

EXAMEN SEMESTRE 2

Matière : INFORMATIQUE

Classes : MP2, PC2 et PT2

Durée : 2 h

Exercice 1 (4.5 pts)

Soit les tables Employés et Départements suivantes :

Tables :

- **Employés** (id, nom, poste, salaire, département_id)
- **Départements** (id, nom, localisation)

1. Traduire les expressions relationnelles en SQL :

- $\sigma_{salaire > 50000}(Employés)$
- $\Pi_{nom, poste}(Employés)$
- $\sigma_{salaire > 60000}(Employés) \cap \sigma_{département_id \neq IT}(Employés)$
- $\sigma_{localisation = "Paris"}(Départements) \cup \sigma_{localisation = "Lyon"}(Départements)$
- $\Pi_{Employés.nom, Départements.nom}(Employés \bowtie_{département_id=id} Départements)$

2. Répondre par Vrai ou Faux :

- SELECT nom FROM Employés ORDER BY nom ASC renvoie les noms triés de Z à A.
- SELECT COUNT(*) FROM Employés compte les employés même si salaire est NULL.
- SELECT * FROM Employés WHERE poste = 'Manager' AND salaire > 70000 liste les managers gagnant plus de 70k.
- SELECT DISTINCT poste FROM (SELECT * FROM Employés UNION SELECT * FROM Employés) élimine les doublons.

Exercice 2 (6 pts)

On dispose d'un fichier texte ventes.txt contenant les enregistrements des ventes effectuées par une entreprise sous la forme suivante :

Produit1	2024-01-15	15	250.50
Produit2	2024-01-16	7	100.75
Produit3	2024-01-18	3	75.00
Produit1	2024-01-20	5	250.50
Produit2	2024-01-22	10	100.75

Chaque ligne contient **nom du produit**, **date de vente**, **quantité vendue**, et **prix unitaire** séparés par des tabulations \t.

1. Écrire une fonction `lire_ventes(fichier)` qui lit le fichier et retourne un dictionnaire où chaque clé est le **nom du produit** et la valeur est une liste de tuples (date, quantité, prix). Cette fonction permet d'obtenir le dictionnaire suivant :

```
{ "Produit1": [("2024-01-15", 15, 250.50), ("2024-01-20", 5, 250.50)],  
  "Produit2": [("2024-01-16", 7, 100.75), ("2024-01-22", 10, 100.75)],  
  "Produit3": [("2024-01-18", 3, 75.00)] }
```

2. Écrire une fonction `calculer_chiffre_affaires(dico)` qui retourne un dictionnaire associant chaque produit à son **chiffre d'affaires total**. Exemple de sortie :

```
{ "Produit1": 5000.0,  
  "Produit2": 1712.75,  
  "Produit3": 225.0 }
```

Chiffre d'affaires = $\sum(\text{quantité} \times \text{prix unitaire})$

3. Écrire une fonction `produit_plus_vendu(dico)` qui retourne le nom du produit ayant généré le **plus grand chiffre d'affaires**.
4. Écrire une fonction `ventes_par_date(dico, date)` qui retourne une liste des produits vendus à une date donnée avec leurs quantités et chiffre d'affaires respectif.

Exercice 3 (6 pts)

Soit une base de données avec les tables :

- ✓ Universités : qui stocke les informations générales sur les universités.
- ✓ Etudiants : qui contient la liste des étudiants et leur université d'appartenance.
- ✓ Cours : qui contient le catalogue des cours offerts par les universités.
- ✓ Inscriptions : qui enregistre les inscriptions des étudiants aux cours et leurs résultats.

Le schéma relationnel de la base de données est le suivant :

Universités :

- 'nom' (clé primaire) : Nom de l'université (ex: "Sorbonne").
- 'pays' : Pays où se trouve l'université (ex: "France").
- 'nb_étudiants' : Nombre total d'étudiants inscrits.

Étudiants :

- 'id' (clé primaire) : Identifiant unique de l'étudiant.
- 'nom' : Nom de l'étudiant (ex: "Jean Dupont").
- 'université' (clé étrangère vers Universités.nom) : Université où l'étudiant est inscrit.
- 'année_inscription' : Année d'inscription (ex: 2020).

Cours :

- 'code' (clé primaire) : Code unique du cours (ex: "INF101").
- 'titre' : Titre du cours (ex: "Introduction à l'informatique").
- 'crédits' : Nombre de crédits académiques (ex: 5).
- 'université' (clé étrangère vers Universités.nom) : Université proposant le cours.

