partager un même numéro de téléphone au travail. De même les personnes d'une même famille auront les mêmes téléphones et adresses maison.

De plus, pour éviter d'avoir à gérer beaucoup d'adresses électroniques, une entreprise peut choisir d'avoir des adresses électroniques "individuelles" pour les cadres, mais des adresses "collectives" pour des groupes d'employés qui n'ont accès qu'à des postes en libre service.

**Remarque :** Les données sont fictives.

**Exercice 1 (Requêtes SQL sur `Adresses` (9 points))**

***Question 1.1 (1 point)*** *Écrire une requête SQL qui retourne les `Nom, Prenom` des employés qui n'ont pas de portable au travail.*

**Réponse :**

```
-- Les employes qui n'ont pas de portable au travail
CREATE VIEW DRH.reponse_1_1 AS
SELECT  Nom, Prenom
FROM    DRH.Adresses
WHERE   Portable_Travail = 0;
```

***Question 1.2 (1 point)*** *Écrire une requête SQL qui retourne pour chaque `Bureau` de chaque `Entreprise`, le numéro de téléphone `Tel_Bureau` et le nombre d'employés qui le partage.*

**Réponse :**

```
-- Le téléphone et le nombre d'employes par bureau et par entreprise
CREATE VIEW DRH.reponse_1_2 AS
SELECT DISTINCT Entreprise, Bureau, Tel_Bureau, COUNT(*) AS Nb
FROM            DRH.Adresses
GROUP BY        Entreprise, Bureau, Tel_Bureau;
```

***Question 1.3 (1 point)*** *Soit l'expression algébrique*

$$\alpha[Nom : Abus](\pi[Nom,\ Prenom](\sigma[Portable_Travail = Tel_Maison\ AND\ Portable_travail <> 0](Adresses)))$$

*Écrivez la requête SQL correspondante, puis expliquez ce qu'elle calcule.*

**Réponse :**

```
-- les noms et prenoms des employés qui utilisent
-- leur portable de travail chez eux.
CREATE VIEW DRH.reponse_1_3 AS
(SELECT DISTINCT Nom AS Abus, Prenom
 FROM            DRH.Adresses
 WHERE           Portable_Travail = Tel_Maison
   AND           Portable_Travail <> 0);
```

*La requête SQL retourne les noms des employés qui utilisent leur portable de travail chez eux.*

***Question 1.4 (1 point)*** *Écrire une expression algébrique qui retourne les `Entreprise` ayant au moins un employé sans portable de travail.*

**Réponse :**

$$\pi[Entreprise](\sigma[Portable_Travail = 0](Adresses))$$

***Question 1.5 (1,5 point)*** *Écrire une expression algébrique qui retourne les `Entreprise` dont tous les employés ont un portable de travail.*

**Réponse :**

$$\pi[Entreprise](Adresses) - \pi[Entreprise](\sigma[Portable_Travail = 0](Adresses))$$

***Question 1.6 (1,5 point)*** *Écrire une requête SQL qui retourne les couples* `Entreprise, Bureau` *ayant au moins deux* `Email_Travail` *différents.*

**Réponse :**

```
-- les (entreprise, bureau) ayant au moins deux Email_Travail differents.
-- Une solution par un produit de relations
CREATE VIEW DRH.reponse_1_6_v1 AS
(SELECT DISTINCT A1.Entreprise, A1.Bureau
 FROM            DRH.Adresses AS A1, DRH.Adresses AS A2
 WHERE           A1.Entreprise = A2.Entreprise
   AND           A1.Bureau = A2.Bureau
   AND           A1.Email_Travail <> A2.Email_Travail);
-- Une solution avec les agregats
CREATE VIEW DRH.reponse_1_6_v2 AS
(SELECT DISTINCT Entreprise, Bureau
 FROM            DRH.Adresses
 GROUP BY        Entreprise, Bureau
 HAVING          COUNT (DISTINCT Email_Travail) > 1):
```

***Question 1.7 (2 points)*** *Écrire une requête SQL qui retourne les couples* `Entreprise, Bureau` *ayant tous les employés avec le même* `Email_Travail`*.*

**Réponse :**

```
-- les (entreprise, bureau) ayant un seul Email_Travail.
-- Une solution par un produit de relations
CREATE VIEW DRH.reponse_1_7_v1 AS
(SELECT DISTINCT Entreprise, Bureau
 FROM            DRH.Adresses
 EXCEPT
 SELECT DISTINCT A1.Entreprise, A1.Bureau
 FROM            DRH.Adresses AS A1, DRH.Adresses AS A2
 WHERE           A1.Entreprise = A2.Entreprise
   AND           A1.Bureau = A2.Bureau
   AND           A1.Email_Travail <> A2.Email_Travail);
-- Une solution avec les agregats
CREATE VIEW DRH.reponse_1_7_v2 AS
(SELECT DISTINCT Entreprise, Bureau
 FROM            DRH.Adresses
 GROUP BY        Entreprise, Bureau
 HAVING          COUNT (DISTINCT Email_Travail) = 1):
```

**Exercice 2 (Normalisation (7 points))** *Le schéma conceptuel de la base de données comprend maintenant les contraintes définies à la page 1 mais également des dépendances fonctionnelles.*

*DF1 :* $\{$`Nom, Prenom`$\} \rightarrow \{$`Entreprise, Bureau`$\}$

*DF2 :* $\{$`Entreprise, Bureau`$\} \rightarrow \{$`Tel_Bureau`$\}$

*DF3 :* $\{$`Nom, Prenom, Entreprise`$\} \rightarrow \{$`Email_Travail, Portable_Travail`$\}$

