

On dispose d'une base de données comportant les tables suivantes :

```
clients
    (idClient, designation, adresse, cp, ville, email, tel)
produits
    (idProduit, designation, prix, idEntrepot)
commandes
    (idCommande, date, idProduit, idClient, quantite)
entrepots
    (idEntrepot, designation, adresse, cp, ville, superficie)
```

Le nom des champs (entre les parenthèses) est choisi de façon implicite pour qu'il n'y ait pas d'ambiguïté.

1 La requête :

```
SELECT * FROM clients ORDER BY ville DESC
```

- a** affiche le nom de tous les clients par ordre alphabétique de leur ville ;
- b** affiche le nom de tous les clients selon un ordre alphabétique inverse de leur ville ;
- c** affiche tous les champs de la table « clients » par ordre alphabétique inverse de leur nom ;
- d** affiche tous les champs de la table « clients » par ordre alphabétique inverse du nom de leur ville.

2 La requête :

```
SELECT *
FROM (
    clients INNER JOIN entrepots
    ON clients.ville = entrepots.ville)
```

- a** affiche tous les champs des tables « clients » et « entrepots » qui ont la même ville ;
- b** affiche tous les champs de la table « clients » où le contenu du champ « ville » est identique à celui d'au moins un champ de la table « entrepots » ;
- c** affiche tous les champs de la table « entrepots » où le contenu du champ « ville » est identique à celui d'au moins un champ de la table « clients » ;
- d** affiche les contenus du champ « ville » communs aux tables « entrepots » et « clients ».

3 La requête :

```
SELECT *  
FROM (  
    clients INNER JOIN entrepots  
    ON clients.ville = entrepots.ville)
```

est équivalente à :

- a** SELECT * FROM clients
WHERE clients.ville = entrepots.ville
- b** SELECT * FROM entrepots
WHERE clients.ville = entrepots.ville
- c** SELECT * FROM clients, entrepots
WHERE clients.ville = entrepots.ville
- d** SELECT ville FROM clients, entrepots
WHERE clients.ville = entrepots.ville

4 La requête :

```
SELECT *  
FROM  
    clients LEFT OUTER JOIN entrepots  
    ON (clients.ville = entrepots.ville)
```

- a** affiche uniquement les clients dont la ville coïncide avec celle d'un entrepôt;
- b** affiche tous les clients;
- c** affiche tous les entrepôts;
- d** n'affiche rien.

5 La requête :

```
SELECT  
    SUM(commandes.total) AS somme  
FROM  
    commandes  
WHERE  
    commandes.date > '2019-09-02'
```

- a** affiche le total des commandes effectuées après le 2019-09-02;
- b** affiche le total de chaque commande effectuée après le 2019-09-02;
- c** affiche toutes les entrées de la table « commandes » où la date est supérieure à 2019-09-02 en effectuant la somme des totaux petit à petit;
- d** n'affiche rien.

Les affirmations suivantes sont-elles vraies ou fausses ? Justifier la réponse.

On se place dans une base de données nommée « bddPerso ».

- 1 Une ligne d'une table de bddPerso est appelée une entrée.
- 2 Une table de bddPerso est définie par la structure relationnelle suivante :

```
| clients (idClient , nom , prenom)
```

Alors « nom » est un attribut de la table « clients ».

- 3 Dans la table « clients » de la question précédente, on peut prendre « nom » comme clé primaire.
- 4 On définit une autre table de la manière suivante :

```
| connexions (idCommande , idClient , date)
```

où « idClient » fait référence à la clé du même nom de la table « clients ».

« idClient » est alors une clé étrangère de la table « connexions ».

3 Festival de musique



On dispose d'une base de données d'un festival de musique, où il y a plusieurs représentations. Un ou plusieurs musiciens peuvent participer à une représentation, mais un musicien ne peut participer qu'à une seule représentation.

La structure de la base de données est la suivante :

```
| Representation (idRep , titreRep , lieu)
```

```
| Musicien (idMus , nom , idRep)
```

```
| Programmer (Date , idRep , tarif)
```

Écrire la requête SQL permettant d'afficher :

- 1 La liste des titres des représentations.
- 2 La liste des titres des représentations ayant lieu sur le lieu nommé « Dionysos ».
- 3 La liste des noms des musiciens et les titres des représentations auxquelles ils participent.
- 4 La liste des titres des représentations, les lieux et les tarifs d'une date précisée.
- 5 Le nombre des musiciens qui participent à la représentation numéro 7.
- 6 Les représentations et leurs dates dont le tarif ne dépasse pas 30 €.

