

Exercice 1 : (Sujet zéro (2) ECRICOME 2023) Sur le marché des véhicules d'occasion, on observe en général une baisse du prix de revente (ou décote) d'un véhicule lorsque le nombre de kilomètres parcourus augmente. Une bonne estimation de cette baisse de prix permet au vendeur de fixer avec précision le prix de revente d'un véhicule.

On dispose d'une base de données comportant deux tables `vehicule` et `annonce` décrites ci-dessous.

- La table `vehicule` recense des informations sur les modèles de véhicules en vente sur le marché. Elle est composée des attributs suivants :
 - `id_vehicule` (de type `INTEGER`) : un code permettant d'identifier de façon unique chaque référence de véhicule (marque et modèle).
 - `marque` (de type `TEXT`) : le nom du constructeur du véhicule.
 - `modele` (de type `TEXT`) : le modèle du véhicule, un constructeur proposant en général plusieurs modèles de véhicules à la vente.
 - `prix_neuf` (de type `INTEGER`) : prix de vente du véhicule neuf.
 - La table `annonce` regroupe des informations sur un grand nombre d'annonces de véhicules d'occasion. Chaque enregistrement correspond à une annonce et possède les attributs suivants.
 - `id_annonce` (de type `INTEGER`) : un code permettant d'identifier chaque annonce de façon unique.
 - `id_vehicule` (de type `INTEGER`) : l'identifiant du modèle de véhicule vendu, qui correspond à l'identifiant utilisé dans la table `vehicule`.
 - `annee` (de type `INTEGER`) : année de première mise en circulation du véhicule.
 - `km` (de type `INTEGER`) : nombre de kilomètres parcourus par le véhicule au moment de la revente.
 - `prix_occasion` (de type `INTEGER`) : prix de vente du véhicule d'occasion.
1. En justifiant brièvement, identifier une clef primaire dans chacune des tables `vehicule` et `annonce`, ainsi qu'une clef étrangère dans la table `annonce`.
 2. Écrire une requête SQL permettant d'extraire les noms de tous les modèles de véhicules mis en vente par le constructeur `Dubreuil Motors`.
 3. Expliquer le fonctionnement de la requête SQL suivante et préciser l'effet éventuel de cette requête sur chacune des tables `vehicule` et `annonce`.

```
1 UPDATE annonce
2 SET prix_occasion = prix_neuf
3 FROM vehicule
4 WHERE vehicule . id_vehicule = annonce . id_vehicule
5 AND vehicule . prix_neuf < annonce . prix_occasion
```

4. A l'aide d'une jointure, écrire une requête SQL permettant d'obtenir, sur une même table, la liste de toutes les annonces de la table `annonce` avec les attributs suivants :
 - l'identifiant de l'annonce `id_annonce`.
 - le kilométrage `km`.
 - le prix de vente du véhicule neuf `prix_neuf`.
 - le prix de vente d'occasion `prix_occasion`.

Exercice 2 : On considère la table ci-dessous, nommée `table_1` :

élèveId	Nom_et_prénom	Moyenne
1	élève1	12,4
2	élève2	8,9
3	élève3	4,7
4	élève4	16,9
5	élève5	2,2

1. Que renvoie la requête suivante ?

```

1 SELECT Nom_et_prénom
2 FROM table_1
3 WHERE Moyenne > 10

```

2. Que fait la requête suivante ?

```

1 DELETE FROM table_1
2 WHERE Moyenne >= 16 or Moyenne <= 4

```

Exercice 3 : On s'intéresse à une base de données cinématographiques.

Plus précisément, on veut manipuler :

- Une table `Comedien-ne-s`, avec les attributs suivants :
 - `id_comédien-ne`, un nombre entier (de type `INTEGER`), identifiant le ou la comédien·ne ;
 - `Nom`, de type `TEXT` ;
 - `Pays_acteur`, de type `TEXT` précisant le pays de provenance ;
 - `Age`, de type `INTEGER` ;
 - `Sexe`, de type `TEXT` (vaudra M ou F) ;
 - `Taille`, de type `INTEGER` (exprimée en cm) ;
- Une table `Films`, avec les attributs suivants :
 - `id_film`, de type `INTEGER`, un nombre identifiant le film ;
 - `Titre`, de type `TEXT` ;
 - `Role_principal`, de type `INTEGER`, qui contiendra le numéro identifiant le ou la comédien·ne dans la table `Comedien-ne-s` ;
 - `Année`, de type `INTEGER` ;
 - `Durée`, de type `INTEGER` (exprimée en minutes) ;
 - `Pays_Film`, de type `TEXT`, précisant le pays d'origine du film.

