TP 5 : Bases de données - SQL

I. Commandes SQL (une enquête sur les *Panama Papers*)

Cette deuxième partie du cours est rédigée sous forme d'un TP. Nous allons y découvrir le langage SQL qui est construit pour dialoguer avec des bases de données relationnelles.

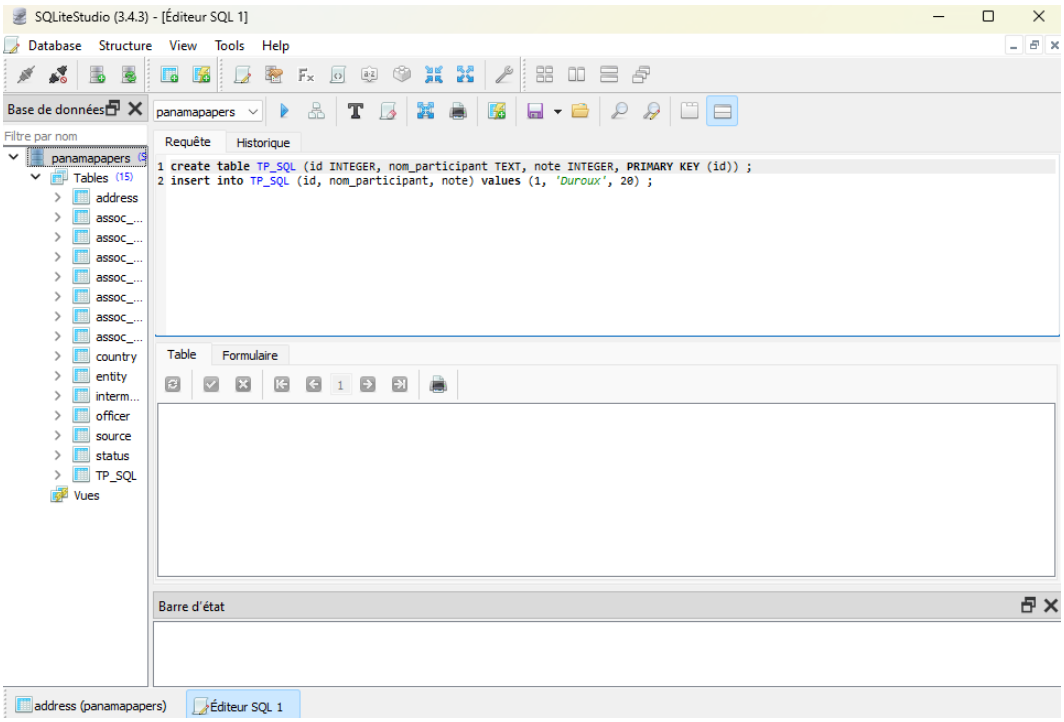
Nous n'entrerons pas (trop) dans les stratégies de création de données. Nous supposerons que la base de données est déjà construite et déjà remplie. Ce qui nous intéresse ici c'est d'interroger ces données.

I.1. Manipuler SQL sur sa machine

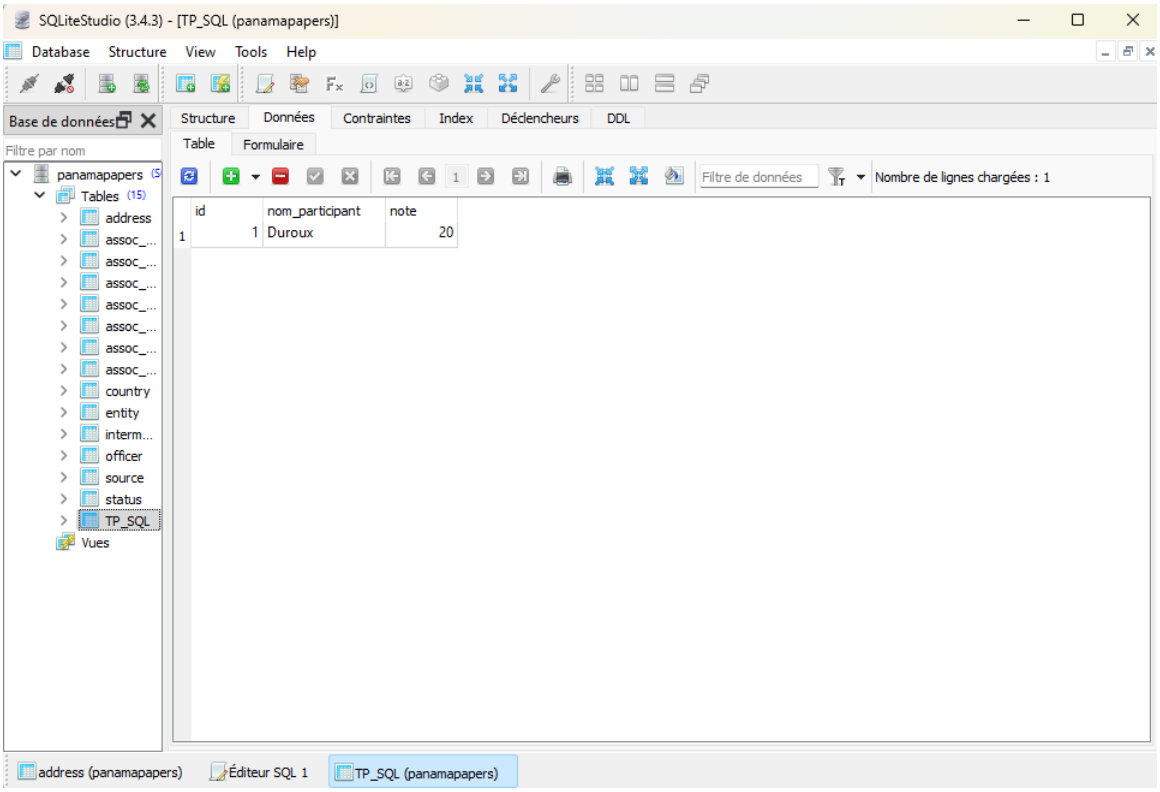
Avant de commencer, il faudra télécharger un interpréteur de commandes SQL et la base de données que nous allons utiliser.

