Projet Foody

Table des matières

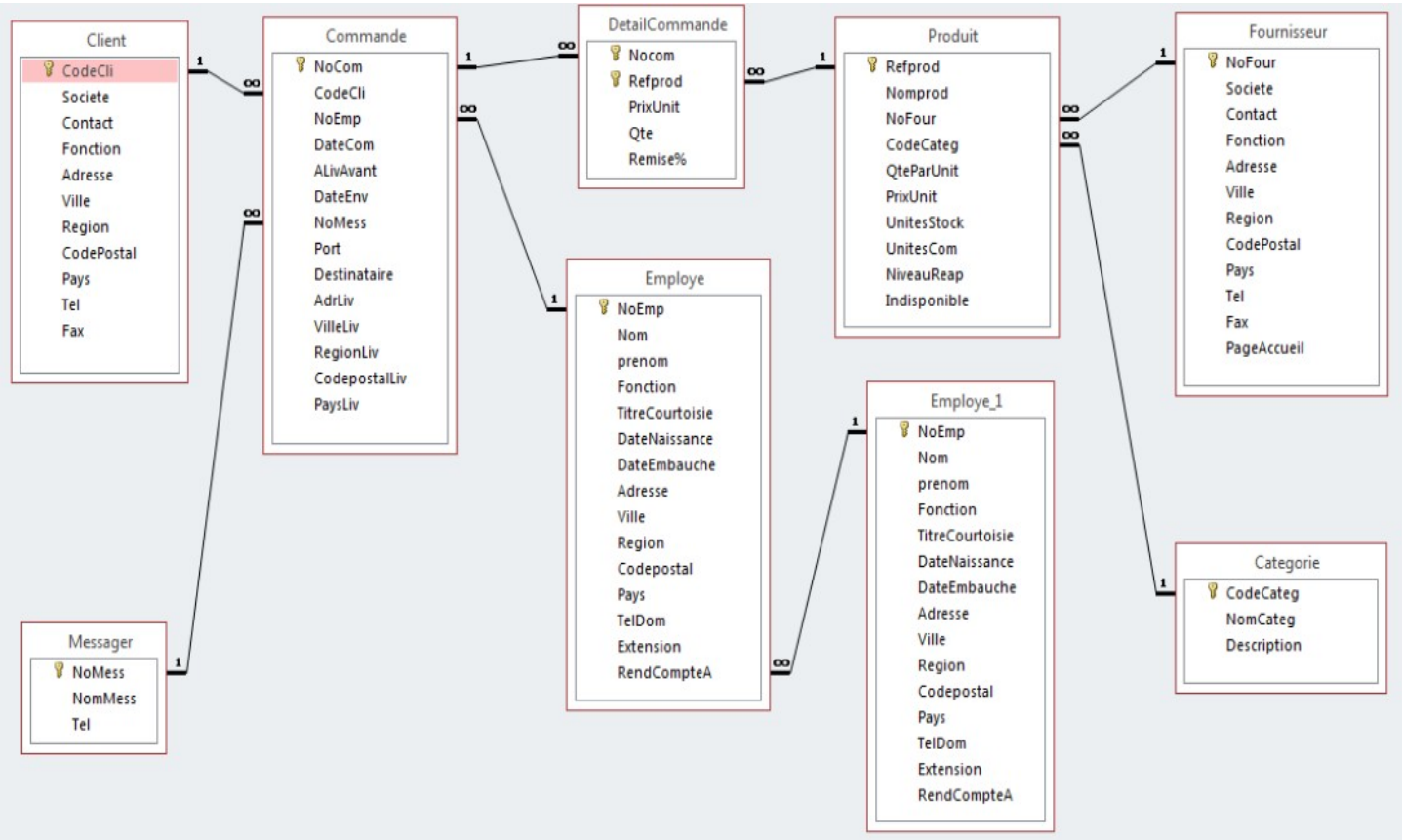
Introduction

Une base de données est un ensemble de tables (ou aussi relations - d'où le nom de base de données **relationnelles**), dont le contenu est la description d'entités (clients, produits, étudiants, matières, ...) ou d'associations (achat d'un produit par un client, note d'un étudiant à une matière, ...).
Elle a pour but de simplifier le fonctionnement d'une organisation ou d'une entreprise via une ou plusieurs applications informatiques.
Nous devons utiliser ce qu'on appelle un Système de Gestion de Bases de Données (ou SGBD) pour gérer ces bases et faire toutes les opérations nécessaires sur celles-ci (création, insertion, modification, interrogation, suppression).

Contexte du Projet :

Soit le schéma de la base de données suivante, qui reprend les caractéristiques de la société Foody spécialisée en import-export d'aliments .

Shéma de la Base de données :



1- Ecrire le Modèle Physique de données correspondant à ce schéma.

Vérifier votre schéma de BDD en y insérant les donnant fournies dans les 8 fichiers csv en Pièces jointes..
(si bloqué je peux aussi vous donner un Foody_data.sql)
(RQ : faites attention aux noms des attributs des fichiers data, ils sont en anglais!!)

2- Exploiter la Base de données :

Répondre aux requêtes du projet en s'appuyant sur le rappel des concepts clés suivant :

*** Modèle relationnel**

Afin de créer des bases de données, le modèle relationnel permet de définir les tables Chaque ligne est appelée un tuple ou enregistrement. Chaque colonne est appelée un attribut ou une variable.
Il permet aussi d'établir des contraintes d'intégrité. Ces contraintes ont pour but de limiter le plus possible les problèmes de cohérence de données à gérer plus tard. On parle de *règles d'intégrité structurelles*.

* Langage SQL

Le langage SQL est un standard pour interroger une base de données relationnelles (comme MySQL,Oracle, SQL Server, SQLite, DB2, ...). L'intérêt de ce langage est qu'il est très puissant et très bien documenté (nombreux livres et sites internet dédiés).
Il permet, via des requêtes, de créer, manipuler, interroger, modifier, supprimer toutes les données présentes.

I- Requêtage simple :

I.1- Requêtage simple

```
SELECT *
    FROM nom_table;
```

I.1.1 Limitation des résultats

```
SELECT *
    FROM nom_table
    LIMIT 3;
```

I.1.2- Ordre des résultats

Il est possible d'indiquer de deux façons le ou les attributs à prendre en compte pour le tri :

- par leur nom
- par leur position dans la table

Les deux requêtes suivantes sont les mêmes et permettent toutes deux d'avoir la liste des employés triés dans l'ordre croissant de leur nom (attribut Nom placé en 2ème position).

```
SELECT *
    FROM Employe
    ORDER BY Nom;
```

```
SELECT *
    FROM Employe
    ORDER BY 2;
```

Dans un souci de clarté, il est tout de même préférable d'utiliser la première option.

Ensuite, pour modifier le type de tri, il est possible d'ajouter le terme DESC pour indiquer qu'on souhaite un tri décroissant. Par défaut, c'est donc un tri croissant qui est fait.

```
SELECT *
    FROM Employe
    ORDER BY Nom DESC;
```

Il est possible d'avoir plusieurs critères de tri. Pour cela, il faut séparer les différents attributs (nom ou position) par une virgule (,). La requête suivante permet donc d'avoir les employés triés d'abord par leur fonction, puis par leur nom.

```
SELECT *
    FROM Employe
    ORDER BY Fonction, Nom;
```

Voici la même requête que précédemment, avec les fonctions triées par ordre alphabétique décroissant.

```
SELECT *
    FROM Employe
    ORDER BY Fonction DESC, Nom;
```

Exercices :

1.Lister le contenu de la table Produit
2.N'afficher que les 10 premiers produits
3.Trier tous les produits par leur prix unitaire
4.Lister les trois produits les plus chers

I.2- Restriction

Une restriction est une sélection de lignes d'une table, sur la base d'une condition à respecter, définie à la suite du terme **WHERE**. Cette condition peut être une combinaison de comparaisons à l'aide de AND, de OR et de NOT (attention donc aux parenthèses dans ce cas).

