

# Maîtrise d'informatique 2001-02

## Langage SQL

### Examen final - janvier 2002

#### Durée de l'épreuve: 2 heures

L'usage des photocopiés du cours est autorisé. Tous les autres documents, sous quelque forme que ce soit (papier, électronique ou autre) sont interdits, y compris les énoncés et corrigés des exercices des TD.

Le barème est donné entre crochets au début de chaque question.

**1** [5 pts] Soit la relation:

Séance-Cours(Enseignement, Date, Enseignant, Salle, Etudiant, Diplôme)

dans laquelle:

- **Enseignement** correspond à un code identifiant chaque enseignement de l'université de façon unique ;
- **Date** correspond à une heure de début de séance et un jour de semaine (on considère un emploi du temps hebdomadaire) ;
- **Enseignant** correspond au nom de l'enseignant (on considère qu'il n'y a pas de doublons) ;
- **Salle** identifie de façon unique une salle de l'université ;
- **Etudiant** correspond au numéro étudiant ;
- **Diplôme** correspond à un nom de diplôme (exemple: Maîtrise Informatique).

On considère les hypothèses suivantes:

- Un enseignement donné n'est assuré que par un seul enseignant.
- A une date donnée, dans une salle donnée n'a lieu qu'un seul enseignement.
- Un enseignant ne peut enseigner que dans une seule salle à une date donnée, de la même manière qu'un étudiant ne peut se trouver que dans une seule salle à une date donnée.
- Il ne peut y avoir plus d'un enseignant par salle à une date donnée.
- Un étudiant peut être inscrit à la préparation de plusieurs diplômes mais chaque enseignement qu'il suit n'entre dans la préparation que d'un seul diplôme.

Questions:

- Combien y-a-t-il de clés candidates pour la relation Séance-Cours? Donnez-les toutes. Justifiez votre réponse.

- Quelle clé doit-on choisir pour cette relation? Justifiez votre réponse.
- Cette relation est-elle en BCNF? Si ce n'est pas le cas:
  - Indiquez pourquoi la relation n'est pas en BCNF
  - Donnez un schéma relationnel en BCNF
  - Indiquez (précisément) s'il y a eu perte de données et/ou de dépendances.

2 [11 pts] Une entreprise imprime des factures du type suivant :

Le 25 septembre 2000

Facture numéro 2800 Client : Martin                      Vendeur : 52

Nom du Produit	Quantité	Prix unitaire	Prix
Table	20	500	10000
Stylo	15	100	1500
Prix total			11500

Pour enregistrer les informations contenues dans ces factures, une base de données relationnelle contient les tables suivantes (clés primaires en italiques) :

Facture(*NumFacture*, DateFacture, NumClient, MatrVendeur)  
 Produit(*NumProduit*, NomProduit, TypeProduit, PrixProduit)  
 Client(*NumClient*, NomClient, AdresseClient)  
 LigneFacture(*NumFacture*, *NumProduit*, QuantiteCommandee)  
 Vendeur(*MatrVendeur*, NomVendeur)

La signification des différents attributs est indiquée par leur nom. Tous les numéros, matricules et quantités sont des entiers d'au plus 5 chiffres. "PrixProduit" est un nombre à 2 décimales.

Donnez l'ordre SQL (ou les ordres SQL si ne pouvez y arriver avec un seul ordre SQL) pour

- (a) [2 pts] créer la table **Facture**. Vous n'oublierez pas les contraintes d'intégrité (avec la bonne syntaxe, SVP).
- (b) [1 pt] enlever la deuxième ligne de la facture ci-dessus (les 15 stylos).
- (c) [1 pt] afficher les numéros des clients pour lesquels on n'a jamais établi de facture.
- (d) [1 pt] afficher les numéros des factures qui comportent au moins un produit de chacun des types de produits (colonne **TypeProduit**). Pour fixer les idées : un type de produit est, par exemple, "Papeterie" ou "Informatique".
- (e) [1 pt] afficher les noms des clients qui ont été facturés par le vendeur nommé "Durand" (on supposera qu'il n'y a qu'un seul vendeur qui a ce nom).
- (f) [1 pt] afficher le nombre de produits de nom "Table" qui ont été vendus par le vendeur nommé "Durand".