Inscriptions :

- 'étudiant_id' (clé étrangère vers Étudiants.id) : Identifiant de l'étudiant.
- 'cours_code' (clé étrangère vers Cours.code) : Code du cours suivi.
- 'note' : Note obtenue (ex: 15.5).

Questions : Exprimer en SQL :

1. Liste des universités ayant plus de 20 000 étudiants.
2. Liste des cours de plus de 5 crédits proposés par les universités situées en France.
3. Les étudiants inscrits à au moins 3 cours.
4. La moyenne des notes par université.

Exercice 4 (3.5 pts)

Lors de la transmission de données numériques, il peut y avoir des erreurs dues à des interférences, des pertes de signal, ou des défaillances matérielles. Pour garantir l'intégrité des données, on utilise des **techniques de détection et de correction d'erreurs**, comme le **bit de parité** et le **code de Hamming**.

Bit de Parité (Détection d'erreurs)

Le **bit de parité** est un mécanisme simple permettant de vérifier si une donnée binaire a été altérée lors de sa transmission.

Principe :

- On compte le nombre de 1 dans une séquence binaire.
- Il existe deux types de parité :
 - **Parité paire** : On ajoute un bit 0 si le nombre de 1 est pair, sinon on ajoute un 1 pour rendre la somme paire.
 - **Parité impaire** : On ajoute un 1 si le nombre de 1 est pair, sinon on ajoute un 0.

Exemple de bit de parité (parité paire) :

Nombre	Valeur binaire	Nombre de 1	Bit de parité (paire)
5	101	2 (pair)	0
7	111	3 (impair)	1
19	10011	3 (impair)	1
45	101101	4 (pair)	0

Code de Hamming (7,4) – Correction d'erreurs

Le **code de Hamming** est un **code détecteur et correcteur d'erreurs**. Il permet :

- De **détecter** jusqu'à **deux erreurs** dans un mot binaire.
- De **corriger** une **seule erreur** sans retransmission.

Nous allons travailler avec le **code (7,4)** :

- On encode **4 bits** en **7 bits** en ajoutant **3 bits de parité**.
- Ces bits de parité permettent d'identifier et de corriger une erreur unique.

Encodage :

Soit une séquence **4 bits** : (d1, d2, d3, d4).

On calcule **3 bits de parité** :

- p1 vérifie les bits (d1, d2, d4).
- p2 vérifie les bits (d1, d3, d4).
- p3 vérifie les bits (d2, d3, d4).

Le message encodé est donc : (p1, p2, d1, p3, d2, d3, d4)

Exemple avec $d = [1, 0, 1, 1]$:

- $p1 = \text{parité}(1, 0, 1) = 0$
- $p2 = \text{parité}(1, 1, 1) = 1$
- $p3 = \text{parité}(0, 1, 1) = 0$

Message encodé : $[0, 1, 1, 0, 0, 1, 1]$

Décodage et Correction d'erreurs

Lorsqu'un **message encodé (7 bits)** est reçu, il peut contenir **une erreur**.

On utilise trois **bits de contrôle** ($c1, c2, c3$) pour identifier la position de l'erreur.

- $c1 = \text{parité}(m4, m5, m6, m7)$
- $c2 = \text{parité}(m2, m3, m6, m7)$
- $c3 = \text{parité}(m1, m3, m5, m7)$

Cas possibles :

1. Si $c1 = c2 = c3 = 0$, pas d'erreur.
2. Sinon, les bits $c1, c2, c3$ donnent la position binaire de l'erreur.
 - Ex : $(c1, c2, c3) = (0, 1, 1)$ signifie **erreur au 3e bit**.
 - On **corrige l'erreur** en inversant le bit erroné.

Questions :

1. Écrire une fonction `calculer_parite(bits)` qui prend une **liste de bits** et retourne 0 ou 1 en fonction du **bit de parité**.
2. Écrire une fonction `encode_hamming(data)` qui prend une **liste de 4 bits** [$d1, d2, d3, d4$] et retourne une **liste de 7 bits encodés**.
3. Écrire une fonction `decode_hamming(m)` qui prend une **liste de 7 bits** et retourne les **4 bits originaux**. Si une erreur est détectée, elle doit :
 - Afficher la **position de l'erreur**.
 - **Corriger l'erreur** si elle est unique.