I.2.1- Opérateurs classiques

Soit les opérateurs classiques de comparaison : =, <>, >, >=, <, <=.
Cette requête permet de lister tous les employés ayant la fonction de représentant.

```
SELECT *
    FROM Employe
    WHERE Fonction = "Représentant(e)";
```

Cette requête permet de lister tous les employés qui ne sont pas représentants, on utilise le symbole <> pour indiquer une non-égalité (ce qui revient à faire NOT(Fonction = "Représentant(e)")).

```
SELECT *
    FROM Employe
```

```
    WHERE Fonction <> "Représentant(e)";
```

Pour les comparaisons de chaînes de caractères, il est important de faire attention à la casse (i.e. minuscule/majuscule). Par définition, un "a" est donc différent d'un "A". Pour remédier à ce problème, il existe les fonction UPPER() et LOWER() pour transformer une chaîne en respectivement majuscule et minuscule.

```
SELECT *
    FROM Employe
    WHERE UPPER(Ville) = "SEATTLE";
```

I.2.2- Données manquantes

Une donnée manquante en SQL est repérée par un NULL. Il y a plusieurs raisons, bonnes ou mauvaises, pour avoir des données manquantes, et il est parfois utile de tester leur présence. Pour cela, nous allons utiliser le terme IS NULL comme condition.
Par exemple, pour lister les employés dont la région n'est pas renseignée, nous devons exécuter la requête suivante.

```
SELECT *
    FROM Employe
    WHERE Region IS NULL;
```

Au contraire, si l'on veut uniquement les employés pour lesquels l'information est présente, nous devrons utiliser la négation avec IS NOT NULL.

```
SELECT *
    FROM Employe
    WHERE Region IS NOT NULL;
```

I.2.3- Opérateurs spécifiques

Les deux premiers opérateurs définis ci-après sont particulièrement utiles pour limiter la taille de la requête. Le dernier est lui utile pour comparer une chaîne de caractères à une pseudo-chaîne.

* BETWEEN

Cet opérateur permet de définir un intervalle fermé dans lequel l'attribut doit avoir sa valeur. La condition suivante est équivalente à NoEmp >= 3 AND NoEmp <= 8.

```
SELECT *
    FROM Employe
    WHERE NoEmp BETWEEN 3 AND 8;
```

* IN

Cet autre opérateur permet de définir une liste de valeurs entre parenthèses et séparées par des virgules. La condition suivante est équivalente à TitreCourtoisie = 'Mlle' OR TitreCourtoisie = 'Mme'.

```
SELECT *
    FROM Employe
     WHERE TitreCourtoisie IN ('Mlle', 'Mme');
```

* LIKE

Comme précisé avant, l'opérateur LIKE permet de comparer une chaîne de caractère à une pseudo-chaîne, dans laquelle nous pouvons ajouter deux caractères spécifiques :

- % : une suite de caractères, éventuellement nulle
- _ : un et un seul caractère

Par exemple, la requête suivante permet de récupérer les employés dont le nom commence par un "D".

```
SELECT *
    FROM Employe
    WHERE Nom LIKE 'D%';
```

La requête suivante permet elle d'avoir tous les employés qui ont un prénom de 5 lettres.

```
SELECT *
    FROM Employe
    WHERE Prenom LIKE '_____';
```

Il faut noter que l'opérateur LIKE est insensible à la casse, i.e. il ne tient pas compte des minuscules/majuscules.

Exercices :

1.Lister les clients français installés à Paris
2.Lister les clients français, allemands et canadiens
3.Lister les clients dont le numéro de fax n'est pas renseigné
4.Lister les clients dont le nom contient "restaurant" (nom présent dans la colonne Societe/CompanyName)

I.3- Projection

Une projection est une sélection de colonnes d'une table, sur la base d'une liste d'attributs placés après le SELECT et séparés par une virgule.
La requête suivante permet d'avoir uniquement les noms et les prénoms des employés.

```
SELECT Nom, Prenom
    FROM Employe;
```

I.3.1- Doublons

Lors d'une projection, on est souvent en présence de doublons dans les résultats, i.e. deux lignes ou plus identiques.
Par exemple, lorsqu'on liste les fonctions des employés, on a plusieurs fois chaque fonction existante.

```
SELECT Fonction
    FROM Employe;
```

Or, dans ce cas, on est souvent intéressé par la liste des valeurs uniques. Pour l'obtenir, il est possible d'ajouter le terme DISTINCT juste après le SELECT, pour supprimer ces doublons.

```
SELECT DISTINCT Fonction
```

```
    FROM Employe;
```

Ceci fonctionne aussi lorsqu'on a indiqué plusieurs attributs dans le SELECT.

I.3.2- Renommage

Pour améliorer la présentation, il est possible de renommer un attribut (et on le verra plus tard le résultat de calcul), avec le terme AS placé après l'attribut à renommer et suivi du nouveau nom.

```
SELECT DISTINCT Fonction AS "Fonctions existantes"
    FROM Employe;
```

Exercices :

1.Lister uniquement la description des catégories de produits (table Categorie)
2.Lister les différents pays des clients
3.Idem en ajoutant les villes, le tout trié par ordre alphabétique du pays et de la ville

4.Lister tous les produits vendus en bouteilles (bottle) ou en canettes(can)

5.Lister les fournisseurs français, en affichant uniquement le nom, le contact et la ville, triés par ville

6.Lister les produits (nom en majuscule et référence) du fournisseur n° 8 dont le prix unitaire est entre 10 et 100 euros, en renommant les attributs pour que ça soit explicite

7.Lister les numéros d'employés ayant réalisé une commande (cf table Commande) à livrer en France, à Lille, Lyon ou Nantes

8.Lister les produits dont le nom contient le terme "tofu" ou le terme "choco", dont le prix est inférieur à 100 euros (attention à la condition à écrire)

II- Calculs et Fonctions :

II.1- Calculs arithmétiques

II.1.1- Calcul simple

Il est possible d'effectuer des calculs arithmétiques dans un SELECT, à l'aide des opérateurs classiques : +, -, *, /, ().
Voici un premier exemple de calcul dans une requête. On additionne les unités en stock avec les unités commandées, pour chaque produit.

```
SELECT *, UnitesStock + UnitesCom
    FROM Produit;
```

Bien évidemment, on peut vouloir faire aussi un projection en même temps, en n'affichant que la référence du produit.

```
SELECT RefProd, UnitesStock + UnitesCom
    FROM Produit;
```

* Renommage → Alias

Pour plus de lisibilité, il est d'usage de renommer un calcul, grâce à AS. Si l'on désire mettre des accents (en français) et/ou des espaces, il est nécessaire d'ajouter les "..." (ou '...').

```
SELECT RefProd AS Reference,
       UnitesStock + UnitesCom AS "Unités disponibles"
   FROM Produit;
```

Vous remarquerez qu'il est possible de passer à la ligne suivante dans un SELECT pour rendre le code plus simple à lire. En effet, en SQL, on peut écrire tout sur une même ligne, jusqu'à un mot par ligne. Le moteur du SGBD supprime les espaces inutiles et les sauts de lignes avant l'exécution de la requête. Mais il est important d'avoir un code lisible (débugage plus simple, meilleure compréhension , réutilisation facile, ...).

Dans cet exemple, nous utilisons la multiplication * pour calculer le montant en stock pour chaque produit, égal au prix unitaire multiplié par la quantité de produits en stock.

```
SELECT RefProd,
       PrixUnit * UnitesStock AS "Montant en stock"
   FROM Produit;
```

* Combinaison de clauses

Puisque nous sommes dans une requête SELECT, nous pouvons bien évidemment utiliser toutes les restrictions que l'on désire, dans le WHERE.

```
SELECT RefProd,
       PrixUnit * UnitesStock AS "Montant en stock indisponible"
   FROM Produit
   WHERE Indisponible = 1;
```

On peut aussi trier le résultat, à l'aide de ORDER BY, et se limiter à n lignes, à l'aide de LIMIT. Nous avons donc ici les trois produits indisponibles avec le plus haut montant en stock.

```
SELECT RefProd,
       PrixUnit * UnitesStock AS "Montant en stock"
   FROM Produit
   WHERE Indisponible = 1
   ORDER BY 2 DESC
   LIMIT 3;
```

II.1.2- Calcul complexe

Les calculs peuvent être un peu plus complexes, grâce à l'utilisation des parenthèses. Par exemple, considérons que nous voulons garder au moins 10 unités de chaque produit. Nous calculons dans la requête suivante le montant en stock directement disponible, en tenant compte de la contrainte précédente.

```
SELECT RefProd,
       PrixUnit * (UnitesStock - 10)
   FROM Produit
   WHERE UnitesStock >= 10;
```

Toute expression mathématique combinant les opérateurs classiques est donc acceptable à ce niveau.

* Arrondi

Il est possible d'obtenir l'arrondi d'un réel grâce à la fonction ROUND(). Voici un exmple de calcul d'une augmentation de 5% des prix des produits.

```
SELECT RefProd,
       ROUND(PrixUnit * 1.05) AS "Nouveau Prix"
   FROM Produit;
```

L'arrondi ci-dessus est à l'entier. Si l'on désire un arrondi à 2 décimales (et donc les centimes dans notre cas), il faut ajouter un 2 comme second paramètre de la fonction ROUND().

```
SELECT RefProd,
       ROUND(PrixUnit * 1.05, 2) AS "Nouveau Prix"
   FROM Produit;
```

Exercices :

La table DetailsCommande contient l'ensemble des lignes d'achat de chaque commande. Calculer, pour la commande numéro 10251, pour chaque produit acheté dans celle-ci, le montant de la ligne d'achat en incluant la remise (stockée en proportion dans la table). Afficher donc (dans une même requête) :

```
- le prix unitaire,
- la remise,
- la quantité,
- le montant de la remise,
- le montant à payer pour ce produit
```

II.2- Traitement conditionnel

Nous avons vu comment se restreindre à un sous-ensemble d'une table via les restrictions dans la partie WHERE d'une requête. Il existe aussi une possibilité de faire un traitement conditionnel avec le terme CASE. Dans un SELECT, celui-ci va nous permettre de conditionner la valeur d'une colonne par les valeurs d'autres colonnes.