- **[BDD]** La base de données est disponible sur ma page Internet à cette adresse. Elle est écrite dans un format .sqlite3 (qui est en fait un groupe de fichiers .csv), qui est bien sûr compatible avec le logiciel SQLite (que nous allons présenter ci-après). Attention, le fichier est gros (environ 80MB).
- **[SQLite]** Pour communiquer (et modifier) avec cette base de données, on utilise le langage SQL (*Structured Query Language*). Pour que les requêtes écrites dans ce langage soient *interprétées* et exécutées, nous avons besoin d'un interpréteur de langage.
  On a choisi SQLite, qui est gratuit (le code est en *OpenSource*) et comme son nom l'indique extrêmement léger (tant dans la taille du fichier source que dans son utilisation). Si SQLite est déjà installé sur certaines versions de macOS, on le trouvera ici.
- **[Interface]** On pourrait se contenter d'une manipulation des requêtes SQL via l'environnement `>sqlite3` d'un terminal, mais on c'est un peu austère. On décide donc d'utiliser un gestionnaire de base de données avec une interface plus intuitive. Il sera plus facile et plus agréable d'y taper du script SQL et de voir immédiatement l'effet sur les tables de la BDD. On a choisi le logiciel SQLite Studio, disponible gratuitement ici.

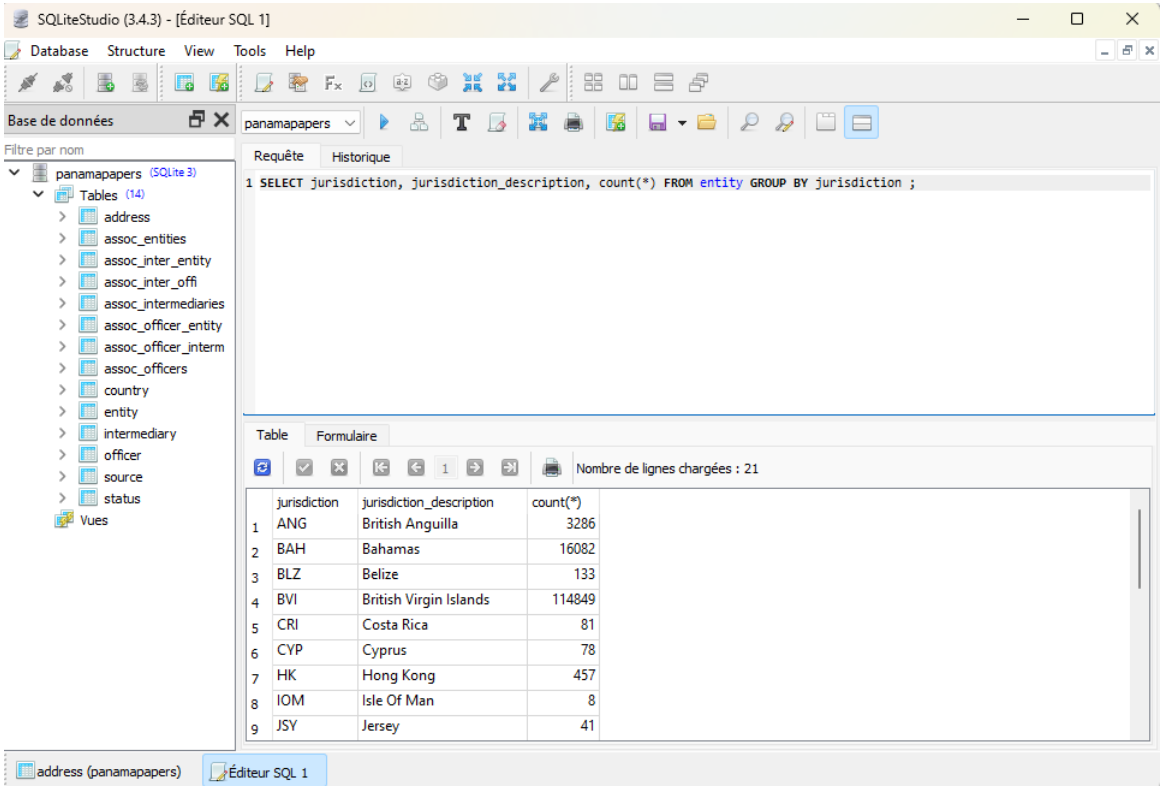
On peut écrire les requêtes dans une fenêtre script SQL.



