TP 3 — Sélection de données

Lors des TP précédents, nous avons vu comment créer une base de données, y ajouter des tables et afficher des données au moyen de requêtes de la forme

```SQL
SELECT …
FROM …
WHERE … -- Clause facultative, pour filtrer.
```

Dans ce TP, nous allons aller plus loin dans l'utilisation de `SELECT`, en travaillant sur une base de données *déjà existante*, que nous allons télécharger.

Bases de données préexistantes

Il est fréquent, dans l'industrie ou plus généralement en environnement professionnel, d'intervenir sur des bases de données déjà constituées, puisque le plus souvent, on ingère un projet déjà commencé.

Il est donc assez rare de construire une base de données *à partir du début*. Il arrive en revanche fréquemment de devoir intervenir pour *modifier la structure* d'une base, en ajoutant des tables ou en modifiant les propriétés de certains attributs. C'est en général une opération complexe, car il ne faut pas corrompre la base de données en cassant l'intégrité relationnelle entre les tables.

1 Mise en place

Téléchargez la base de données appelée [2]Northwind, accessible via https://but-info-iutcv.fr/northwind.sqlite3. Enregistez-la sous `R105_BDD` ▸ `TP3` ▸ `northwind.sqlite3`. Ouvrez cette base avec `DBeaver`.

À rendre

Vous devrez déposer sur EPREL l'ensemble des requêtes SQL que vous aurez écrites, sous la forme d'un fichier `tp3_<nomDeFamille>.sql` (à ne pas confondre avec le fichier `northwind.sqlite3` qui contient votre base de données).

2 Prise en main de la base `Northwind`

2.1 Liste des tables existantes

Une base de données peut contenir plusieurs tables. Voyons combien de tables nous avons dans la base de données `Northwind`. Chaque base de données SQLite a une table spéciale appelée `sqlite_master`. Elle contient une liste principale de tous les objets de base de données (tables, index, etc.) dans la base de données et le SQL utilisé pour créer chaque objet.

Nous pouvons interroger cette table pour savoir combien de tables (hors `sqlite_master`) nous avons dans notre base de données `Northwind`. Recopiez et exécutez la requête suivante (que l'on ne vous demande pas de comprendre pour le moment) :

```SQL
SELECT name FROM sqlite_master WHERE type='table' AND name NOT LIKE 'sqlite_%';
```

Notez que les tables spéciales dans SQLite commencent par `sqlite_`. Ce sont des tables réservées à l'utilisation du système de moteur SQLite. Ci-dessus, nous avons filtré ces tables. Dans la plupart des cas, nous ne devrions pas toucher à ces tables spéciales.