II.2.1- Comparaison à des valeurs (égalité)

La première utilisation de cette commande est de comparer un attribut à un ensemble de valeurs. Puisque la comparaison est l'égalité, ceci concerne principalement des attributs de type texte ou avec un nombre de valeurs restreint. Dans ce cas, l'ordre des comparaisons à l'aide de WHEN n'a aucune importance.

Dans l'exemple ci-dessous, nous voulons afficher les titres de courtoisie au complet pour "Mlle", "Mme" et "M.". Pour "Dr.", nous voulons le garder par contre. Vous pouvez donc voir la syntaxe de cette commande.

```
SELECT NoEmp, Nom, Prenom, TitreCourtoisie,
       CASE TitreCourtoisie
          WHEN "Mlle" THEN "Mademoiselle"
```

```
            WHEN "Mme"  THEN "Madame"
            WHEN "M."   THEN "Monsieur"
            ELSE TitreCourtoisie
        END AS Titre
    FROM Employe;
```

On peut aussi l'utiliser pour mettre un message en fonction de la valeur d'un attribut. Ci-dessous, nous affichons à partir de quel niveau de stock le produit doit être commandé pour réapprovisionnement. Si c'est à 0, on indique qu'il n'y a pas de niveau minimum. Sinon, on utilise la valeur stockée dans NiveauReap pour créer le message.

```
SELECT Refprod, Nomprod, NiveauReap,
       CASE NiveauReap
           WHEN 0 THEN "Pas de niveau minimum"
           ELSE "Réapprovisionnement à partir de " || NiveauReap ||
" unités restantes"
       END AS Reapprovisionnement
    FROM Produit;
```

II.2.2- Comparaison à des valeurs

Pour pouvoir comparer un attribut avec des valeurs mais autrement que via l'égalité, il est aussi possible d'utiliser un CASE.

Dans l'exemple ci-dessous, nous comparons le prix unitaire des produits à deux valeurs seuils. Si le prix est inférieur ou égal à 50, alors le produit est considéré dans la gamme des petits prix. Ensuite, on teste pour savoir s'il est inférieur ou égal à 500 et si oui, le produit sera dans la gamme moyenne. Enfin, par défaut, le produit sera dans la gamme de luxe.

```
SELECT Refprod, Nomprod, PrixUnit,
       CASE
           WHEN PrixUnit <= 50 THEN "Petits prix"
           WHEN PrixUnit <= 500 THEN "Gamme moyenne"
           ELSE "Produits de luxe"
       END AS Gamme
    FROM Produit;
```

Ceci revient à écrire en algo ce qui suit :

```
SI (PrixUnit <= 50) ALORS
    gamme = "Petits prix"
SINON
    SI (PrixUnit <= 500) ALORS
        gamme = "Gamme moyenne"
    SINON
        gamme = "Produits de luxe"
    FIN SI
FIN SI
```

Dans le deuxième test, il n'est pas nécessaire de tester si le prix est supérieur strictement à 50, car on est dans la partie SINON du premier test.

II.2.3- Comparaison entre attributs

Enfin, il est aussi possible d'utiliser ce test CASE pour comparer plusieurs attributs entre eux.
Dans l'exemple ci-dessous, nous voulons afficher un message en fonction de l'action à réaliser ou non pour le réapprovisionnement. Si le produit a déjà été comamndé (i.e. UnitesCom > 0), alors on l'indique. Par contre, s'il ne l'est pas mais que le stock est inférieur au niveau de réapprovisionnement, alors on indique qu'il faut commander le produit. Pour les produits qui ne sont plus en stock (et dont le niveau de réapprovisionnement est égal à 0), on indique juste qu'il n'y a plus de produits à disposition. Pour les autres, il n'y a, a priori, rien à faire.

```
SELECT Refprod, UnitesStock, UnitesCom, NiveauReap,
       CASE
          WHEN (UnitesCom > 0) THEN "Déjà commandé"
          WHEN (UnitesStock < NiveauReap) THEN "A commander"
          WHEN (UnitesStock = 0) THEN "Plus en stock"
          ELSE "rien à faire"
       END AS Informations
   FROM Produit;
```

Exercices :

1.A partir de la table Produit, afficher "Produit non disponible" lorsque l'attribut Indisponible vaut 1, et "Produit disponible" sinon.
2.Dans la table DetailsCommande, indiquer les infos suivantes en fonction de la remise

- si elle vaut 0 : "aucune remise"
- si elle vaut entre 1 et 5% (inclus) : "petite remise"
- si elle vaut entre 6 et 15% (inclus) : "remise modérée"
- sinon :"remise importante"

3.Indiquer pour les commandes envoyées si elles ont été envoyées en retard (date d'envoi DateEnv supérieure (ou égale) à la date butoir ALivAvant) ou à temps

II.3- Fonctions sur chaînes de caractères

II.3.1- Concaténation

La première opération que l'on souhaite faire avec des chaînes de caractères est la concaténation : le regroupement des deux chaînes en une seule. Par exemple, la concaténation de "bon" et de "jour" donne la chaîne "bonjour". L'opérateur dédié en SQL est ||. L'exemple ci-dessous nous permet d'avoir le nom et le prénom dans une seule chaîne.

```
SELECT NoEmp, Nom || Prenom
   FROM Employe;
```

Avec la requête ci-dessus, les deux chaînes sont collées, i.e. il n'y a pas d'espace entre les deux. Pour cela, il est tout à fait possible de concaténer en une expression plusieurs chaînes pour introduire un espace, comme ci-après.

```
SELECT NoEmp, Nom || " " || Prenom
    FROM Employe;
```

II.3.2- Extraction d'une sous-chaîne

Une commande intéressante sur les chaînes est la commande SUBSTR(chaine, debut, longueur) qui permet d'extraire une sous-chaîne d'une chaîne, en partant du caractère précisé dans debut et sur une longueur précisé par longueur. Dans l'exemple ci-dessous, nous extrayons l'initiale du prénom.

```
SELECT NoEmp, Nom || " " || SUBSTR(Prenom, 1, 1)
    FROM Employe;
```

Et on ajoute un "." pour indiquer que c'est une initiale. Il n'y a pas de limite sur le nombre de chaînes que l'on peut concaténer en une seule expression.

```
SELECT NoEmp, Nom || " " || SUBSTR(Prenom, 1, 1) || "."
    FROM Employe;
```

II.3.3- Majuscule/Minuscule

Pour pouvoir transformer une chaîne en majuscule (et respectivement en minuscule), nous avons à disposition la commande UPPER(chaine) (et resp. LOWER(chaine)). Cette commande, comme toutes les autres, peut aussi être utilisé dans un WHERE.

```
SELECT NoEmp, UPPER(Nom) || " " || SUBSTR(Prenom, 1, 1) || "."
    FROM Employe;
```

La commande LENGTH(chaine) permet de renvoyer la longueur de la chaîne (i.e. le nombre de caractères, y compris les espaces).

```
SELECT NoEmp, Nom, LENGTH(Nom)
    FROM Employe;
```

II.3.4- Modification d'une sous-chaîne

La commande REPLACE(chaîne, sc1, sc2) permet de remplacer la sous-chaîne sc1 par la sous-chaîne sc2 dans la chaîne de caractères passée en premier paramètre. Ci-dessous, nous remplaçons donc le terme "Chef" par le terme "Responsable" dans les titres de fonction des employés.

```
SELECT Nom, Prenom, Fonction,
       REPLACE(Fonction, "Chef", "Responsable")
    FROM Employe;
```

II.3.5- Recherche d'une sous-chaîne

Pour rechercher la première apparition d'une sous-chaîne dans une chaîne, nous disposons de la commande INSTR(chaîne, souschaine). Celle-ci recherche donc à quelle position dans la chaîne se trouve la première occurence de la sous-chaîne. Si la sous-chaîne n'est pas présente, la fonction renvoie 0.

