

INTRODUCTION AUX BASES DE DONNÉES AVEC SQL

Enseignant :	Louis RAYNAL	Établissement :	ICES
Contact :	l-raynal@ices.fr	Année :	2025-2026
Public :	L3 Maths et M2 Bio-Santé		
Ressources :	https://github.com/LouisRaynal/coursSQL		

Cours-TP 1 : Contexte et notions essentielles

1. Contexte

A peu près tous les organismes/entreprises stockent des informations en rapport avec leurs activités, sans forcément les utiliser pleinement.

Être capable d'*exploiter ces données* est une compétence recherchée dans le milieu professionnel. Cela peut se traduire par la création de tableaux de bord afin de suivre et optimiser les activités de l'entreprise, et ainsi aider à des prises de décision basées sur des données. Ce sont des missions en rapport avec les métiers de l'informatique décisionnelle (*Business Intelligence*) et de l'analyse/science des données (*Data analysis/science*).

Les informations recueillies sont stockées dans une base de données, qu'il est nécessaire de *créer, organiser et maintenir* à travers le temps. Ces missions sont celles d'un gestionnaire de base de données (*Data Manager*).

Même si vous ne vous destinez pas aux métiers ci-dessus, vous serez probablement amenés à travailler sur des bases de données ou avec des personnes les exploitant. Par exemple, dans de petits établissements de santé ou entreprises, il est possible que vous ayez plusieurs fonctions, dont celle de récupérer des informations dans une base de données selon des critères simples.

2. Notions de base

2.1. Donnée et base de données

Une **donnée** désigne toute **valeur** décrivant de manière élémentaire quelque chose de connu. Cela peut par exemple être :

- une chaîne de caractères, telle que **Dumont**, **Google** ou **Interstellar** ;
- une valeur numérique, telle que 24, 0812415474 ou -178.2 ;
- une date, telle que 08/02/2022.

Une donnée, seule, n'a pas de sens.

Une donnée est toujours associée à un **contexte d'interprétation**, qui permet de savoir ce que la donnée représente. Selon le contexte, la donnée **Dumont** peut être le nom d'un réalisateur, d'un client d'une banque ou même le nom d'une commune. Lorsque l'on sait interpréter une donnée, grâce à son contexte, elle devient alors une information. Ainsi, une **information** désigne une **valeur et son contexte d'interprétation**. (Par soucis de simplicité, dans la suite on pourra assimiler les notions de donnée et d'information).

Les informations sont issues d'un **domaine d'application**, et décrivent des objets, des personnes, des faits ou des concepts pour lesquels il est d'intérêt de collecter des données. Une chose d'intérêt sur laquelle est collectée des données s'appelle une **entité**.

Dans le domaine médical, une entité peut par exemple être un *patient* sur lequel nous recueillerons des informations telles que le nom, prénom, sexe, date de naissance.

Dans ce même domaine, une autre entité pourrait être une *consultation* dont les informations seraient la date de consultation, le nom et prénom du patient consulté et l'identité du médecin consultant.

Les entités, et donc les informations sur ces entités, sont naturellement **liées** entre elles selon le domaine d'application. Pour continuer sur l'exemple médical, la phrase

“le *patient* Chris Dumont, homme, né le 01/08/1964
a eu une *consultation* le 04/09/2022
avec la *médecin* Edite Finch”,

illustre bien le lien entre les trois entités *patient*, *consultation* et *médecin* ainsi que leurs informations.

Une **base de données** est un **ensemble d'informations liées entre elles**, et **organisées selon une structure pré-définie** lors de la conception de la base de données. La base de données stocke les informations de manière **persistante**, c'est-à-dire sur un support de stockage qui permet aux informations de persister à travers le temps, par exemple sur un disque dur, voire même sur papier.

Voici quelques exemples de bases de données :

- Un fichier .csv (*comma-separated values*) listant les informations des réservations d'un hôtel peut être considéré comme une base de données. Le fichier pourrait ressembler à

```
Marc ; Dumas ; 04/08/2021 ; 06/08/2021 ; 101
Ambre ; Adams ; 05/08/2021 ; 09/08/2021 ; 201
Jess ; Robin ; 05/08/2021 ; 15/08/2021 ; 203
```

Les informations stockées sont alors le **prénom** et **nom** du *client*, **date de début** et **de fin** de *réserve*
tion et **numéro** de la *chambre*. Chaque ligne désigne une réservation différente.