- (g) [1 pt] créer une vue **TotalFacture** qui comporte les colonnes de la table **Facture** et en plus le prix total de la facture.
- (h) [1 pt] afficher le chiffre d'affaire de chaque vendeur (c'est-à-dire la somme de toutes les factures relatives aux ventes effectuées par ce vendeur). Les vendeurs seront repérés par leur matricule.
- (i) [1 pt] afficher le nom du produit de la facture numéro 1235 qui correspond au prix unitaire le plus important. Par exemple, s'il s'agissait de la facture 2800 ci-dessus, il faudrait faire afficher "Table".
- (j) [1 pt] afficher le type de produit qui comporte le plus de produits différents dans la table **Produit**.

**3** [4 pts] Attention, les 2 questions sont liées. Lisez donc cet exercice jusqu'à la fin avant de rédiger la première question.

- (a) [3 pts] Écrivez une méthode **save** de la classe Java **Produit**, qui sauvegarde dans la base de données les informations sur une instance de la classe. La classe **Produit** est la classe qui correspond à la table **Produit** de la base de données.

Obligatoire :

- vous utiliserez des requêtes paramétrées ;
- votre méthode devra voir si le produit est déjà dans la base ou non (il est repéré par sa clé primaire) et agir en conséquence ;
- la transaction devra être validée dans la méthode.

On ne vous demande pas d'écrire toute la classe **Produit**. On vous demande seulement d'écrire le code de la méthode **save** et les déclaration des variables d'instance que vous utiliserez. Pour ces variables, n'oubliez pas les modificateurs (d'accès, de classe,...). Vous pouvez éventuellement ajouter un autre code si ça vous facilite l'écriture de la méthode **save**.

Si vous récupérez la connexion à la base dans la méthode **save** (voir question suivante), vous utiliserez la classe Java **Connexion** qui contient une méthode de classe **getConnexion()** (sans paramètre) qui renvoie une connexion à la base (classe **Connection**). On ne vous demande ni d'écrire cette méthode **getConnexion()**, ni d'indiquer comment cette instance de **Connection** est obtenue. On suppose que tout ceci est fait dans une autre partie du code.

- (b) [1 pt] En quelques lignes (soyez court et précis), donnez des raisons qui pourraient influencer le choix entre :
  - récupérer les connexions à la base en dehors de la méthode **save** ;
  - récupérer une connexion à chaque appel de la méthode **save**.

Indiquez pourquoi vous avez fait le choix qui correspond au code de la question précédente.

# Maîtrise Informatique 2001-2002

## Introduction aux SGBD

Correction des exercices de l'examen de janvier 2002

2

- (a) `create table facture`  
`(numfacture integer constraint pk_facture primary key,`  
`datefacture date,`  
`numclient integer constraint ref_fact_client references client,`  
`matrvendeur integer constraint ref_fact_vendeur references vendeur)`
- (b) `delete from LigneFacture`  
`where numFacture = 2800`  
`and numProduit = (select numProduit from Produit`  
`where nomProduit = 'Stylo')`
- (c) `select numClient from Client`  
`where numClient not in (select numClient from Facture)`
- (d) Division de (LigneFacture J{numProduit} Produit)[numFacture, typeProduit] par Produit[typeProduit]

En SQL :

```
select numFacture from Facture
where not exists
  (select null from Produit
   where not exists
     (select null from LigneFacture, Produit
      where LigneFacture.numProduit = Produit.numProduit
        and numFacture = Facture.numFacture
        and typeProduit = Produit.typeProduit))
```