4 Les employés du département



15 min

Corrigé
p. 258

On dispose d'une base de données dont la structure est la suivante :

```
Departements (DNO, DNOM, DIR, VILLE)
Employes (ENO, ENOM, PROF, DATEEMB, SAL, COMM, DNO)
```

Donner les requêtes SQL qui permettent d'obtenir :

- 1 la liste des employés ayant une commission (attribut « COMM » déclaré comme booléen) ;
- 2 les noms, emplois et salaires des employés obtenus en classant les emplois dans l'ordre alphabétique, et pour chaque emploi, en classant les salaires du plus grands au plus petit ;
- 3 le salaire moyen des employés.
- 4 le salaire moyen dans le département nommé « Production » (DNOM = 'Production').

5 Obtenir la ligne d'un maximum



10 min

Corrigé
p. 259

On dispose de la table :

```
commandes (idCommande, date, idClient, montant)
```

Quelle requête permet d'afficher la (les) enregistrements où le prix est le plus grand ?

6 Notes annuelles des étudiants



20 min

Corrigé
p. 260

On considère le modèle relationnel suivant concernant la gestion des notes annuelles d'une promotion d'étudiants :

```
ETUDIANT
  (NumEtudiant, Nom, Prenom)
MATIERE
  (CodeMat, LibelleMat, CoeffMat)
EVALUER
  (NumEval, #NumEtudiant, #CodeMat, Date, Note)
```

Les clés étrangères sont précédées d'un « # » et les clés primaires sont soulignées.

Exprimez en SQL les requêtes suivantes.

- 1 Quel est le nombre total d'étudiants ?
- 2 Quelles sont, parmi l'ensemble des notes, la note la plus haute et la note la plus basse ?

- 3 Quelles sont les moyennes de chaque étudiant dans chacune des matières ?
- 4 Quelles sont les moyennes de la classe par matière ?
- 5 Quelle est la moyenne générale de chaque étudiant ?
- 6 Quelle est la moyenne générale de la promotion ?
- 7 Quels sont les étudiants qui ont une moyenne générale supérieure ou égale à la moyenne générale de la promotion ?

Aide : on pourra utiliser « GROUP BY » pour plusieurs questions, qui regroupe les entrées. Par exemple, si l'on considère la table suivante :

clients		
id	nom	montant
1	Marisol Pleureur	125.65
2	Jacques Umul	45.7
3	Marisol Pleureur	100.5
4	Jacques Umul	78.6

alors, la requête :

```
SELECT
    nom ,
    SUM(montant) AS total
FROM
    clients
GROUP BY
    nom
```

calculera le total des valeurs des attributs « montant » en les regroupant par nom, et donnera la table :

nom	total
Jacques Umul	124.3
Marisol Pleureur	226.15

7 Dans un lycée



On considère une base de données dont le modèle relationnel est le suivant :

```
eleves (idEleve,nom,prenom,age)
controles (idControle,#idEleve,#idMatiere,date,note)
matieres (idMatiere,nomMatiere,coef)
```

Les champs soulignés sont des clés primaires et les champs marqués d'un « # » désignent des clés étrangères.

- 1 Indiquer une requête SQL permettant d'afficher le nombre de contrôles passés par chaque élève pour une discipline donnée (par exemple, NSI).
Aide : pour afficher des résultats par élève, on peut utiliser GROUP BY. Par exemple :

```
| SELECT * FROM table GROUP BY colonne
```

On souhaite avoir comme résultat une table ayant pour attributs les nom, prénom et nombre de contrôles.

- 2 Créer une requête qui permet d'afficher la moyenne de chaque élève par matière.

On souhaite obtenir une table ayant pour attributs les nom et prénom des élèves, les noms des matières, leurs coefficients et la moyenne de la matière pour chaque élève.

- 3 Créer une requête qui permet d'afficher la moyenne générale de chaque élève en s'aidant de la vue obtenue à l'aide de la requête de la question précédente.

8 Un cas pratique : site web marchand



Un webmaster souhaite créer une base de données nommée « venteAdonf » pour un site internet marchand `vente-a-donf.com` dont la clientèle est uniquement française (pour simplifier les formats des enregistrements).