- Un registre médical est une base de données qui lie les informations sur les *patients*, les *consultations*, les *prescriptions*, les *actes médicaux* effectués, etc...
- Un annuaire téléphonique est une base de données papier comprenant **prénom**, **nom**, **adresse** et **numéro de téléphone** de *personnes*.

Les bases de données en version papier sont fréquemment utilisées, mais souffrent de nombreux inconvénients. Reprenons le cas de l'annuaire téléphonique pour en repérer certains.

1. Trouver le numéro de téléphone d'une personne est une tâche assez longue, en particulier lorsque la base contient de nombreuses entrées.
⇒ Accès aux données coûteux en temps
2. L'annuaire est indexé uniquement par nom-prénom, il est donc presque impossible de trouver le nom d'une personne uniquement grâce à son adresse.
⇒ Difficulté de recherche d'informations
3. Dès le moment où l'annuaire est imprimé, les informations qu'il contient deviennent de moins en moins exactes avec le temps (changement d'adresse, numéro de téléphone, ...).
⇒ Difficulté de mise à jour et obsolescence rapide

Il y a aussi un danger en termes de sécurité et d'intégrité de l'information. Pour un annuaire téléphonique ce n'est pas le sujet, mais stocker des informations confidentielles en version papier peut être plus dangereux.

Ces problèmes ont poussé à l'utilisation grandissante de bases de données informatisées, et notamment au développement de ce que l'on appelle des systèmes de gestion de bases de données.

2.2. Système de Gestion de Base de Données

Un **Système de Gestion de Base de Données (SGBD)** est un logiciel qui permet la gestion de l'ensemble des informations stockées dans une base de données. Il va nous permettre de créer une base de données, d'y ajouter des données, de modifier ou lire des données contenues dans une base. Cela se fait grâce à des langages d'interrogation et de manipulation de données (notamment le langage SQL).

Il y a de nombreux SGBD. Parmi les plus connus, vous trouverez :

- Oracle Database
- Microsoft SQL Server
- MySQL
- PostgreSQL
- SQLite

D'une certaine manière, le SGBD va jouer le rôle d'intermédiaire entre l'utilisateur (nous) et la base de données (les fichiers stockant les informations). L'utilisateur ne verra qu'une **représentation de la base de données**, dont la structure est définie par un **modèle de base de données**.

Il existe de nombreux modèles de base de données, tels que les modèles hiérarchiques ou les modèles en réseau, mais le modèle le plus populaire est le **modèle relationnel**. Ce dernier est le modèle que nous utiliserons dans ce cours. Lorsqu'un SGBD emploie ce type de modèle, on parle alors de **SGBDR** (Système de Gestion de Base de Données Relationnelle).

3. Le modèle relationnel

Le modèle relationnel a été proposé en 1970, par le Dr. Edgar Frank CODD dans l'article "A Relational Model of Data for Large Shared Data Banks". Ce modèle représente une base de données comme un **ensemble de tables liées entre elles**.

3.1. Table

Avant d'aller plus loin, il est nécessaire de bien comprendre la notion de **table**. Une table possède un **nom**, et est composée de **lignes** et de **colonnes**. À l'intersection d'une ligne et d'une colonne est stockée une donnée (valeur). Le **nom d'une table** désigne habituellement le type d'entité qui nous intéresse et pour laquelle nous recueillons des informations dans cette table. Chaque **ligne** (ou **enregistrement**) correspond à une entité spécifique, et chaque **colonne** désigne un type d'information recueilli par entité.

Pratique

Par exemple, la table ci-dessous a pour nom `clients`. L'entité d'intérêt est le *client d'une banque*, ici au nombre de trois car nous avons trois lignes. Pour chaque client, le **nom** et le **prénom** sont recueillis et stockés dans la table.

clients	
nom	prénom
Blanc	Clémence
Dumont	Charles
Blake	George

3.2. Clé primaire

Dans le modèle relationnel, chaque table doit inclure des informations qui identifient de manière **unique** chaque ligne de cette table. Il s'agit d'une **clé primaire**.