Ci-dessous, nous cherchons la présence du terme "Ave." dans l'adresse des employés.

```
SELECT Nom, Adresse,
       INSTR(Adresse, "Ave.")
   FROM Employe;
```

Exercices :

Dans une même requête, sur la table Client :

1. Concaténer les champs Adresse, Ville, CodePostal et Pays dans un nouveau champ nommé Adresse complète, pour avoir :

   Adresse, CodePostal Ville, Pays

2. Extraire les deux derniers caractères des codes clients
3. Mettre en minuscule le nom des sociétés
4. Remplacer le terme "Owner" par "Freelance" dans ContactTitle
5. Indiquer la présence du terme "Manager" dans ContactTitle

II.4- Fonctions sur les dates

Nous disposons en SQL (comme dans d'autres langages) de plusieurs formats pour les dates. Soit nous avons uniquement la date (jour, mois et année - stockée sous la forme d'un entier représentant le nombre de jours depuis une date de référence, souvent le 1/1/1970). Il existe aussi un format où la date, l'heure et le fuseau horaire sont stockées (précisemment, le nombre de secondes depuis la même date de référence et le fuseau horaire).
Vous trouverez sur cette page plus d'informations sur les fonctions disponibles.

II.4.1- Génération de dates

En premier lieu, si nous désirons avoir la date du jour (de l'exécution de la requête bien sûr), nous pouvons exécuter cette requête. La date est affichée au format "YYYY-MM-DD".

```
SELECT DATE("now");
```

La commande DATE() peut prendre d'autres paramètres après le premier contenant la date (ou "now"), permettant de modifier cette date. La requête suivante permet d'avoir la date de la veille.

```
SELECT DATE("now", "-1 day");
```

Il est possible de cumuler plusieurs modificateurs pour, par exemple, obtenir le dernier jour du mois dernier.

```
SELECT DATE("now", "start of month", "-1 day");
```

La commande DATE() accepte aussi en premier paramètre une date au bon format, pour par exemple lui appliquer une modification par la suite. Nous avons ici la date du lendemain de la commande.

```
SELECT DATE(DateCom, "+1 day")
   FROM Commande;
```

Exercices :

1.Afficher le jour de la semaine en lettre pour toutes les dates de commande
2.Compléter en affichant "week-end" pour les samedi et dimanche,
et "semaine" pour les autres jour
3.Calculer le nombre de jours entre la date de la commande (DateCom) et la date butoir de livraison (ALivAvant), pour chaque commande
4.On souhaite aussi contacter les clients 1 an, 1 mois et 1 semaine après leur commande. Calculer les dates correspondantes pour chaque commande

Bonus :

1.Récupérer l'année de naissance et l'année d'embauche des employés
2.Calculer à l'aide de la requête précédente l'âge d'embauche et le nombre d'années dans l'entreprise
3.Afficher le prix unitaire original, la remise en pourcentage, le montant de la remise et le prix unitaire avec remise (tous deux arrondis aux centimes), pour les lignes de commande dont la remise est strictement supérieure à 10%
4.Calculer le délai d'envoi (en jours) pour les commandes dont l'envoi est après la date butoir, ainsi que le nombre de jours de retard
5.Rechercher les sociétés clientes, dont le nom de la société contient le nom du contact de celle-ci

III- Aggrégats

III.1- Dénombrements

Les dénombrements sont très utilisés en SQL. D'une part, car il est important de savoir combien il y a de clients, commandes, ... D'autre part, c'est un moyen intéressant de contrôler son travail lors de l'écriture d'un programme complexe, demandant plusieurs requêtes.
C'est le premier agrégat (ou calcul d'agrégat) à voir. Nous utilisons pour cela la commande COUNT().

III.1.1- Nombre de lignes d'une table

Pour calculer le nombre de lignes d'une table, quelque soit les valeurs dans celle-ci, le plus correct est d'utiliser COUNT(*). L'étoile ici indique que l'on prend toute la table, comme dans une requête SELECT * FROM table.
Nous avons donc ici le nombre de clients contenus dans la base de données.

```
SELECT COUNT(*)
    FROM Client;
```

III.1.2- Nombre de valeurs d'un attribut

Il est possible de spécifier un attribut dans le COUNT(). Ceci permettra de compte combien il y a de lignes avec une valeur dans cet attribut. Si on l'utilise sur une clé primaire, nous devrions obtenir le même résultat que précédemment.

```
SELECT COUNT(CodeCli)
    FROM Client;
```

Par contre, si on indique un attribut dans lequel il y a des valeurs manquantes (i.e. NULL en SQL), nous n'aurons pas le même résultat. Nous aurons donc le nombre de valeurs non nulles de cet attribut. Ici, l'attribut est le numéro de fax du client. Comme il y a quelques clients sans numéro de fax, nous n'obtenons pas le même résultat.

```
SELECT COUNT(Fax)
    FROM Client;
```

III.1.3- Nombre de valeurs distinctes d'un attribut

Pour aller plus loin, il est aussi possible d'ajouter la clause DISTINCT avant l'attribut, pour obtenir le nombre de valeurs distincts de cet attribut. Ci-dessous, nous avons donc le nombre de valeurs distinctes de l'attribut Pays. Ce qui nous donne le nombre de pays de la clientèle.

```
SELECT COUNT(DISTINCT Pays)
    FROM Client;
```

Ce nombre correspond au nombre de lignes de la table résultante de la requête suivante.

```
SELECT DISTINCT Pays
    FROM Client;
```

III.1.4- Restriction dans le dénombrement

Pour dénombrer des sous-ensembles d'une table, il est bien évidemment possible d'ajouter des restrictions à la requête. Par exemple, ci-dessous, nous calculons le nombre de clients français présents dans la base.

```
SELECT COUNT(*)
    FROM Client
    WHERE Pays = "France";
```

Exercices :

1.Calculer le nombre d'employés qui sont "Sales Manager"
2.Calculer le nombre de produits de moins de 50 euros
3.Calculer le nombre de produits de catégorie 2 et avec plus de 10 unités en stocks
4.Calculer le nombre de produits de catégorie 1, des fournisseurs 1 et 18
5.Calculer le nombre de pays différents de livraison
6.Calculer le nombre de commandes réalisées le 28/03/2016.

III.2- Calculs statistiques simples

Outre les dénombrements, il est possible de faire quelques calculs statistiques de base, tels que somme, moyenne, minimum et maximum.

III.2.1- Somme

La fonction SUM(attribut) permet donc de faire la somme des valeurs non nulles de l'attribut passé en paramètre. La requête suivante nous permet d'avoir le nombre total d'unités de produits en stock.

```
SELECT SUM(UnitesStock)
    FROM Produit;
```

Bien évidemment, ce calcul peut se faire suite à une restriction, c'est-à-dire sur un sous-ensemble de la table. Ici, nous calculons le nombre d'unités en stock pour tous les produits de la catégorie 1.

```
SELECT SUM(UnitesStock)
    FROM Produit
    WHERE CodeCateg = 1;
```

III.2.2- Moyenne et médiane

Bien qu'avec un SUM() et un COUNT(), on puisse obtenir la moyenne, il existe la fonction AVG(attribut) (pour average) permettant de la calculer directement. La requête ci-dessous permet de calculer le prix unitaire moyen des produits.

```
SELECT AVG(PrixUnit)
    FROM Produit;
```

Dans la plupart des cas, il sera nécessaire d'améliorer la lisibilité du résultat en arrondissant les valeurs, très souvent à 2 décimales, comme ci-après.

```
SELECT ROUND(AVG(PrixUnit), 2)
    FROM Produit;
```

En statistique, il est souvent préférable de calculer la médiane plutôt que la moyenne, ce qu'on peut faire avec la fonction MEDIAN(attribut), tel que l'on peut voir dans la requête suivante.

```
SELECT MEDIAN(PrixUnit)
    FROM Produit;
```

III.2.3- Minimum et maximum

Enfin, deux autres fonctions utiles sont disponibles : MIN(attribut) et MAX(attribut), permettant d'obtenir respectivement le minimum et le maximum d'un attribut, sans tenir compte des valeurs nulles. Nous obtenons donc avec cette requête le prix minimum et le prix maximum.

```
SELECT MIN(PrixUnit), MAX(PrixUnit)
    FROM Produit
```

Exercices

1.Calculer le coût moyen du port pour les commandes du client dont le code est "QUICK" (attribut CodeCli)
2.Calculer le coût du port minimum et maximum des commandes
3.Pour chaque messager (par leur numéro : 1, 2 et 3), donner le montant total des frais de port leur correspondant

III.3- Agrégats selon attribut(s)

Si l'on désire avec un calcul statistique selon un critère (que l'on appellera critère d'agrégation), il n'est pas nécessaire de répéter la requête autant de fois qu'on a de valeurs pour le critère. Il existe pour cela la commande GROUP BY, qui permet de faire un calcul d'agrégat (COUNT(), SUM(), AVG(), ...) pour chaque valeur du critère d'agrégation.
Le GROUP BY doit être obligatoirement placé après le WHERE dans la requête.