Dans cette base de données, il devra y avoir les quatre tables suivantes :

- `users (idUser, nom, prenom, email, rue, cp, ville, tel)`
où « cp » désigne le code postal de sa ville de résidence, les autres attributs étant explicites ;
- `modeles (idModele, nomModele)`
- `articles (idArticle, nom, descriptif, idModele, prix)`
où « idModele » est une clé étrangère relative à la colonne `idModele` de la table `modeles`.
- `panier (idPanier, idUser, idArticle, idModele, quantite, total)`
Ici, « total » représente le résultat de « quantite * prix de l'article ».

On décide de mettre en clés primaires les premières colonnes de chaque table.

- 1 Écrire un fichier SQL nommé « `venteAdonf-create.sql` » permettant de créer ces quatre tables.

On rappelle qu'un tel fichier est composé de quatre requêtes séparées par un point-virgule.

- 2 Voici les premiers articles mis en vente :

Nom	Descriptif	Modèle	Tarif
Mug	Magnifique mug personnalisable avec votre photo.	Unique	7,99 €
T-shirt « I Kiffe Beef »	T-shirt unique en son genre.	S / M / L / XL / XXL	24,99 €

Écrire un fichier SQL nommé « venteAdonf-insertArt.sql » permettant d'insérer ces informations dans la base de données.

- 3 Le jour de la mise en ligne du site, deux personnes ont passé commande :

Nom	Dupont	Aimaun
Prénom	Jean	Anne
Email	jdupont@free.fr	amz@free.fr
Rue	3 rue des tulipes	1 rue du glas
Cp	75000	33000
Ville	Paris	Bordeaux
Tel	+33601020304	+33799989796

Écrire un fichier SQL nommé « venteAdonf-insertUsers.sql » permettant d'insérer ces informations dans la base de données.

- 4 Jean Dupont a commandé 2 mugs et 3 tee-shirts (tailles S, M et L). Anne Aimaun a, quant à elle, commandé 1 mug et 1 tee-shirt (taille M).
- Écrire un fichier SQL nommé « venteAdonf-insertPanier.sql » permettant d'insérer ces informations dans la base de données.
 - Écrire une requête permettant d'afficher le montant total du panier de Jean Dupont.
 - Si la table panier n'avait pas de colonne « total », quelle requête permettrait de renvoyer le résultat de la question précédente ?

- 1** Réponse **[d]**. En effet, « `SELECT * FROM clients` » signifie que l'on veut voir tous les champs de la table « clients » et « `ORDER BY ville DESC` » signifie que l'on souhaite effectuer un tri sur le contenu de « ville » mais dans un ordre alphabétique inverse (DESC).
- 2** Réponse **[a]**. « `clients INNER JOIN entrepots` » désigne l'intersection des deux tables et « `ON clients.ville = entrepots.ville` » désigne le critère à prendre en compte pour trouver l'intersection (ici, le nom des villes). « `SELECT *` » signifie que l'on souhaite afficher tout donc la requête affiche tous les champs des deux tables où le nom des villes coïncide.
- 3** Réponse **[c]**. La requête `SELECT * FROM clients,entrepots WHERE clients.ville = entrepots.ville` est explicite : on sélectionne tous les champs des deux tables « clients » et « entrepots » où le contenu des champs « ville » est commun. Elle fait donc exactement la même chose que la requête de la question précédente.
- 4** Réponse **[b]**. La requête `SELECT * FROM clients LEFT OUTER JOIN entrepots ON (clients.ville = entrepots.ville)` prend en compte la table de gauche (LEFT), donc « clients », et affiche toutes ses entrées en commençant par celles où les noms de villes coïncident. Elle affiche ensuite les entrées où le contenu du champ « ville » ne coïncide avec aucun contenu du champ « ville » de la table « entrepots ».
- 5** Réponse **[a]**. La requête `SELECT SUM(commandes.total)` renvoie un tableau d'une ligne et d'une colonne dont la valeur est la somme des entrées de la colonne « total ». Il suffit ensuite de regarder la condition (ici, « `WHERE date > '2019-09-02'` » ainsi que l'alias (ici, « AS somme », qui signifie que le nom de la somme sera accessible via l'appellation de somme).