Pratique

Pour la table précédente `clients`, afin d'identifier de manière unique chaque ligne (c'est-à-dire chaque client), il nous faut considérer une nouvelle colonne `id_client`, qui introduit la notion d'identifiant unique du client.

clients		
id_client	nom	prénom
1	Blanc	Clémence
2	Dumont	Charles
3	Blake	George

La colonne `id_client` est la clé primaire de la table `clients`. Clémence Blanc a pour identifiant client la valeur 1. Aucun autre client ne recevra cet identifiant.

Considérons un autre exemple de table, en rapport avec les entités que sont les *comptes bancaires*. Cette nouvelle table se nomme `comptes`, et stocke les informations sur les comptes bancaires détenus par les clients. On y trouve le type de compte `type_cpt`, le `solde` du compte et l'identifiant de chaque compte `id_cpt`.

comptes		
id_cpt	type_cpt	solde
101	épargne	500.00
102	courant	250.00
103	courant	300.00
104	épargne	0.00
105	courant	1000.00

La clé primaire de la table `comptes` est la colonne `id_cpt`.

Notez que la notion “détenus par les clients” n’est pas présente dans la table `comptes`. Il n'existe (pour l'instant) aucun lien entre les tables `clients` et `comptes`.

Remarque

Une clé primaire n'est pas forcément composée d'une seule colonne. Plusieurs colonnes peuvent être considérées conjointement afin de former une clé primaire. On parle alors de **clé primaire composite**.

Dans l'exemple de la table `clients`, nous aurions pu prendre conjointement les colonnes `nom` et `prénom` comme clé primaire (composite donc), mais cela n'aurait pas été un bon choix, car dans le futur il aurait été possible qu'un nouveau client porte le même nom et prénom qu'un client déjà existant.

3.3. Clé étrangère et liens entre les tables

Afin d'établir des **relations** entre deux tables, il est nécessaire que des valeurs de clé primaire d'une première table apparaissent dans la deuxième table. On parle alors de **clé étrangère** pour cette deuxième table. Les valeurs des clés primaires et étrangères permettent donc d'établir des liens entre les lignes des différentes tables, et ainsi de reconstituer les informations stockées dans la base.

Pratique

Dans notre exemple bancaire, pour pouvoir déterminer par qui est détenu un certain compte, il est nécessaire d'ajouter une nouvelle colonne dans la table **comptes**. Cette colonne sera une clé étrangère pour cette table, et permettra d'accueillir les valeurs de la clé primaire de la table **clients**. Ainsi, on introduit la colonne **fk_id_client**, (où **fk** fait ici référence à *foreign key*).

id_client	nom	prénom
1	Blanc	Clémence
2	Dumont	Charles
3	Blake	George

id_cpt	type_cpt	fk_id_client	solde
101	épargne	1	500.00
102	courant	1	250.00
103	courant	2	300.00
104	épargne	3	0.00
105	courant	3	1000.00

fk_id_client est une clé étrangère pour la table **comptes**.

Nous pouvons maintenant faire le lien entre les lignes des tables **clients** et **comptes** (le trait plein représente ce lien).

Si l'on veut par exemple déterminer à qui appartient le compte épargne dont l'identifiant **id_cpt** est 104, il faut consulter la valeur de **fk_id_client** qui se trouve sur cette même ligne (ici 3) ; et rechercher cette valeur dans la colonne **id_client** de la table **clients** afin de récupérer **prénom** et **nom** associés (ici George Blake). Ce processus de mise en lien entre les valeurs d'une clé primaire et d'une clé étrangère s'appelle une **jointure** (notion développée dans un prochain cours).

Au cas où c'est encore flou, une **clé étrangère** d'une table est une colonne qui contient des valeurs d'une clé primaire d'une autre table.

Remarque

Notez qu'une clé étrangère n'est pas une clé primaire, car il est possible d'avoir plusieurs valeurs identiques dans une clé étrangère. Dans notre exemple, lorsqu'un client détient plusieurs comptes au sein de la banque.