III.3.1- Agrégat simple

Le critère d'agrégation est souvent un seul attribut (par exemple, on souhaite le nombre d'étudiants de chaque sexe).

* Utilisation classique

Le premier exemple que nous allons voir est le dénombrement. Nous désirons le nombre de clients de la société, pour chaque pays d'origine. Donc, nous voudrions voir afficher l'attribut Pays en plus du compte. La requête suivante, erronée, est celle qu'on pourrait écrire en premier.

```
SELECT Pays, COUNT(*)
    FROM Client;
```

Une fois exécutée, on se rend compte qu'elle ne renvoie qu'une seule ligne, avec un seul pays (celui en dernier dans la table) et le nombre total de clients. Pour être correct, il faut spécifier le critère d'agrégation (ici le pays) dans la clause GROUP BY, comme ci-dessous.

```
SELECT Pays, COUNT(*)
    FROM Client
    GROUP BY Pays;
```

Ici, le résultat est ordonné par pays. On peut améliorer la lisibilité du résultat en renommant le dénombrement et en ordonnant de manière décroissante par celui-ci.

```
SELECT Pays, COUNT(*) AS "Nb clients"
    FROM Client
    GROUP BY Pays
    ORDER BY 2 DESC;
```

Ce mécanisme fonctionne bien évidemment avec tous les autres calculs d'agrégats que nous avons vu précédemment (SUM(), AVG(), ...).

* Combinaison d'agrégats

Il est aussi possible de calculer directement plusieurs agrégats en une seule requête, comme ci-dessous. Nous cherchons donc à avoir, pour chaque fournisseur :

- le nombre de produits
- le prix moyen (arrondi à l'entier)
- le prix minimum
- le prix maximum

```
SELECT NoFour,
       COUNT(*) AS "Nb produits",
       ROUND(AVG(PrixUnit)) AS "Prix moyen",
       MIN(PrixUnit) as "Prix minimum",
       MAX(PrixUnit) as "Prix maximum"
   FROM Produit
   GROUP BY NoFour;
```

III.3.2- Agrégat complexe

Par contre, il arrive que nous souhaitions avoir un critère d'agrégation prenant en compte plusieurs attributs. Par exemple, on peut vouloir connaître le nombre d'étudiants en DUT STID par sexe et par année (1ère ou 2ème). Dans ce cas, nous aurons à calculer quatre valeurs :

- le nombre d'hommes en 1ère année
- le nombre d'hommes en 2ème année
- le nombre de femmes en 1ère année
- le nombre de femmes en 2ème année

Dans ce cas, il faut spécifier les attributs à la fois dans le SELECT et dans le GROUP BY. Ci-dessous, nous cherchons donc à connaître le nombre de produits pour chaque fournisseur et chaque catégorie. La première ligne nous indique qu'on a 2 produits en stock du fournisseur 1 et de la catégorie 1.

```
SELECT NoFour, CodeCateg, COUNT(*)
   FROM Produit
   GROUP By NoFour, CodeCateg;
```

Plus généralement, il est obligatoire que les attributs présents dans le SELECT soient aussi présents dans le GROUP BY. Dans le cas contraire, le résultat ne correspondra pas à ce qu'on cherche à obtenir et ce n'est pas toujours facile à repérer.
Par exemple ici, en ne mettant pas CodeCateg dans le GROUP BY, on a bien un résultat mais seul un numéro de catégorie est retenu (au hasard) pour chaque fournisseur. Le fournisseur 4, qui a trois catégories (6, 7 et 8) avec chacune 1 seul produit, n'est plus que sur une seule ligne (avec la catégorie 7 affichée, mais bien 3 produits en tout).

```
SELECT NoFour, CodeCateg, COUNT(*)
   FROM Produit
   GROUP By NoFour;
```

Exercices

1.Donner le nombre d'employés par fonction
2.Donner le montant moyen du port par messager(shipper)
3.Donner le nombre de catégories de produits fournis par chaque fournisseur
4.Donner le prix moyen des produits pour chaque fournisseur et chaque catégorie de produits fournis par celui-ci

III.4- Restriction sur agrégats

Il arrive que nous souhaitions restreindre les résultats avec une condition sur un calcul d'agrégat. Par exemple, on peut vouloir chercher le nombre de clients par pays, uniquement pour les pays avec plus de 10 clients. Dans la requête suivante, il faudrait donc chercher à la main les pays avec plus de 10 clients, ce qui est à proscrire car non-automatique.

```
SELECT Pays, COUNT(*)
    FROM Client
    GROUP BY Pays;
```

La première idée serait de faire une restriction sur le COUNT(*) dans la clause WHERE, comme ci-dessous. Comme vous pourrez le voir, cette requête ne fonctionne pas, car le COUNT(*) est mal placé.

```
SELECT Pays, COUNT(*)
    FROM Client
    WHERE COUNT(*) >= 10
    GROUP BY Pays;
```

Pour effectuer ces restrictions, il est nécessaire d'utiliser la clause HAVING, situé obligatoirement après le GROUP BY. Dans notre exemple, nous devons donc écrire la requête suivante.

```
SELECT Pays, COUNT(*)
    FROM Client
    GROUP BY Pays
    HAVING COUNT(*) >= 10;
```

Pour améliorer la lisibilité, il est aussi possible de renommer le résultat de l'agrégat, et d'utiliser ce nouveau nom dans la condition du HAVING.

```
SELECT Pays, COUNT(*) AS Nb
    FROM Client
    GROUP BY Pays
    HAVING Nb >= 10;
```

III.4.1- Placement des différentes clauses

Nous avons maintenant vu tous les clauses existantes dans une requête SQL de type SELECT. Il est important de se souvenir de l'ordre d'apparition de celles-ci, tel que présenté ci-dessous. Si cet ordre n'est pas respecté, le moteur SGBD ne pourra pas traiter la requête et renverra une erreur.

```
SELECT attributs, calculs, agrégats
```

```
FROM tables
WHERE conditions
GROUP BY attributs
HAVING conditions
ORDER BY attributs
LIMIT nombre;
```

Exercices

1.Lister les fournisseurs ne fournissant qu'un seul produit
2.Lister les catégories dont les prix sont en moyenne supérieurs strictement à 50 euros
3.Lister les fournisseurs ne fournissant qu'une seule catégorie de produits
4.Lister le Products le plus cher pour chaque fournisseur, pour les Products de plus de 50 euro

Bonus :

1.Donner la quantité totale commandée par les clients, pour chaque produit
2.Donner les cinq clients avec le plus de commandes, triés par ordre décroissant
3.Calculer le montant total des lignes d'achats de chaque commande, sans et avec remise sur les produits
4.Pour chaque catégorie avec au moins 10 produits, calculer le montant moyen des prix
5.Donner le numéro de l'employé ayant fait le moins de commandes

IV- Jointures :

IV.1- Jointures naturelles

Une jointure entre deux tables permet de combiner l'information contenue entre les deux tables. Comme vous avez pu le remarquer, les données sont découpées de façon très détaillées dans différentes tables, ceci pour différentes raisons. Pour recouper ces données, il est nécessaire de les combiner et donc de faire des jointures.

IV.1.1- Principe

La jointure naturelle (NATURAL JOIN) permet de recouper les lignes de chaque table ayant les mêmes valeurs pour les attributs ayant le même nom entre les deux tables.
Par exemple, nous souhaitons connaître le nom de la catégorie de chaque produit. Pour cela, nous devons joindre les deux tables Produit et Categorie. Dans les deux, il existe l'attribut CodeCateg. La jointure va donc permettre d'ajouter, pour chaque produit, le nom de la catégorie (NomCateg) et la descrition (Description).

```
SELECT *
    FROM Produit NATURAL JOIN Categorie;
```

Bien évidemment, il est possible de réaliser toute autre opération vu précédemment, dont les projections. Ici, nous nous restreignons à la référence du produit, son nom et la catégorie de celui-ci.

```
SELECT RefProd, NomProd, NomCateg
    FROM Produit NATURAL JOIN Categorie;
```

Si on souhaite avoir l'ensemble des colonnes d'une des tables, il est aussi possible de l'indiquer dans le SELECT avec le formalisme table.*:

```
SELECT RefProd, NomProd, Categorie.*
    FROM Produit NATURAL JOIN Categorie;
```

Ce processus nous permet aussi de se faire des restrictions sur un attribut d'une table pour qu'elles soient répercuter sur l'autre. Par exemple, ici, on ne retient que les produits fournis par des entreprises françaises.

```
SELECT NomProd, Societe
    FROM Produit NATURAL JOIN Fournisseur
    WHERE Pays = "France";
```