- 1** *Faux*. Dans une base de données, une ligne d'une table est appelée un *enregistrement*.
- 2** *Vrai*. Les étiquettes des colonnes d'une table sont appelées les attributs de la table.
- 3** *Faux*. Une clé primaire a pour but de repérer de manière unique un enregistrement. On ne peut donc pas prendre un attribut (ici « nom ») qui peut prendre plusieurs fois la même valeur. Il serait ici préférable de prendre « idClient » comme clé primaire, si cet attribut a été créé comme étant un entier auto-incrémenté (par exemple).
- 4** *Vrai*. En effet, toute clé primaire d'une table, recopiée en tant qu'attribut d'une autre table, est appelé *clé étrangère* de cette dernière.

- 1 La liste des titres des représentations est donnée par la requête :

```
SELECT
    titreRep
FROM
    Representation
```

- 2 La liste des titres des représentations ayant lieu sur le lieu nommé « Dionysos » est donnée par la requête :

```
SELECT
    titreRep
FROM
    Representation
WHERE
    lieu = "Dionysos"
```

- 3 La liste des noms des musiciens et les titres des représentations auxquelles ils participent est donnée par la requête :

```
SELECT
    M.nom , R.titreRep
FROM
    Musicien M
INNER JOIN
    Representation R
ON
    R.idRep = M.idRep
```

- 4 La liste des titres des représentations, les lieux et les tarifs d'une date précisée (par exemple le 24/12/2020) est donnée par la requête :

```
SELECT
    R.titreRep ,
    R.lieu ,
    P.tarif
FROM
    Programmer P
INNER JOIN
    Representation R
ON
    P.idRep = R.idRep
WHERE
    P.date = "2020-12-24"
```

- 5 Le nombre des musiciens qui participent à la représentation numéro 7 est donné par la requête :

```
SELECT
    COUNT (*)
FROM
    Musicien
WHERE
    idRep = 7
```

- 6 Les représentations et leurs dates dont le tarif ne dépasse pas 30 € sont données par la requête :

```
SELECT
    R.idRep ,
    R.titreRep ,
    P.Date
FROM
    Representation R
INNER JOIN
    Programmer P
ON
    R.idRep = P.idRep
WHERE
    P.tarif <= 30
```

4 Les employés du département

➔ **Enoncé**
p. 252

- 1 La liste des employés ayant une commission :

```
SELECT
    *
FROM
    Employes
WHERE
    COMM = 1
```

2

MÉTHODE

Pour obtenir une liste de résultats selon un ordre *croissant* ou *décroissant*, on utilise une requête avec le mot-clé **ORDER BY ... ASC** (pour un ordre croissant) et **ORDER BY ... DESC** (pour un ordre décroissant).

Les noms, emplois et salaires des employés par emploi croissant, et pour chaque emploi, par salaire décroissant :

```
SELECT
    ENOM ,
    PROF ,
    SAL
FROM
    Employes
ORDER BY
    PROF ASC ,
    SAL DESC
```

3 Le salaire moyen des employés :

```
SELECT
    AVG ( SAL )
FROM
    Employes
```

4 Le salaire moyen du département « Production » :

```
SELECT
    AVG ( E.SAL )
FROM
    Employes E
INNER JOIN
    Departements D
ON
    E.DNO = D.DNO
WHERE
    D.DNOM = "Production"
```

5 Obtenir la ligne d'un maximum

 **Enoncé**
p. 252

Au prime abord, on pourrait penser que la requête :

```
SELECT
    *, MAX(prix)
FROM
    commandes
```

suffirait.

Mais ce n'est pas le cas car elle affiche uniquement la première ligne de la table, en ajoutant une colonne où est inscrite la valeur maximale du prix, ce qui n'est pas ce que l'on souhaite.

On peut alors penser à la requête :

```
SELECT
    * , MAX(prix)
AS
    m
FROM
    commandes
WHERE
    prix = m
```

mais cela ne fonctionne pas : un message d'erreur apparaît car le m n'est qu'un alias et ne se substitue pas ici à la valeur du maximum.

L'idée consiste donc à insérer une requête après le WHERE dans la requête. La requête demandée est alors :

```
SELECT
    *
FROM
    commandes
WHERE
    prix = (
        SELECT
            MAX(total)
        FROM
            commandes
    )
```

Il est important que le nombre après « WHERE prix = » soit retourné par une requête SQL (ou soit spécifié concrètement, mais nous ne le connaissons pas dans ce cas de figure).