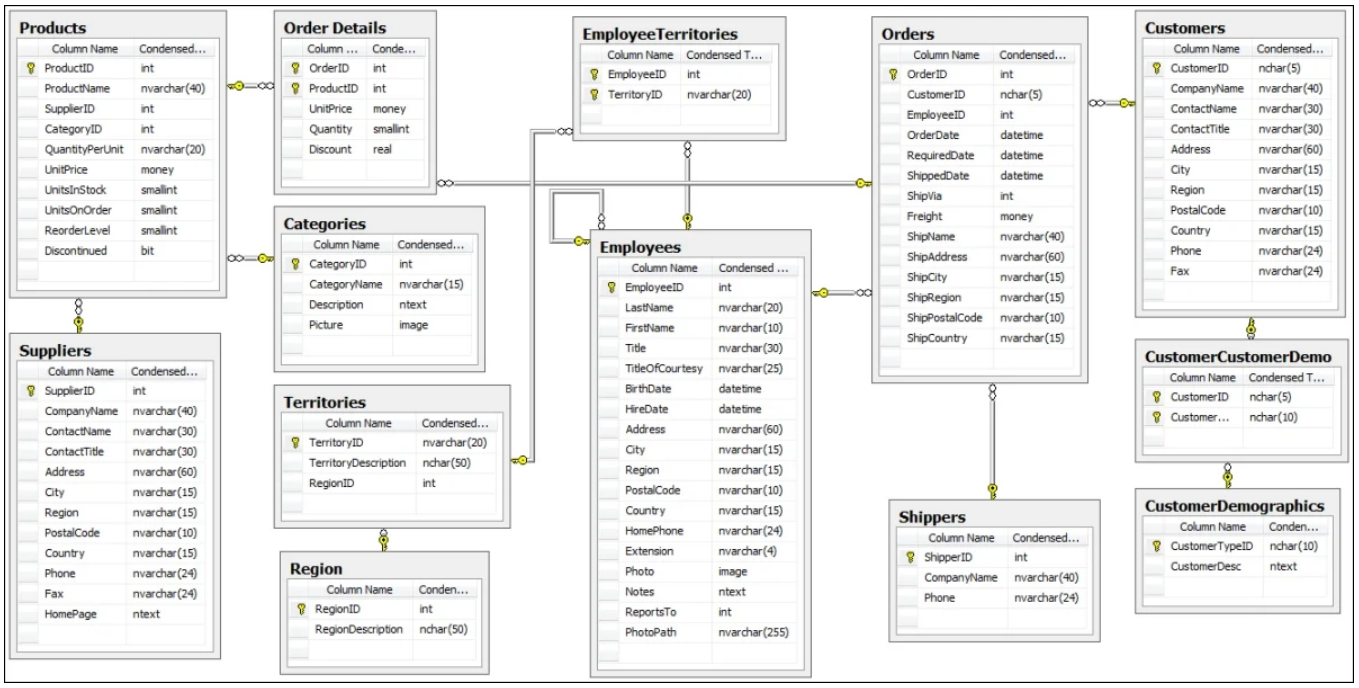
2.2 Analyse de quelques tables

Parmi les tables présentes, on trouve notamment `Categories` et `Customers`. Pour obtenir la structure de ces tables, on utilise les requêtes :

```SQL
PRAGMA table_info([Categories]); -- N'oubliez pas les crochets !
PRAGMA table_info([Customers]);  -- Ici non plus…
```

2.3 Relations entre les tables

Les relations entre les différentes tables sont présentées dans un diagramme entité-relation (vous pouvez récupérer l'image sur EPREL en plus grande taille pour faciliter la lecture) :



La présence d'une clé jaune *à l'intérieur* des encadrés indique la clé primaire de chaque table. Lorsque deux tables sont reliés, cela indique une contrainte de *clé étrangère*, avec une relation *1 à plusieurs* $(1, n)$; la clé est du côté « 1 » et le symbole ∞ du côté n.

Jusqu'à maintenant, nous n'avons rencontré que des clés primaires qui sont des identifiants (telles que `enseignant_id` dans le TP précédent, ou `Categories.CategoryID` dans la base `Northwind`). Examinons plus précisément la table `OrderDetails`. On remarque que la clé primaire est constituée

de *deux* attributs : `OrderID` et `ProductID`. Ceci est logique car la table `OrderDetails` contient (sans surprise…) les détails de chaque commande. Chaque ligne de la table est un article qui a été commandé. Toute commande qui contient strictement plus d'un article apparait donc plusieurs fois dans la table (une ligne par article) ; et un même article peut apparatre dans plusieurs commandes différentes. Dit autrement, les tables `Products` et `Orders` sont reliées par une relation *plusieurs à plusieurs* (également notée (n, n)). C'est la même situation que dans le TP 2 avec les tables `competences` et `enseignements`. Dans la table `OrderDetails`, pour identifier chaque ligne de manière unique, il faut donc préciser la valeur de *deux attributs* : l'identifiant d'une commande et l'identifiant d'un produit.

Dans cette situation, on dit que la clé (`OrderID`, `ProductID`) est une *clé composite*.

Clé primaire composite : attention au piège

Il ne faut pas penser que la table `OrderDetails` contient deux clés primaires : ce n'est pas possible et cela n'a aucun sens. En revanche, la clé primaire est constitué de deux attributs. Il n'est pas obligatoire en SQL qu'une clé primaire soit un attribut unique.

On peut définir une clé primaire comme un sous-ensemble d'attributs qui permettent d'identifier de manière unique chaque ligne de la table, sans avoir besoin de connaitre la valeur des autres attributs.

Il s'agit alors d'une contrainte qui n'est plus associée à un attribut, mais à la table dans son ensemble. La requête permettant de créer la table `OrderDetails` pourrait par exemple s'écrire :

SQL
```sql
CREATE TABLE OrderDetails(
  OrderID INTEGER,
  ProductID INTEGER,
  UnitPrice DECIMAL,
  Quantity SMALLINT,
  Discount REAL,
  FOREIGN KEY(OrderID)   REFERENCES Orders(OrderID),
  FOREIGN KEY(ProductID) REFERENCES Products(ProductID),
  PRIMARY KEY(OrderID, ProductID) -- Clé composite : c'est une contrainte de table
                                  -- On la place en fin de requête
);
```

On peut obtenir les relations entre les tables en interrogeant la table `sqlit_master` :

SQL
```sql
SELECT sql
FROM sqlite_master
WHERE name="Orders"; -- On peut mettre ici la table de son choix
```

Cette requête renvoie une seule valeur, dans une chaine de caractères.