Mais il est d'autant plus intéressant quand on veut faire des agrégats. Par exemple, si nous souhaitons connaître le nombre de produits par catégorie, plutôt que de présenter les codes de catégorie, nous allons chercher à présenter le nom des catégories. Le résultat sera ainsi plus parlant.

```
SELECT NomCateg, COUNT(*) AS "Nb Produits"
    FROM Produit NATURAL JOIN Categorie
    GROUP BY NomCateg
    ORDER BY 2 DESC;
```

IV.1.2- Multiples jointures

Il est bien sûr possible de faire plusieurs jointures naturelles dans une même requête. Pour cela, il faut ajouter des parenthèses chaque NATURAL JOIN (et les tables concernées).
Ici, nous ajoutons à la fois les informations de Categorie, mais aussi de Fournisseur, à la table Produit.

```
SELECT *
    FROM (Produit NATURAL JOIN Categorie)
       NATURAL JOIN Fournisseur;
```

IV.1.3- Problème possible

Des problèmes peuvent survenir quand les tables ont des noms de variables identiques, mais qu'on ne souhaite pas à ce qu'elles servent pour la jointure. Dans les tables Client et Employe, il existe les attributs Ville, Pays, ... présents dans les deux tables. Ceci va perturber la jointure, puisqu'il va chercher à faire correspondre (i.e. chercher l'égalité) tous les attributs ayant le même nom.

```
SELECT *
    FROM (Commande NATURAL JOIN Client)
       NATURAL JOIN Employe;
```

Un jointure naturelle n'est donc pas réalisable lorsque :

- les tables ont des attributs ayant le même nom qu'on ne cherche pas à mettre en relation
- les tables n'ont pas d'attributs avec le même nom

Exercices

1. Récupérer les informations des fournisseurs pour chaque produit
2. Afficher les informations des commandes du client "Lazy K Kountry Store"
3. Afficher le nombre de commande pour chaque messager (en indiquant son nom)

IV.2- Jointures internes

IV.2.1- Principe

Une jointure interne (INNER JOIN) est faite pour pallier aux problèmes de la jointure naturelle. Ici, nous allons préciser sur quel(s) attribut(s) nous allons chercher l'égalité. Par contre, si un attribut est présent dans les deux tables, il va falloir préciser auquel on fait référence en indiquant la table d'origine avant (avec le formalisme table.attribut).
En reprenant le premier exemple précédent, voici la requête reliant les produits avec les catégories, réalisée avec une jointure interne.

```
SELECT *
    FROM Produit INNER JOIN Categorie
       ON Produit.CodeCateg = Categorie.CodeCateg;
```

De même pour les jointures naturelles, il est possible de réaliser d'autres opérations, en plus de la jointure.
Si nous souhaitons refaire la requête récupérant les produits (nom du produit et du fournisseur) fournis par des entreprises françaises, nous aurons la requête suivante.

```
SELECT NomProd, Societe
    FROM Produit INNER JOIN Fournisseur
       ON Produit.NoFour = Fournisseur.NoFour
    WHERE Pays = "France";
```

Si on souhaite maintenant le nom de catégorie et le nombre de produits associés, avec une jointure interne, voici comment faire.

```
SELECT NomCateg, COUNT(*) AS "Nb Produits"
    FROM Produit INNER JOIN Categorie
       ON Produit.CodeCateg = Categorie.CodeCateg
    GROUP BY NomCateg
    ORDER BY 2 DESC;
```

IV.2.2- Alias pour les tables

Pour simplifier l'écriture des requêtes, il est possible de renommer temporairement une table (valable uniquement dans la requête présente), avec l'opérateur AS.
Reprenons la requête précédent en renommant Produit en P et Categorie en C. Ce processus est utile quand on écrit des requêtes longues.

```
SELECT *
    FROM Produit AS P INNER JOIN Categorie AS C
       ON P.CodeCateg = C.CodeCateg;
```

Il est aussi possible de ne pas indiquer le terme AS, le renommage sera tout de même pris en compte.
Ainsi, la requête précédente devient la suivante.

```
SELECT *
    FROM Produit P INNER JOIN Categorie C
       ON P.CodeCateg = C.CodeCateg;
```

IV.2.3- Jointures multiples

De même que pour une jointure naturelle, il est possible d'enchaîner les jointures internes, autant de fois que nécessaire. Ceci peut produire des requêtes très longues et difficiles à lire.

```
SELECT *
    FROM (Produit P INNER JOIN Categorie C
       ON P.CodeCateg = C.CodeCateg)
          INNER JOIN Fournisseur F
             ON P.NoFour = F.NoFour;
```

IV.2.4- Lignes manquantes

Le défaut de ce type de jointure (naturelle ou interne) est qu'une ligne d'une table n'ayant pas de correspondances dans l'autre table n'est pas conservé dans le résultat.
Si nous comptons le nombre de clients dans la table Client, et le nombre de clients différents dans la table Commande, nous voyons que ce n'est pas le même résultat. Certains client de la table Client n'ont pas de commandes associés.

```
SELECT COUNT(*)
    FROM Client;

SELECT COUNT(DISTINCT CodeCli)
    FROM Commande;
```

Exercices

1.Récupérer les informations des fournisseurs pour chaque produit, avec une jointure interne
2.Afficher les informations des commandes du client "Lazy K Kountry Store", avec une jointure interne
3.Afficher le nombre de commande pour chaque messager (en indiquant son nom), avec une jointure interne

IV.3- Jointures externes

IV.3.1- Principe

Comme indiqué précédemment, si une ligne d'une table n'a pas de correspondance dans l'autre table, celle-ci ne sera pas conservé dans le résultat.
Ainsi, les clients sans commande associée ne seront pas dans la table résultante de la jointure naturelle ou interne entre Client et Commande.
Pour cela, nous devons utiliser une jointure externe (OUTER JOIN) . Celle-ci a pour but de garder toutes les lignes des deux tables (ou d'une seule des deux).
Une jointure se fait entre deux tables. Nous parlerons de jointure externe gauche (LEFT OUTER JOIN) quand nous garderons les lignes de la table à gauche (la première citée donc). Et nous parlerons de jointure externe droite (RIGHT OUTER JOIN) quand nous garderons les lignes de la table à droite (la deuxième donc). Enfin, si l'on souhaite garder toutes les lignes des deux tables, il faut faire une jointure externe complète (FULL OUTER JOIN).
ATTENTION Dans l'outil utilisé ici, seule la jointure externe gauche est implémenter. Pour obtenir la jointure externe droite, il suffit d'inverser les tables. Pour la jointure complète, nous devrons utiliser en plus un opérateur que nous verrons plus tard.
Dans notre cas, si nous souhaitons avec les clients avec les détails de commande, tout en gardant la table de résultat les clients sans commande, nous devons donc réaliser une jointure externe gauche entre Client (à gauche) et Commande (à droite).

```
SELECT *
    FROM Client Cl LEFT OUTER JOIN Commande Co
       ON Cl.CodeCli = Co.CodeCli;
```

Si vous regardez les lignes pour les clients "ESPAC" et "ETOUR" (entre autres), vous verrez qu'il n'y a aucune information sur les attributs de la table Commande (à partir de NoCom).
De plus, nous voyons qu'il y a 835 lignes dans la table résultat, correspondant aux 830 commandes plus les 5 clients n'ayant pas de commande associée.

IV.3.2- Décompte avec prise en compte du 0

L'intérêt de ce type de jointure réside principalement dans du dénombrement, en gardant l'information comme quoi certaines modalités de dénombrement n'ont aucune ligne associée.
Le problème est que si on réalise un décompte classique (avec COUNT(*) donc), ces lignes ne sont pas gardées finalement.
Par exemple, si nous souhaitons connaître le nombre de commandes par client, il serait usuel de faire la requête suivante. On remarque qu'il y a bien tous les clients (dont ESPAC et ETOUR) mais tous avec au minimum un décompte à 1. C'est normal car COUNT(*) compte le nombre de lignes. Et il y a bien au moins une ligne par client.

```
SELECT Cl.CodeCli, COUNT(*)
    FROM Client Cl LEFT OUTER JOIN Commande Co
       ON Cl.CodeCli = Co.CodeCli
    GROUP BY Cl.CodeCli
    ORDER BY 2;
```

Si l'on veut avoir le nombre de commandes, et donc 0 pour ceux n'ayant aucune commande, il faut compter le nombre de valeurs non nulles d'un attribut, et de préférence la clé primaire de la deuxième table.
Ainsi, si nous comptons le nombre de valeurs non nulles de NoCom (avec COUNT(NoCom)), comme fait dans la requête qui suit, nous avons bien un 0 qui apparaît pour les clients n'ayant aucune commande associée.

```
SELECT Cl.CodeCli, COUNT(NoCom)
    FROM Client Cl LEFT OUTER JOIN Commande Co
       ON Cl.CodeCli = Co.CodeCli
    GROUP BY Cl.CodeCli
    ORDER BY 2;
```