6 Notes annuelles des étudiants

 **Enoncé**
p. 252

1 Le nombre total d'étudiants est donné par la requête :

```
SELECT
    COUNT(*)
FROM
    ETUDIANT
```

- 2 Parmi l'ensemble des notes, la note la plus haute et la note la plus basse sont données par la requête :

```
SELECT
    MIN(Note) AS minimum,
    MAX(Note) AS maximum
FROM
    EVALUER
```

- 3 Les moyennes de chaque étudiant dans chacune des matières sont données par la requête :

```
SELECT
    ETU.NumEtudiant,
    ETU.Nom,
    ETU.Prenom,
    MAT.LibelleMat,
    MAT.CoeffMat,
    AVG(EVA.Note) AS MoyenneMat
FROM
    ETUDIANT AS ETU
    INNER JOIN EVALUER AS EVA
        ON EVA.NumEtudiant = ETU.NumEtudiant
    INNER JOIN MATIERE AS MAT
        ON MAT.CodeMat = EVA.CodeMat
GROUP BY
    ETU.NumEtudiant,
    ETU.Nom,
    ETU.Prenom,
    MAT.LibelleMat,
    MAT.CoeffMat
ORDER BY
    ETU.Nom
```

- 4 Les moyennes de la classe par matière peuvent être données par la requête :

```
SELECT
    MAT.CodeMat ,
    MAT.LibelleMat ,
    AVG(EVA.Note) AS MoyClasse
FROM
    MATIERE AS MAT
    INNER JOIN EVALUER AS EVA
        ON EVA.CodeMat=MAT.CodeMat
```

(suite du script page suivante)

```
GROUP BY
    MAT.CodeMat,
    MAT.LibelleMat
ORDER BY
    MAT.LibelleMat
```

5

Pour obtenir la moyenne générale de chaque étudiant, on va se servir d'une *vue*, celle obtenue avec la requête de la question 3 :

```
CREATE VIEW MoyGenEtu AS
SELECT
    ETU.NumEtudiant,
    ETU.Nom,
    ETU.Prenom,
    MAT.LibelleMat,
    MAT.CoeffMat,
    AVG(EVA.Note) AS MoyenneMat
FROM
    ETUDIANT AS ETU
    INNER JOIN EVALUER AS EVA
        ON EVA.NumEtudiant = ETU.NumEtudiant
    INNER JOIN MATIERE AS MAT
        ON MAT.CodeMat = EVA.CodeMat
GROUP BY
    ETU.NumEtudiant,
    ETU.Nom,
    ETU.Prenom,
    MAT.LibelleMat,
    MAT.CoeffMat
ORDER BY
    ETU.Nom
```

Une vue est une façon de construire une table qui peut servir dans une autre requête.

À l'aide de cette vue, nous pouvons maintenant afficher les moyennes générales par élève :

```
SELECT
    Nom , Prenom ,
    SUM(CoeffMat * MoyenneMat) / SUM(CoeffMat) AS
    moyenne
FROM MoyGenEtu
GROUP BY Nom , Prenom
```

- 6 Pour déterminer la moyenne générale de la promotion, on peut utiliser la vue basée sur la réponse à la question précédente :

```
CREATE VIEW SecondeVue AS
SELECT
    Nom ,
    Prenom ,
    SUM(CoeffMat * MoyenneMat) / SUM(CoeffMat) AS
    moyenne
FROM
    MoyGenEtud
GROUP BY
    Nom ,
    Prenom
```

Il ne reste plus qu'à écrire la requête suivante :

```
SELECT
    AVG ( moyenne )
FROM
    SecondeVue
```

- 7 Pour obtenir les étudiants qui ont une moyenne générale supérieure ou égale à la moyenne générale de la promotion, on peut utiliser une fois de plus la vue SecondeVue :

```
SELECT
    Nom , Prenom , moyenne
FROM
    SecondeVue
WHERE
    moyenne >= (
        SELECT
            AVG( moyenne )
        FROM
            SecondeVue
    )
```

7 Dans un lycée

➔ Enoncé
p. 253

1



MÉTHODE

Une jointure interne permet de lier plusieurs tables (ici, élèves, contrôle et matière). On l'utilise pour ce genre de question.

```

SELECT
    E.idEleve,
    E.nom,
    E.prenom,
    COUNT(*) AS nbControle
FROM
    eleves E
    INNER JOIN controle C ON C.idEleve = E.idEleve
    INNER JOIN matiere M ON M.idMatiere = C.
        idMatiere
WHERE
    M.nomMatiere = 'NSI'
GROUP BY
    E.idEleve,
    E.nom,
    E.prenom