- (e) `select nomClient from Client`  
`where numClient in`  
`(select numClient from Facture`  
`where matrVendeur =`  
`(select matrvendeur from Vendeur`  
`where nomVendeur = 'Durand'))`
- (f) `select sum(quantiteCommandee) from LigneFacture`  
`where numFacture in`  
`(select numFacture from Facture`  
`where matrVendeur =`  
`(select matrvendeur from Vendeur`  
`where nomVendeur = 'Durand'))`  
`and numProduit = (select numProduit from Produit`  
`where nomProduit = 'Table')`

- (g) create view TotalFacture  
 (numFacture, dateFacture, numClient, matrVendeur, totalFacture) as  
 select numFacture, datefacture, numClient, matrVendeur,  
       sum(quantiteCommandee \* prixProduit)  
 from Facture, LigneFacture, Produit  
 where Facture.numFacture = LigneFacture.numFacture  
       and Produit.numProduit = LigneFacture.numProduit  
 group by numFacture, dateFacture, numClient, matrVendeur
- (h) select matrVendeur, sum(prixTotal) from TotalFacture  
 group by matrVendeur  
 ou  
 select matrVendeur, sum(quantiteCommandee \* prixProduit)  
 from Facture, LigneFacture, Produit  
 where Facture.numFacture = LigneFacture.numFacture  
       and Produit.numProduit = LigneFacture.numProduit
- (i) select distinct nomProduit from Produit, LigneFacture  
 where Produit.numProduit = LigneFacture.numProduit  
       and numFacture = 1235  
       and prixProduit = (select max(prixProduit)  
                           from LigneFacture, Produit  
                           where LigneFacture.numProduit = Produit.numProduit  
                           and numFacture = 1235)
- (j) select typeProduit from Produit  
 group by typeProduit  
 having count(\*) =  
       (select max(count(\*)) from Produit  
       group by typeProduit)

**3**

- (a) private int numero;  
 private String nom;  
 private String type;  
 private double prix;
- private Connection connexion;
- private static PreparedStatement pstmtQuery = connexion.prepareStatement(  
 "select numProduit from Produit where numProduit = ?");  
 private static PreparedStatement pstmtInsert = connexion.prepareStatement(  
 "insert into Produit(numProduit, nomProduit, typeProduit, prixProduit)"  
 + " values (?, ?, ?, ?)";  
 private static pstmtUpdate = connexion.prepareStatement(  
 "update Produit"  
 + " set numProduit = ?"  
 + " set nomProduit = ?, set typeProduit = ?, set prixProduit = ?)";

```

public void save() {
    PreparedStatement pstmt;
    // On regarde si le produit est déjà dans la base
    pstmtQuery.setInt(numero);
    ResultSet rset = pstmt.executeQuery();
    if (rset.next()) {
        // Le produit est déjà dans la base
        pstmt = pstmtUpdate;
    }
    else {
        // Le produit n'existe pas dans la base
        pstmt = pstmtInsert;
    }
    pstmt.setInt(numero);
    pstmt.setString(nom);
    pstmt.setString(type);
    pstmt.setDouble(prix);
    pstmt.executeUpdate();

    connexion.commit();
}

```

(b) Des raisons pour le choix :

- Coût d'une connexion (en tant que ressource) ;
- Temps de connexion et charge pour le processeur ;
- Est-ce qu'on utilise un pool de connexions?
- Est-ce que les traitements se feront par lots ou produit par produit?

Le seul argument pour obtenir une connexion à chaque **save** serait de faciliter la tâche du programmeur car rien n'empêche de récupérer une connexion et de la fermer à chaque appel de la méthode **save** mais en dehors de la méthode.

Il vaut donc mieux laisser le choix aux clients de la classe. Il ne faut donc pas acquérir une nouvelle connexion à chaque sauvegarde et laisser l'acquisition de la connexion à l'extérieur de la méthode **save**.