Ainsi, ce processus, combiné avec une condition sur l'agrégat (avec HAVING), nous permet de pouvoir récupérer uniquement les lignes d'une table n'ayant pas de correspondance dans l'autre.
Par exemple, si nous souhaitons connaître le nom des sociétés clientes pour lesquelles nous n'avons pas de commandes associées, nous pourrions faire la requête suivante. Ici, Cl.* permet de récupérer toutes les informations de la table Client.

```
SELECT Cl.*
    FROM Client Cl LEFT OUTER JOIN Commande Co
       ON Cl.CodeCli = Co.CodeCli
    GROUP BY Cl.CodeCli
    HAVING COUNT(NoCom) == 0;
```

Maintenant, si on ajoute un autre calcul d'agrégat (somme, moyenne, ...), le résultat sera NULL pour les lignes n'ayant pas de correspondances.
Par exemple, si nous calculons les frais de port moyens pour chaque client, nous n'avons pas de résultat pour les clients n'ayant aucune commande.

```
SELECT Cl.CodeCli, COUNT(NoCom), AVG(Port)
    FROM Client Cl LEFT OUTER JOIN Commande Co
       ON Cl.CodeCli = Co.CodeCli
    GROUP BY Cl.CodeCli
    ORDER BY 2;
```

Exercices

1.Compter pour chaque produit, le nombre de commandes où il apparaît, même pour ceux dans aucune commande
2.Lister les produits n'apparaissant dans aucune commande
3.Existe-t'il un employé n'ayant enregistré aucune commande ?

IV.4- Jointures à la main

Comme dit précédemment, le défaut des jointures internes (et dans une moindre mesure naturelles) est la lourdeur du code écrit dans le cas de plusieurs jointures.

IV.4.1- Principe

* Produit cartésien

On parle de produit cartésien quand on cherche à coupler toutes les lignes d'une table avec chaque ligne de l'autre table. Considérons que la table A à nbA lignes, et la table B nbB lignes. En résultat, nous obtenons donc une table avec nbA * nbB lignes. Ce qui peut vite faire beaucoup. Par exemple, la table Client fait 94 lignes, la table Commande 830. Le produit cartésien des deux tables produit une table de 78020 lignes. Il faut donc faire attention quand on fait une telle opération.
Pour mieux comprendre le produit cartésien, on peut regarder la requête suivante. Elle associe chaque produit (84 lignes), avec chaque catégorie (8 lignes), ce qui fait une table résultat de 672 lignes. Vous remarquerez que la requête est un peu lente à être exécutée.

```
SELECT *
    FROM Produit, Categorie;
```

* Restriction sur produit cartésien

Bien évidemment, dans un produit cartésien, toutes les lignes ne sont pas intéressantes dans le résultat. Celles qui nous intéresse est celles où il y a correspondance entre certains attributs de chaque table. Une jointure est finalement une restriction sur produit cartésien, celle-ci apparaissant dans le WHERE.
Pour reprendre notre exemple précédent, pour faire correspondre chaque produit avec sa catégorie via une jointure à la main, nous allons réaliser la requête suivante.

```
SELECT *
    FROM Produit, Categorie
    WHERE Produit.CodeCateg = Categorie.CodeCateg;
```

Comme précédemment, il est aussi possible de renommer temporairement les tables pour simplifier l'écriture de la requête.

```
SELECT *
    FROM Produit P, Categorie C
    WHERE P.CodeCateg = C.CodeCateg;
```

IV.4.2- Jointures multiples

Nous allons comparer ici les requêtes permettant de réaliser de multiples jointures. Si nous souhaitons compter pour chaque client et pour chaque messager, le nombre de commandes passées, nous devons réaliser les requêtes suivantes.

* Jointure naturelle

Cette requête ne fonctionne pas, car l'attribut Tel est présent dans Client et dans Messager. Il va donc chercher l'égalité sur cet attribut aussi. Et il y a peu de chances qu'un client ait le même numéro de téléphone qu'un messager.

```
SELECT Societe, NomMess, COUNT(*)
    FROM (Client NATURAL JOIN Commande)
       NATURAL JOIN Messager
    GROUP BY Societe, NomMess;
```

* Jointure interne

Pour qu'elle fonctionne, nous devons passer par des jointures internes. Cette requête s'écrit donc de la manière suivante. La lecture n'est pas la plus aisée, surtout pour repérer les tables prises en compte.

```
SELECT Societe, NomMess, COUNT(*)
    FROM (Client Cl INNER JOIN Commande Co
           ON Cl.CodeCli = Co.CodeCli)
       INNER JOIN Messager M
           ON Co.NoMess = M.NoMess
    GROUP BY Societe, NomMess;
```

* Jointure à la main

Dans ce type de jointure, nous devons déjà lister dans le FROM les tables nécessaires, puis dans le WHERE mettre les conditions de jointures. Il en résulte une requête plus facile à décoder dans la lecture des tables prises en compte.

```
SELECT Societe, NomMess, COUNT(*)
    FROM Client Cl, Commande Co, Messager M
    WHERE Cl.CodeCli = Co.CodeCli
    AND Co.NoMess = M.NoMess
    GROUP BY Societe, NomMess;
```

IV.4.3- Limitations

Ce type de jointure ne permet pas de faire de jointure externe.

Exercices

1.Récupérer les informations des fournisseurs pour chaque produit, avec jointure à la main
2.Afficher les informations des commandes du client "Lazy K Kountry Store", avec jointure à la main
3.Afficher le nombre de commande pour chaque messager (en indiquant son nom), avec jointure à la main

Bonus :

1.Compter le nombre de produits par fournisseur
2.Compter le nombre de produits par pays d'origine des fournisseurs
3.Compter pour chaque employé le nombre de commandes gérées, même pour ceux n'en ayant fait aucune
4.Afficher le nombre de pays différents des clients pour chaque employe (en indiquant son nom et son prénom)
5.Compter le nombre de produits commandés pour chaque client pour chaque catégorie

V- Sous-requêtes

V.1- Sous-requêtes

Il est possible et souvent intéressant d'utiliser des sous-requêtes, renvoyant donc une table, dans une requête. On peut ainsi

- éviter des jointures, pouvant être parfois longue à effectuer
- faire des calculs et utiliser le résultat dans un autre
- ...

V.1.1- dans WHERE

Il est déjà possible de comparer un attribut avec le résultat d'une requête. Ici, nous cherchons les commandes du client "Bon app". On peut bien sûr réaliser cette opération avec une jointure, comme ci-dessous.

```
SELECT NoCom
    FROM Commande NATURAL JOIN Client
    WHERE Societe = "Bon app";
```

Si nous avons beaucoup de commandes et de clients, cette jointure peut prendre beaucoup de temps. On va donc chercher les commandes pour lesquelles CodeCli est égal à celui de l'entreprise "Bon app". La sous-requête ici nous permet de retrouver cette valeur.

```
SELECT NoCom
    FROM Commande
    WHERE CodeCli = (SELECT CodeCli
                        FROM Client
                        WHERE Societe = "Bon app");
```

V.1.2- Idem mais avec plusieurs retours

Si la recherche concerne plusieurs valeurs, il faut donc utiliser l'opérateur IN, qui teste si une valeur est présente dans une liste. Ici, nous cherchons les commandes des clients français, toujours possible avec une jointure.

```
SELECT NoCom
    FROM Commande NATURAL JOIN Client
    WHERE Pays = "France";
```

Pour les mêmes raisons que précédemment, on peut choisir de ne pas faire de jointure et d'utiliser une sous-requête. Celle-ci va rechercher les code des clients français. Et la requête va rechercher les commandes pour lesquelles CodeCli est dans la liste renvoyée par la sous-requête.

```
SELECT NoCom
    FROM Commande
    WHERE CodeCli IN (SELECT CodeCli
                        FROM Client
                        WHERE Pays = "France");
```

V.1.3- dans le FROM

On a aussi la possibilité de faire une sous-requête dans la partie FROM du requête. Ceci peut permettre de faire une restriction avant la jointure. Ou aussi de faire des calculs. En reprenant l'exemple du client "Bon app", on peut aussi faire la requête suivante.

```
SELECT NoCom
    FROM Commande NATURAL JOIN
        (SELECT *
            FROM Client
            WHERE Societe = "Bon app");
```

Les commandes des clients français peuvent aussi s'obtenir de cette façon.

```
SELECT NoCom
    FROM Commande NATURAL JOIN
        (SELECT *
            FROM Client
            WHERE Pays = "France");
```

Mais si on souhaite calculer le coût d'une commande, nous sommes obligé de passer par ce mécanisme. En effet, nous devons d'abord faire la somme des montants pour chaque produit et ajouter les frais de port au total. Il n'est pas possible de faire tout en une requête, car dans ce cas, en faisant la jointure entre Commande et DetailCommande, nous dupliquons les frais de port par autant de fois qu'il y a de produits différents dans la commande. Pour le faire proprement, il faut donc réaliser la commande suivante.

```
SELECT NoCom, Port + TotalProd AS Total
    FROM Commande NATURAL JOIN
        (SELECT NoCom,
                SUM(PrixUnit * Qte * (1 - Remise)) AS TotalProd
            FROM DetailCommande
            GROUP BY NoCom);
```

V.1.4- Comparaison dans une sous-requête

Il est possible de faire référence dans une sous-requête à une valeur de la table (ou des tables) de la requête initiale.