Naturellement, une base de données structurée par le modèle relationnel n'est en rien limitée à deux tables. Autant de tables que de types d'entité d'intérêt peuvent être considérées.

Pratique

Une banque n'est pas seulement intéressée par les informations sur ses **clients** ou sur les **comptes** détenus. Les *transactions* effectuées sur les comptes sont aussi stockées. Ainsi le modèle de la base de données sera constitué d'une table supplémentaire : **transactions**, qui stockera pour chaque transaction effectuée, son type dans **type_tran**, son **montant** et sa **date**. La clé primaire de cette table est **id_tran**, qui est l'identifiant unique de chaque transaction. De plus, une clé étrangère **fk_id_cpt** est présente afin d'établir le lien entre les tables **comptes** et **transactions**.

clients		
id_client	nom	prénom
1	Blanc	Clémence
2	Dumont	Charles
3	Blake	George

comptes			
id_cpt	type_cpt	fk_id_client	solde
101	épargne	1	500.00
102	courant	1	250.00
103	courant	2	300.00
104	épargne	3	0.00
105	courant	3	1000.00

transactions				
id_tran	type_tran	fk_id_cpt	montant	date
501	débit	102	55.00	2022-01-02
502	crédit	105	10000.00	2022-02-05
503	débit	105	9000.00	2022-02-06
504	débit	103	200.00	2022-02-08
505	crédit	101	500.00	2022-03-09

Les liens entre ces trois tables permettent de retrouver toutes les informations stockées dans la base de données.

Par exemple, nous pouvons maintenant retrouver l'identité du client ayant effectué une transaction de type crédit d'un montant de 10000 euros, en passant d'une table à l'autre grâce aux valeurs de clés primaires et étrangères.

3.4. Normalisation

Le modèle relationnel se base sur des tables séparées par type d'entité, **afin d'éviter le stockage d'informations redondantes dans une même table** (autres que les valeurs de clés étrangères).

Pratique

Prenons un exemple à ne pas faire, d'une base de données avec une mauvaise structure. Considérons une table qui contient à la fois les informations sur les **clients** et les **comptes** :

id_cpt	type_cpt	nom	prénom	solde
101	épargne	Blanc	Clémence	500.00
102	courant	Blanc	Clémence	250.00
103	courant	Dumont	Charles	300.00
104	épargne	Blake	George	0.00
105	courant	Blake	George	1000.00

Notez que le nom et prénom des clients sont répétés plusieurs fois. Il s'agit d'une information redondante. Si un client change de **nom**, il faudra le changer à chaque endroit dans la table ci-dessus ce qui peut générer des incohérences dans la base de données (en cas d'oubli de modification par exemple).

Avoir une table **clients** séparée, avec une seule ligne où les informations **nom** et **prénom** sont stockées est beaucoup plus pratique. Il suffirait alors de changer une seule valeur en cas de changement de nom d'un client.

Le processus qui consiste à (ré)organiser une base de données afin de s'assurer que chaque information se trouve dans seulement une table de la base (excepté pour les clés étrangères) s'appelle le processus de **normalisation**.