*DF4 :* $\{$`Nom`$\} \rightarrow \{$`Tel_Maison`$\}$

*DF5 :* $\{$`Nom`$\} \rightarrow \{$`Email_Maison`$\}$

***Question 2.1 (1 point)*** *Donner toutes les clefs candidates de la relation* `Adresses`*.*

**Réponse :**
– *Les dépendances fonctionnelles donnent :* $C = \{$`Nom, Prenom`$\}$

***Question 2.2 (1 point)*** *Même si l'on suppose qu'il n'y a aucun doublon dans* `Adresses`*, justifiez pourquoi la relation* `Adresses` *n'est pas en troisième forme normale.*

**Réponse :** *Une seule des explications suivantes est suffisante (liste non exhaustive).*

**Non 2NF :** *La clef* $\{$`Nom, Prenom`$\}$ *et* $\{$`Nom`$\} \longrightarrow \{$`Tel_Maison`$\}$*.*

**Non 3NF :** *La clef* {*Nom, Prenom*} *et* {*Entreprise, Bureau*} $\longrightarrow$ {*Tel_Bureau*}.

***Question 2.3 (2 points)*** *Appliquer un algorithme (ou une technique) de normalisation pour obtenir une décomposition, sans perte d'information, de la relation* `Adresses` *en un ensemble de relations BCNF. Vous n'écrirez sur la copie que les nouvelles relations et les dépendances fonctionnelles utilisées lors des projections effectuées.*

**Réponse :** *Décomposition en BCNF :*
- *Societes (Entreprise, Bureau, Tel_Bureau) qui provient de* {*Entreprise, Bureau*} $\longrightarrow$ {*Tel_Bureau*}
- *Maisons (Nom, Tel_Maison, Email_Maison) qui provient de* {*Nom*} $\longrightarrow$ {*Tel_Maison, Email_Maison*}
- *Employes (Nom, Prenom, Entreprise, Bureau, Portable_Travail, Email_Travail) formés des attributs restant, et qui correspond à* {*Nom, Prenom*} $\longrightarrow$ {*Entreprise, Bureau, Portable_Travail, Email_Travail*}

***Question 2.4 (1,5 point)*** *Donner pour chaque relation de votre normalisation, les contraintes d'intégrité et les clefs étrangères présentes.*

**Réponse :**

```
CREATE TABLE DRH.Societes (
  Entreprise text NOT NULL,
  Bureau text NOT NULL,
  Tel_Bureau integer NOT NULL,
  PRIMARY KEY (Entreprise, Bureau),
  CONSTRAINT TelBureau CHECK(Tel_Bureau > 100000000)
);

CREATE TABLE DRH.Maisons (
  Nom text NOT NULL,
  Tel_Maison integer NOT NULL DEFAULT 0,
  Email_Maison text NOT NULL DEFAULT 'INCONNU',
  PRIMARY KEY (Nom),
  CONSTRAINT TelMaison CHECK(Tel_Maison > 100000000 OR Tel_Maison = 0),
  CONSTRAINT EmailMaison CHECK(Email_Maison LIKE '%@%' OR Email_Maison LIKE 'INCONNU')
);

CREATE TABLE DRH.Employes (
  Nom text NOT NULL,
  Prenom text NOT NULL,
  Entreprise text NOT NULL,
  Bureau text NOT NULL,
  Portable_Travail integer NOT NULL DEFAULT 0,
  Email_Travail text NOT NULL,
  PRIMARY KEY (Nom, Prenom),
  FOREIGN KEY (Nom) REFERENCES DRH.Maisons ON DELETE CASCADE,
  FOREIGN KEY (Entreprise, Bureau) REFERENCES DRH.Societes ON DELETE CASCADE,
  CONSTRAINT PortableTravail CHECK(Portable_Travail > 100000000 OR Portable_Travail = 0),
  CONSTRAINT EmailTravail CHECK(Email_Travail LIKE '%@%')
);
```

***Question 2.5 (1,5 point)*** *Créer une vue* `AdressesEquiv` *à partir de votre décomposition qui contiennent exactement les tuples de la relation* `Adresses` *initiale.*

**Réponse :**

```
CREATE VIEW DRH.AdressesEquiv AS
SELECT Employes.Nom, Employes.Prenom, Employes.Entreprise, Employes.Bureau,
       Tel_Bureau, Portable_Travail, Email_Travail, Tel_Maison, Email_Maison
FROM   DRH.Employes NATURAL JOIN DRH.Maisons NATURAL JOIN DRH.Societes;
```

**Exercice 3 (Reprise après une panne (4 points))** *Nous supposons que le SGBD écrit dans un journal le début, les opérations et la fin de chaque transaction, et que chaque écriture est aussitôt sauvegardée sur disque. Nous supposons également que le SGBD utilise la technique des points de synchronisation : à intervalles réguliers le SGBD écrit dans le journal la liste des transactions en cours, puis aussitôt sauvegarde le journal et la base de données sur le disque dur.*
*Soit pc le dernier point de synchronisation réalisé à une date tc, et une panne du SGDB à la date* $tf > tc$.

***Question 3.1 (1 point)*** *Donner les objectifs de l'algorithme de reprise après la panne.*

**Réponse :** *L'objectif principal est la reprise du service. Cette reprise doit pénaliser le moins possible les applications clientes. Elle doit donc remettre la base dans un état cohérent vis à vis de l'information connue des clients, ainsi :*
- *Un client ne doit pas avoir à refaire une transaction terminée avant la panne.*
- *Un client doit avoir à refaire une transaction non terminée avant la panne.*

***Question 3.2 (3 points)*** *Donner l'algorithme de reprise après la panne.*

**Réponse :** *(cf pages 53 et 54 du polycopie)*