On peut par exemple chercher les produits pour lesquels il existe une vente (table DetailCommande) de celui-ci au même prix que le prix actuel (donc celui dans Produit).

```
SELECT RefProd, NomProd, PrixUnit
    FROM Produit P
    WHERE PrixUnit IN (SELECT PrixUnit
                       FROM DetailCommande
                       WHERE RefProd = P.RefProd);
```

Exercices

1.Lister les employés n'ayant jamais effectué une commande, via une sous-requête

2.Nombre de produits proposés par la société fournisseur "Mayumis", via une sous-requête

3.Nombre de commandes passées par des employés sous la responsabilité de "Patrick Emery"

V.2- Opérateur EXISTS

L'opérateur EXISTS permet de tester si une sous-requête renvoie un résultat ou non. En faisant en plus référence à une valeur d'un attribut de la table dans la première requête, cela permet de tester des existences de faits.
Par exemple, si l'on souhaite avoir les clients ayant au moins une commande, on peut faire comme ci-dessous.

```
SELECT *
    FROM Client Cl
    WHERE EXISTS (SELECT *
                  FROM Commande
                  WHERE CodeCli = Cl.CodeCli);
```

On peut faire aussi l'inverse de cette requête en cherchant les clients n'ayant pas de commande.

```
SELECT *
    FROM Client Cl
    WHERE NOT EXISTS (SELECT *
                      FROM Commande
                      WHERE CodeCli = Cl.CodeCli);
```

Dans les deux requêtes ci-dessous, il est possible de faire autrement car nous ne comparons qu'avec un seul attribut (CodeCli). On aurait pu donc passer par une joitnure ou un IN (ou NOT IN) avec une sous-requête.
Par contre, si on cherche à comparer plus d'un attribut, il devient difficile (voire impossible parfois) d'utiliser l'opérateur = ou IN, ou une jointure. On passe donc par EXISTS.
On cherche ici à trouver des clients qui habitent dans la même ville (et donc le même pays) qu'un employé.

```
SELECT Societe, Ville, Pays
    FROM Client Cl
    WHERE EXISTS (SELECT *
                  FROM Employe
                  WHERE Ville LIKE Cl.Ville
                  AND Pays LIKE Cl.Pays);
```

Exercices

1.Lister les produits n'ayant jamais été commandés, à l'aide de l'opérateur EXISTS
2.Lister les fournisseurs dont au moins un produit a été livré en France
3.Liste des fournisseurs qui ne proposent que des boissons

Bonus :

1.Lister les clients qui ont commandé du "Camembert Pierrot" (sans aucune jointure)

2.Lister les fournisseurs dont aucun produit n'a été commandé par un client français
3.Lister les clients qui ont command´e tous les produits du fournisseur "Exotic liquids"
4.Quel est le nombre de fournisseurs n'ayant pas de commandes livrées au Canada ?
5.Lister les employés ayant une clientèle sur tous les pays

VI- Opérations Ensemblistes

VI.1- Union

L'union est une opération ensembliste qui consiste à prendre les éléments présents dans A et dans B, A et B étant deux ensembles obtenus grâce à des requêtes, et réuni grâce à l'opérateur UNION. L'exemple ci-dessous renvoie tous les clients qui sont français (premier SELECT) ou dont le contact est le propriétaire de l'entreprise (deuxième SELECT).

```
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Pays = "France"
UNION
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Fonction = "Propriétaire";
```

Il faut noter que les deux SELECT doivent absolument renvoyer les mêmes champs.

VI.1.1- Ordre et limite

Si l'on souhaite faire un tri du résultat, et/ou se limiter aux premières lignes, les commandes ORDER BY et LIMIT doivent se placer tout à la fin. Par exemple, nous trions ici la réquête précédente sur le nom de la société.

```
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Pays = "France"
UNION
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Fonction = "Propriétaire"
ORDER BY 1;
```

Et si l'on veut se limiter aux 10 premières lignes, la requête devient la suivante.

```
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Pays = "France"
UNION
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Fonction = "Propriétaire"
```

```
ORDER BY 1
LIMIT 10;
```

VI.1.2- Résultats dupliqués

Il arrive que des lignes soient présentes dans les deux requêtes. Ici, cela est le cas pour les entreprises françaises dont le contact est le propriétaire de l'entreprise. Dans la version précédente, ces entreprises n'apparaissent qu'une seule fois. Si l'on veut avec les doublons, on rajoute ALL à la suite du terme UNION. Ce qui revient à la requête suivante (avec un tri sur le nom pour mieux voir les doublons).

```
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Pays = "France"
UNION ALL
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Fonction = "Propriétaire"
ORDER BY 1;
```

Exercices

En utilisant la clause UNION :

1. Lister les employés (nom et prénom) étant "Représentant(e)" ou étant basé au "Royaume-Uni"
2. Lister les clients (société et pays) ayant commandés via un employé situé à Londres ("London" pour rappel) ou ayant été livré par "Speedy Express"

VI.2- Intersection

L'intersection, avec l'opérateur INTERSECT, entre deux ensembles A et B, obtenus grâce à deux requêtes, permet de ne récupérer que les lignes présentes dans les deux ensembles. En reprenant l'exemple précédent, nous récupérons ici tous les clients français dont le contact est le propriétaire de l'entreprise.

```
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Pays = "France"
INTERSECT
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Fonction = "Propriétaire";
```

Tout comme l'union, les champs des deux SELECT doivent être identiques.
Les règles pour le tri et la limitation des résultats sont aussi les mêmes.

Alternative pour MySQL

MySQL ne propose malheureusement pas cette commande SQL, heureusement le fonctionnement de cette requête peut-être simulé grâce à une petite astuce. La requête SQL ci-dessous est l'alternative à INTERSECT :

```
SELECT DISTINCT value FROM `table1`
WHERE value IN (
  SELECT value
  FROM `table2`
);
```

A noter : la colonne "value" est à remplacer par la colonne de votre choix. La commande DISTINCT n'est pas obligatoire, mais est la plupart du temps utile pour éviter d'afficher plusieurs fois les mêmes valeurs.

Exercices

Avec l'opérateur INTERSECT :

1. Lister les employés (nom et prénom) étant "Représentant(e)" et étant basé au "Royaume-Uni"
2. Lister les clients (société et pays) ayant commandés via un employé basé à "Seattle" et ayant commandé des "Desserts"

VI.3- Différence

Enfin, la différence entre deux éléments A et B, renvoie les éléments de A qui ne sont pas présents dans B. Il faut donc faire attention à l'ordre des SELECT, puisque les résultats ne seront pas identiques entre A - B et B - A. L'opérateur est EXCEPT en SQL classique (et MINUS dans certains cas - Oracle par exemple). En reprenant le même exemple, la requête suivante renvoie les clients français dont le contact n'est pas propriétaire.

```
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Pays = "France"
EXCEPT
SELECT Societe, Fonction, Pays
    FROM Client
    WHERE Fonction = "Propriétaire";
```

De même que l'union et l'intersection, les champs des deux SELECT doivent être les mêmes.
Et le tri et la limitation doivent se faire aussi à la fin.

Exercices

Avec l'opérateur EXCEPT :

1. Lister les employés (nom et prénom) étant "Représentant(e)" mais n'étant basé au "Royaume-Uni"
2. Lister les clients (société et pays) ayant commandés via un employé situé à Londres ("London" pour rappel) et n'ayant jamais été livré par "United Package"

Bonus :

1. Lister les employés ayant déjà pris des commandes de "Boissons" ou ayant envoyés une commande via "Federal Shipping"
2. Lister les produits de fournisseurs canadiens et ayant été commandé par des clients du "Canada"
3. Lister les clients (Société) qui ont commandé du "Konbu" mais pas du "Tofu"

3- Ecrire une application web java qui se connecte à votre base de données et affiche sur une page les listes de clients, de produits et de catégories

Bonus : Afficher sur votre appli web le résultat de 5 requêtes de votre choix.