Remarque

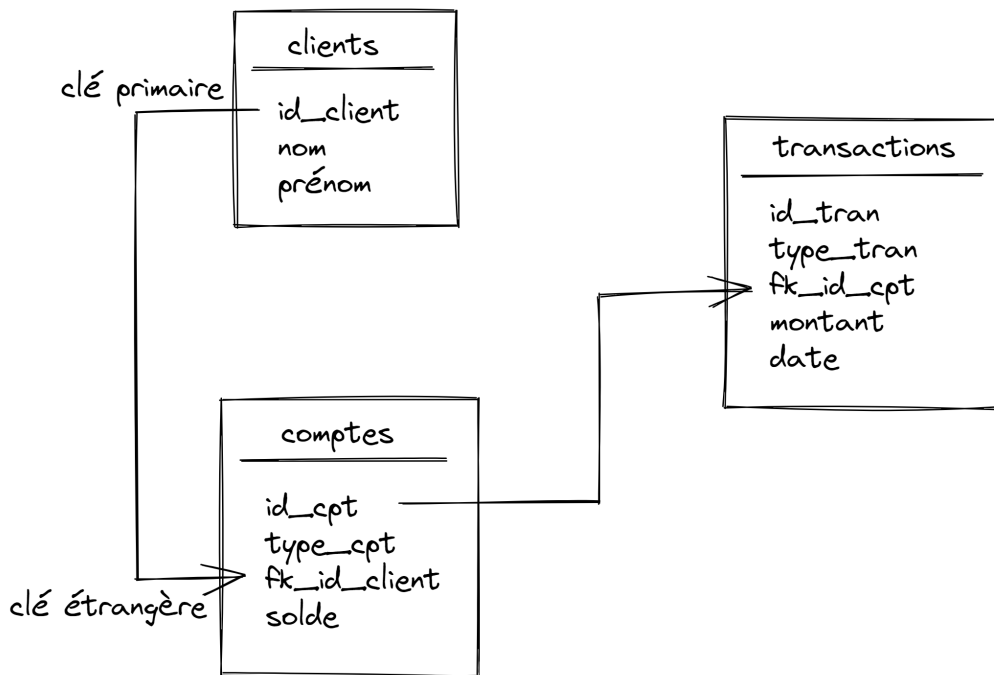
La base de données bancaires précédente n'est pas complètement normalisée. On pourrait par exemple créer une table **types_cpt** qui contiendrait les types de comptes possibles, et utiliser des identifiants pour chaque type de compte au lieu des mots "épargne", "courant". De même pour le type de transactions.

Cependant, il y a un moment où le *design* d'une base de données est assez normalisé. Créer des tables de nomenclatures pour les types de comptes et types de transactions n'a ici pas grand intérêt, d'autant plus qu'il n'y a pas d'informations particulières à recueillir les concernant.

3.5. Schéma relationnel

Lorsqu'une base de données est en phase de conception, ou pour aider l'utilisateur à comprendre comment une base de données existante est organisée selon le modèle relationnel (en termes de relations entre les tables), il est courant que la base soit accompagnée de ce que l'on appelle un **schéma relationnel**. C'est un schéma qui n'affiche pas les données des tables, mais représente uniquement **le nom des tables**, **le nom des colonnes** (voir le type des colonnes), et **les relations entres les tables**. Ces relations sont visualisées par des flèches pointant d'une **clé primaire** vers une **clé étrangère**.

Pour la base de données bancaires précédente, le schéma relationnel est le suivant :



4. SQL

Chaque SGBDR utilise un langage qui lui est propre pour permettre à l'utilisateur de créer, manipuler et interroger une base de données. Cependant, tous se basent sur le langage SQL (ou SEQUEL).

Créé dans les années 1970, le langage SQL est un **langage** informatique **de requête**. Il permet d'interagir avec les données d'une base de données relationnelle au travers de codes, aussi appelés **requêtes**. Le SQL va de pair avec le modèle relationnel, car le résultat d'une requête SQL est (bien souvent) une **table** non-persistante. Non-persistante signifie ici qu'elle n'est pas sauvegardée en mémoire.

C'est un langage nécessaire pour les métiers de la science des données (*Data Science*), informatique décisionnelle (*Business Intelligence*) ou autres métiers de l'analyse des données. Bien que nous allons apprendre le langage SQL propre à un SGBDR en particulier, la syntaxe générale restera la même peu importe le SGBDR.

Le SQL utilise des **commandes** divisées en quatre catégories distinctes (ou langages) :

1. Le **Langage de Définition de Données (LDD)** permet de définir la structure des tables et de la base de données en général.
Exemples : les commandes **CREATE TABLE** (pour créer une table) ou **DROP TABLE** (pour supprimer une table).
2. Le **Langage de Manipulation de Données (LMD)** permet d'insérer, de mettre à jour, de supprimer ou d'interroger des valeurs de données.
Exemples : les commandes **INSERT** (pour insérer des données) ou **SELECT** (pour interroger des données de tables).
3. Le **Langage de Contrôle de Transactions (LCT)** permet d'exécuter plusieurs requêtes à la fois.
Exemples : **BEGIN** (pour commencer une transaction multi-requête) ou **COMMIT** (pour terminer et accepter une transaction).
4. Le **Langage de Contrôle de Données (LCD)** permet d'accorder ou supprimer des droits sur la base aux utilisateurs. Certaines commandes LDD ou LMD seront utilisables ou non selon vos droits sur la base. Cet aspect de SQL ne sera pas traité dans ce cours.