1. Quelles clés primaires et clés étrangères doit-on déclarer sur ces tables ? (On précisera vers quoi doivent pointer les clés étrangères).
2. On suppose la table `Comedien-ne-s` créée. Donner une syntaxe SQL permettant de créer la table `Films`, en tenant compte de la réponse à la question précédente.
3. Donner une requête SQL permettant de lister les titres des films français réalisés avant 1985.
4. Donner une requête SQL permettant de calculer la durée moyenne des films norvégiens.
(On admettra que la commande `AVG` permet de calculer la moyenne des valeurs présentes sur une même colonne)
5. Donner une requête SQL permettant d'afficher les titres et durées des films français et des films japonais, triés par ordre chronologique, du plus ancien au plus récent.
(On admettra que la commande `ORDER BY` permet de trier les résultats dans l'ordre croissant)

On utilise maintenant les deux tables.

6. Écrire une jointure de ces deux tables, avec la condition qui semble pertinente.
7. Donner une requête SQL permettant d'afficher les titres et années des films italiens dont le rôle principal est tenu par un ou une comédien·ne français·e.
8. Donner une requête SQL permettant d'afficher le nombre de films dont le rôle principal est tenu par une comédienne. (On admettra que la commande `count(*)` compte le nombre de lignes de la table renvoyée)

On veut maintenant enrichir les informations, en répertoriant plusieurs acteurs qui jouent dans un même film. On adopte pour cela le modèle suivant :

- La table `Comedien-ne-s` est inchangée.
- Dans la table `Films`, on retire la colonne `Role_principal`.
- On crée une nouvelle table `JoueDans`, contenant 2 colonnes `Comédien-ne` et `Film`, dont chaque enregistrement signale qu'un·e comédien·ne joue dans un film. Dans cette table, comédien·nes et films sont représenté·es par leurs identifiants des tables précédentes.

Par exemple, si cette table contient les lignes :

JoueDans	
Comédien-ne	Film
⋮	⋮
1	2
3	2
1	4
⋮	⋮

cela signifie que le ou la comédien·ne tel·le que

`Comedien-ne-s.id_comédien-ne = 1`

joue dans le film tel que

`Films.id_Film = 2`

etc

9. La table `JoueDans` contient-elle des clés primaires ?
10. La table `JoueDans` contient-elle des clés étrangères ? Si oui, vers quoi pointent-elles ?
11. Donner les noms de tou·te·s les comédien·nes jouant dans des films français. On fera en sorte d'éviter qu'un·e même comédien·ne apparaisse plusieurs fois dans les résultats.

(On admettra que la commande `distinct` permet de supprimer tous les doublons d'une table)

Pour cela on aura à effectuer une double jointure pour relier les 3 tables. La syntaxe consiste à enchaîner à la suite deux commandes de jointure :

1	<code>Table1</code>
2	<code>[jointure avec Table2 avec la condition]</code>
3	<code>[jointure avec Table 3 avec la condition]</code>

Exercice 4 : On considère la base de données d'une entreprise vendant des donuts, dont le schéma relationnel est donné par :

- **Donuts** (id_donut, glaçage, poids, prix)
- **Clients** (id_client, nom, prénom, adresse, mail, tél)
- **Commandes** (id_commande, donut, client, date, quantité, prix_total)

1. Identifier les clés primaires et les éventuelles clés étrangères de chacune des tables.
2. Le prix d'un donut (en euros) peut être l'un des nombres suivants : 2, 3, 4, 5. Déterminer la liste des donuts qui font partie des deux types de donuts les plus chers.
3. (a) Déterminer le nombre de commandes effectuées dans la journée.
(On admettra que la commande `getdate()` renvoie la date de la journée actuelle et que la commande `count(*)` compte le nombre de lignes de la table renvoyée)
(b) Déterminer le nombre total de donuts vendus dont le glaçage est au « chocolat ».
(On admettra que la commande `sum` permet de faire la somme de toutes les lignes sur chaque colonne)
4. Déterminer tous les clients (identifiant, nom, prénom) qui ont acheté un donut dont le poids est supérieur ou égal à 60 grammes (on supposera que le poids est donné en grammes dans la table **Donuts**).
On pourra réaliser une requête imbriquée, de la forme :

```
1  SELECT *
2  FROM table1
3  WHERE attribut1 IN (
4      SELECT attribut2
5      FROM table2
6  )
```