```

- 2 Une requête qui permet d'afficher la moyenne de chaque élève par matière (NomMat, Coef) peut être la suivante :

```

SELECT
    E.nom ,
    E.prenom ,
    M.nom ,
    M.coef ,
    AVG(C.note) AS moyenne
FROM
    controles C
    INNER JOIN
        eleves E ON C.idEleve = E.idEleve
    INNER JOIN
        matieres M ON C.idMatiere = M.idMatiere
GROUP BY
    E.nom ,
    E.prenom ,
    M.nom ,
    M.coef

```

3



MÉTHODE

Lorsqu'une requête semble être compliquée à construire, il est souvent utile de la décomposer en créant une ou plusieurs vues. C'est ici le cas : nous avons créé une première vue dans la question précédente, et nous allons nous en servir pour en créer une autre, qui nous servira dans la requête principale répondant à notre question.

On crée une vue « mavue » à partir de la requête précédente :

```
CREATE VIEW mavue AS
SELECT
    E.nom ,
    E.prenom ,
    M.nom ,
    M.coef ,
    AVG(C.note) AS moyenne
FROM
    controles C
INNER JOIN
    eleves E ON C.idEleve = E.idEleve
INNER JOIN
    matieres M ON C.idMatiere = M.idMatiere
GROUP BY
    E.nom ,
    E.prenom ,
    M.nom ,
    M.coef
```

Ensuite, on écrit une requête permettant de calculer comme demandé la moyenne générale de chaque élève :

```
SELECT
    nom ,
    prenom ,
    SUM(coef*moyenne)/(SELECT SUM(coef) FROM
    matieres) AS MoyGen
FROM
    mavue
GROUP BY
    nom ,
    prenom
```

8 Un cas pratique : site web marchand

 **Enoncé**
p. 254

- 1 Un fichier SQL permettant de créer les quatre tables est le suivant :

live

```
CREATE TABLE IF NOT EXISTS users (
    idUser BIGINT PRIMARY KEY NOT NULL
    AUTO_INCREMENT ,
    nom VARCHAR (30) ,
    prenom VARCHAR (30) ,
```

(Suite page suivante)

```
email VARCHAR (50) ,  
rue VARCHAR (100) ,  
cp VARCHAR (10) ,  
ville VARCHAR (50) ,  
tel VARCHAR (12) );
```

```
CREATE TABLE IF NOT EXISTS modeles (  
    idModele BIGINT PRIMARY KEY NOT NULL  
    AUTO_INCREMENT ,  
    nomModele VARCHAR (20) );
```

```
CREATE TABLE IF NOT EXISTS articles (  
    idArticle BIGINT PRIMARY KEY NOT NULL  
    AUTO_INCREMENT ,  
    nom VARCHAR (50) ,  
    descriptif VARCHAR (255) ,  
    idModele BIGINT ,  
    prix DOUBLE ,  
    FOREIGN KEY (idModele) REFERENCES modeles(  
        idModele) );
```

```
CREATE TABLE IF NOT EXISTS panier (  
    idPanier BIGINT PRIMARY KEY NOT NULL  
    AUTO_INCREMENT ,  
    idUser BIGINT ,  
    idArticle BIGINT ,  
    idModele BIGINT ,  
    quantite INT ,  
    total DOUBLE ,  
    FOREIGN KEY (idUser) REFERENCES users(idUser) ,  
    FOREIGN KEY (idModele) REFERENCES modeles(idModele)  
    ) ,  
    FOREIGN KEY (idArticle) REFERENCES articles(  
        idArticle) )
```

La taille des colonnes est ici arbitraire : nous les avons choisi en essayant d'être cohérent.

Pour ce qui est de la colonne « tel », nous avons fait le choix de la déclarer comme chaîne de caractères (pour que le format des numéros soient de la forme +336XXXXXXXX ou +337XXXXXXXX).

Pour les clés primaires, nous avons fait le choix du format « BIGINT » car nous sommes optimistes... et souhaitons beaucoup d'entrées pour cette base de données, qui signifierait un grand succès du site.