Pratique

Pour vous donner un aperçu de ce à quoi ressemble des requêtes écrites en SQL, voici quelques exemples simples de codes pour effectuer divers tâches.

Ci-dessous la commande LDD **CREATE TABLE** est utilisée pour créer une table **clients**, qui accueillera comme informations sur les patients : l'identifiant du client **id_client**, un **nom** et un **prénom**. Une contrainte spécifie que **id_client** est une clé primaire pour cette table.

Code 1 – Création d'une table **clients**

```
1 CREATE TABLE clients
2 (
3     id_client INTEGER PRIMARY KEY,
4     nom       VARCHAR(100),
5     prénom    VARCHAR(100)
6 )
7 ;
```

Pour insérer une ligne dans cette table **clients** nouvellement créée, nous utilisons l'instruction LMD **INSERT INTO**. Les valeurs 1, Blanc, Clémence sont respectivement insérées dans les colonnes **id_clients**, **nom** et **prénom**.

Code 2 – Insertion d'une ligne de données dans la table **clients**

```
1 INSERT INTO clients (id_client, nom, prénom) VALUES (1, 'Blanc', 'Clémence')
2 ;
```

Une fois la table créée et remplie, vous pouvez interroger la table **clients**, pour récupérer le nom et prénom du client dont l'identifiant est égal à 1.

Code 3 – Sélection d'informations dans la table **clients**

```
1 SELECT nom, prénom
2 FROM clients
3 WHERE id_client = 1
4 ;
```

5. Choix du SGBD : SQLite

Parmi les nombreux SGBD disponibles, certains stockent de grosses bases de données sur des serveurs et supportent l'accès à ces bases par des milliers d'utilisateurs et applications simultanément. C'est le cas d'Oracle Database, Microsoft SQL Server, MySQL, PostgreSQL.

D'autres SGBD stockent une base dans un simple fichier enregistré sur votre ordinateur, qui est accessible par un faible nombre d'utilisateurs. C'est le cas de **SQLite**, qui est le SGBD que nous utiliserons pendant ce cours.

SQLite a de nombreux avantages pour l'apprentissage du langage SQL.

1. Il est gratuit et facilement installable sur la majorité des systèmes d'exploitation.
2. Il ne nécessite aucune configuration, contrairement à la mise en place d'un serveur.

3. Il stocke une base de données dans un simple fichier `.db` pouvant être lu peu importe le système d'exploitation.
4. Il est léger en termes d'utilisation des ressources de l'ordinateur.

6. Interface : SQLiteStudio

Dans ce cours nous utiliserons le logiciel **SQLiteStudio**. Il s'agit d'une interface graphique à SQLite, comprenant notamment un éditeur SQL.

D'une manière similaire à RStudio qui est une interface graphique pour le langage R, le développement de SQLiteStudio n'a pas de lien direct avec le développement de SQLite.

Pour installer SQLiteStudio :

1. Rendez-vous à l'adresse <http://sqlitestudio.pl/?act=download>.
2. Cliquez sur "Download" en vous assurant que l'icône est bien celle de votre système d'exploitation et enregistrez le fichier.
3. (Windows) Décompressez ensuite le fichier `.zip` téléchargé dans le dossier de votre choix. (Ou lancez l'exécutable selon la version).
(Mac OS X) Ouvrez le fichier `.dmg` téléchargé, et déplacez son contenu dans le dossier de votre choix.
4. Vous n'avez plus qu'à lancer SQLiteStudio en double cliquant sur `SQLiteStudio(.exe/.app)` .

Remarque

En cas d'un problème d'ouverture du logiciel sur Mac OS X (message : *impossible d'ouvrir SQLiteStudio car le développeur ne peut pas être vérifié*) :

- soit cliquer sur l'icône de l'application en maintenant la touche CTRL ;
- soit aller dans "Préférences système", "Sécurité et confidentialité", "Général" et cliquer sur "Ouvrir quand même".