3 Exercices

Exercice 1 (Sélections très simples). 1) Afficher `CategoryName` et `Description` de la table `Categories`.

2) Afficher tous les expéditeurs (table `Shippers`).

3) Quel est l'effet de la clause `LIMIT 10` dans la requête suivante ? Modifier la valeur pour faire des tests.

tp3_<nomDeFamille>.sql

```sql
SELECT FirstName, LastName, HireDate, Title, Country
FROM Employees
LIMIT 10;
```

4) Exécuter la requête suivante : `SELECT DISTINCT Country FROM Employees;`. À quoi le mot-clé `DISTINCT` sert-il ?

5) Quel est le rôle du mot-clé `COUNT` dans la requête suivante ?

tp3_<nomDeFamille.sql>

```sql
SELECT COUNT(DISTINCT Country) FROM Employees;
```

Il est possible d'utiliser les opérateurs logiques dans une clause `WHERE` pour combiner des conditions. Par exemple :

SQL

```sql
SELECT *
FROM etudiants
WHERE etudiant_age>=20 AND etudiant_groupe == 'B';
```

Les opérateurs logiques disponibles sont `OR`, `AND` et `NOT`. On peut également utiliser les cinq opérations arithmétiques usuelles (`+ - * / %`) et les opérateurs de comparaison (`> < > <= >= != =`).

Exercice 2. Afficher les valeurs des attributs `supplierID`, `ContactNames` et `ContactTitle` pour les personnes dont `ContactTitle` n'est pas `"Marketing Manager"`.

Exercice 3. Trouver tous les produits dont le nom contient `Chef`. On pourra utiliser la clause `LIKE` et le caractère générique `%`. Il signifie « n'importe quele chaine de caractères » (éventuellement vide). Lire ce tutoriel pour en savoir davantage. Vous pouvez aussi consulter la documentation (plus difficile à lire, mais plus précise et exhaustive).

Toujours dans une clause `WHERE`, on peut réaliser un test d'appartenance à un ensemble avec le mot-clé `IN`. Par exemple :

SQL

```sql
SELECT *
FROM relation_enseignements_cptces
WHERE competence_id IN ("C1", "C2", "C3") -- Ne garde que les compétences 1 à 3.
ORDER BY competence_id; -- Pour trier (faites des tests !).
```

Exercice 4. Trouvez toutes les commandes expédiées au Canada, au Mexique ou aux États-Unis.

Exercice 5. Trouver l'employé le plus âgé et les valeurs des attributs `FirstName`, `LastName`, `Title` et `BirthDate`.

Il peut être utile, pour améliorer la lisibilité des résultats rendus par une requête, de modifier le libellé des colonnes. Cela se fait avec le mot-clé **AS**, dans l'instruction **SELECT** :

```sql
SELECT competence_id AS "Code compétence", enseignement_id,
       rel_enseignement_cptces_coefficient AS Coefficient
FROM relation_enseignements_cptces
ORDER BY "Code compétence" DESC; -- Pour trier, en ordre décroissant.
```

Exercice 6. Afficher les champs `FirstName` et `LastName` depuis la table `Employees`, les assembler dans une nouvelle colonne appelée `FullName`, contenant le prénom suivi du nom, par exemple « DAVOLIO, Nancy ». On pourra utiliser la fonction SQL concat(), qui concatène deux chaines de caractères. Par exemple, `concat("abc", "def")` vaut `"abcdef"`.

Exercice 7. Dans la table `OrderDetails`, nous avons les colonnes `UnitPrice` et `Quantity`. Créer une nouvelle colonne (temporaire), `TotalPrice`, qui représente le produit des deux (on ignorera la colonne `Discount`).
Afficher les colonnes `OrderID`, `ProductID`, `UnitPrice`, `Quantity` et `TotalPrice`.
Trier les résultats d'abord par `ProductID` (ordre croissant) puis par `TotalPrice` (ordre décroissant). On utilisera pour cela le mot-clé **ORDER BY**. N'afficher que les articles pour lesquels le total des ventes est supérieur à 12 000 $.

Exercice 8 (Produits devant être recommandés)**.** Un produit a besoin d'être recommandé si :

```
    UnitsInStock + UnitsOnOrder <= ReorderLevel
et  Discontinued = 0
```

Afficher `ProductID`, `ProductName`, `UnitsInStock`, `UnitsOnOrder`, `ReorderLevel`, et `Discontinued`.