Concernant le format numérique, nous avons le choix entre :

Type de données	Plage	Stockage
BIGINT	-2^{63} à $2^{63} - 1$	Huit octets
INT	-2^{31} à $2^{31} - 1$	Quatre octets
SMALLINT	-2^{15} à $2^{15} - 1$	Deux octets
TINYINT	0 à 255	Un octet

- 2 Pour cette question, il faut anticiper : il faut avant tout enregistrer les différents modèles cités (Unique, S, M, L, XL et XXL). Ce n'est qu'ensuite que l'on pourra enregistrer les articles. On a alors :

live

INSERT INTO

```
modeles (nomModele)
```

VALUES

```
( 'Unique' ), ( 'S' ),
( 'M' ), ( 'L' ),
( 'XL' ), ( 'XXL' );
```

INSERT INTO

```
articles (nom,descriptif,idModele,prix)
```

VALUES

```
( 'Mug' ,
  'Magnifique mug personnalisable avec votre
photo.' ,
  1 ,
  '7.99' ),
( 'T-Shirt "I Kiffe Beef"' ,
  'T-shirt unique en son genre.' ,
  2 ,
  '24.99' ),
( 'T-Shirt "I Kiffe Beef"' ,
  'T-shirt unique en son genre.' ,
  3 ,
  '24.99' ),
(
  'T-Shirt "I Kiffe Beef"' ,
  'T-shirt unique en son genre.' ,
  4 ,
  '24.99'
),
```

(suite du fichier page suivante)

```
(
    'T-Shirt "I Kiffe Beef"' ,
    'T-shirt unique en son genre.' ,
    '5 ,
    '24.99'
),
(
    'T-Shirt "I Kiffe Beef"' ,
    'T-shirt unique en son genre.' ,
    6 ,
    '24.99'
)
```

live

3

Insertion des utilisateurs

```
INSERT INTO
    users (nom,prenom,email,rue,cp,ville,tel)
VALUES
    (
        'Dupont' ,
        'Jean' ,
        'jdupont@free.fr' ,
        '3 rue des tulipes' ,
        '75000' ,
        'Paris' ,
        '+33601020304'
    ) ,
    (
        'Aimaun' ,
        'Anne' ,
        'amz@free.fr' ,
        '1 rue du glas' ,
        '33000' ,
        'Bordeaux' ,
        '+33799989796'
    )
```

live

4

(a)

Insertion des commandes dans le panier

```
INSERT INTO
    panier
    (idUser,idArticle,idModele,quantite,total)
```

(suite page suivante)

VALUES

```
( '1' , '1' , '2' , '15.98' ) ,  
( '1' , '2' , '1' , '24.99' ) ,  
( '1' , '3' , '1' , '24.99' ) ,  
( '1' , '4' , '1' , '24.99' ) ,  
( '2' , '1' , '1' , '24.99' ) ,  
( '2' , '3' , '1' , '24.99' )
```

- (b) La requête permettant d'afficher le total du panier de Jean Dupont est la suivante :

```
SELECT  
    SUM(total)  
FROM  
    panier  
WHERE  
    idUser = 1
```

- (c) Pour cette question, nous allons avant tout créer une vue, basée notamment sur la table « panier », donnant une table avec deux attributs : le « idUser » d'un enregistrement de la table « panier » ainsi que le total de la commande correspondant à l'enregistrement. Pour cela, nous allons prendre la jointure interne de la table « panier » avec la table « articles » sur l'attribut « idArticle », en regroupant selon l'attribut « idUser ».

live

```
CREATE VIEW V_TOTAL_PANIER AS  
SELECT  
    PAN.idUser ,  
    SUM(PAN.quantite*ART.prix) AS TOTAL  
FROM  
    panier AS PAN  
    INNER JOIN  
        articles AS ART  
    ON ART.idArticle = PAN.idArticle  
GROUP BY  
    PAN.idUser;  
  
SELECT TOTAL  
FROM V_TOTAL_PANIER  
WHERE idUser = 1
```

Une fois la vue créée, il suffit de sélectionner l'attribut « TOTAL » qui correspond à notre client, à savoir celui pour lequel idUser = 1.

Autre solution (sans « GROUP BY ») :

```
CREATE VIEW V AS  
  SELECT * FROM panier WHERE idUsers = 1;  
  
SELECT SUM( quantite * (  
  SELECT prix  
  FROM articles A  
  WHERE A.idArticle = V.idArticle )  
  ) AS TOTAL  
FROM V
```