7. Découverte de SQLiteStudio

Cette dernière partie est dédiée à la découverte du logiciel SQLiteStudio et de son interface.



Pratique

Commencez par ouvrir SQLiteStudio.

7.1. Première base de données

Pour nous aider dans la découverte de SQLiteStudio, nous allons télécharger une première base de données puis s'y connecter via SQLiteStudio. Cette base s'appelle `rexon_metals` * et contient des informations relatives à des *compagnies* (`CUSTOMER`) qui ont effectué des *commandes* (`CUSTOMER_ORDER`) de *métaux* (`PRODUCT`).

Pratique

- Rendez-vous à l'adresse <https://github.com/LouisRaynal/coursSQL>.
- Téléchargez le fichier `rexon_metals.db` situé dans le dossier TP1, et sauvegardez le sur votre ordinateur.
- Dans SQLiteStudio, ajoutez une base de données avec le symbole , puis sélectionnez le fichier `rexon_metals.db` sauvegardé via le symbole  et validez.


* Source : Livre *Getting Started with SQL* (O'Reilly)

SQLiteStudio est divisé en deux parties.


1. A gauche, une partie “Base de données” qui permet de naviguer dans les différentes bases de données disponibles (**rexon_metals** devrait être présente).
2. A droite, une partie “Environnement de travail” grisée lorsque rien n’est ouvert, qui affichera les différentes fenêtres notamment l’éditeur SQL et les fenêtres d’exploration des tables. Vous pouvez basculer d’une fenêtre à l’autre via la barre horizontale inférieure.

Pratique

Prenez un peu de temps pour explorer la base **rexon_metals**.

- Connectez-vous à cette base, pour cela cliquez sur le nom de la base, puis sur  pour créer la connexion.
- Trois tables et deux vues constituent la base de données. (Une vue est une table résultant d’une requête sur des tables de la base). Double cliquez sur le nom d’une table afin d’ouvrir sa fenêtre d’édition.
Explorez les différents onglets de cette nouvelle fenêtre. Vous pouvez notamment voir la “Structure” de la table (nom, type et contraintes sur chaque colonne) et les “Données” qui la peuplent.
- Les deux vues dans la table sont issues d’une requête SQL, dont le résultat a été stocké dans la base pour pouvoir être plus facilement accessible. Double cliquez sur une vue pour permettra notamment d’accéder à la “Requête” qui a permis d’obtenir cette vue, ainsi que les “Données” qui la peuplent.


7.2. Première requête SQL

Afin de créer des requêtes en SQL nous avons besoin de l’éditeur intégré à SQLiteStudio. Pour ouvrir cet éditeur, cliquez sur . Cela ouvre une nouvelle fenêtre nommée “Éditeur SQL 1”.

Cette fenêtre contient :

- un onglet “Requête” qui est l’éditeur de SQL et permet de rédiger et lancer des requêtes ;
- un onglet “Historique” qui liste les requêtes exécutées.



Pratique

Dans l’onglet “Requête”, réécrivez le code suivant puis exécutez le avec .

Code 4 – Sélection de toutes les données de la table **CUSTOMER**

```
1 SELECT *
2 FROM CUSTOMER
3 ;
```

Un onglet “Table” s’est ouvert avec le résultat de la requête. Ici nous avons sélectionné toutes les lignes et colonnes de la table **CUSTOMER**.

Prenez l’habitude de sauvegarder vos requêtes dans un fichier **.sql** (icône ). Réouvrez ce fichier via . (Pensez à transférer vos fichiers sur votre Google drive ou autres moyens de stockage à la fin du cours).

8. Exercices

En utilisant la base de données `rexon_metals` :

1. identifiez la clé primaire et la ou les clés étrangères des tables `CUSTOMER`, `CUSTOMER_ORDER` et `PRODUCT` (lorsqu'elles existent) ;
2. dessinez le schéma relationnel liant ces trois tables ;
3. trouvez le nom de la compagnie cliente ayant commandé de l'acier (steel), (*le faire en visualisant le contenu des tables*) ;
4. trouvez le prix des produits commandés par le client Re-Barre Construction, (*le faire en visualisant le contenu des tables*).