On peut visualiser le contenu d'une table et le schéma de la BDD.



On peut visualiser le résultat d'une requête.



II. Introduction : le contexte

En avril 2016, le journal allemand *Süddeutsche Zeitung* ainsi que le Consortium International des Journalistes d'Investigation (ICIJ) publient des documents confidentiels provenant d'un cabinet d'avocats panaméen.

Cette publication fait grand bruit à travers le monde, car les documents sur lesquels ont enquêté les journalistes du consortium international révèlent des informations sur plus de 214 000 sociétés offshores ainsi que le nom des actionnaires de celles-ci. C'est l'affaire des Panama Papers.

Si l'affaire a fait tant de bruit, c'est parce qu'elle dévoile un système complexe, massif et secret permettant à des entreprises ou à des particuliers de cacher de grosses sommes d'argent sur des comptes bancaires. Ces comptes sont généralement situés dans des pays où la législation est avantageuse, que ce soit en termes de secret bancaire, de taxation, ou de contrôle de la provenance de l'argent. Ce phénomène est appelé l'évasion fiscale.

Si ces pratiques sont souvent légales, l'opinion publique les voit en général d'un mauvais œil. En effet, des sommes d'argents générées au sein d'un État donné (les bénéfices d'une entreprise par exemple) sont taxées par ce même Etat. Le fruit de cette taxation est redistribué (entre autres) aux services publics dont bénéficie la population du pays en question.

Cependant, l'évasion fiscale consiste à transférer les bénéfices du pays d'origine vers des pays à législation avantageuse (appelés les paradis fiscaux). Ainsi, les sommes d'argent générées dans un pays donné échappent en partie à l'impôt, et ne bénéficient donc plus aux populations locales. Dans certains pays, le montant estimé de l'évasion fiscale est égal ou supérieur au budget annuel de l'Etat, lorsqu'en parallèle leurs hôpitaux peinent à assurer les soins nécessaires (source : ICIJ).

Dans les *Panama Papers* se trouvaient par exemple les noms de plusieurs responsables politiques à travers le monde. Certains d'entre eux ont dû démissionner suite à la pression de l'opinion publique. Mais les *Panama Papers* ont aussi mis en lumière des moyens de financement de réseaux criminels ou terroristes.

En France, l'affaire a été révélée par 2 groupes de journalistes : Premières Lignes Production (qui a réalisé ce documentaire), et Le Monde.

III. La base de données

Les *Panama Papers* sont composés de près de 11,5 millions de documents (emails, courriers, contrats, etc.), pour un volume d'environ 2 Go. De ces documents écrits, l'ICIJ a tenté d'extraire les informations essentielles grâce à des algorithmes. Le résultat de cette extraction a été placé dans une base de données rendue publique.

Cette base n'est pas exacte, elle contient par exemple beaucoup de doublons et de champs erronés.

Grossièrement, la BDD des *Panama Papers* contient des sociétés offshores. Celles-ci sont créées pour des bénéficiaires par des fournisseurs de services offshores. Des intermédiaires se chargent généralement de faire le lien entre les bénéficiaires et les fournisseurs de services offshores.

Il y a 4 tables principales dans la base de données.

1. La table **entity**. C'est elle qui contient les sociétés offshores.
2. La table **intermediary**, qui contient les intermédiaires.
3. La table **address** qui contient les adresses de certaines sociétés intermédiaires.
4. La table **officer**, contenant entre autres les bénéficiaires des sociétés.

Ces tables contiennent les données publiées par l'ICIJ, auxquelles ont été ajoutées quelques données fictives spécialement pour ce TP, notamment la société *Big Data Crunchers Limited*. Elle a été créée de toutes pièces pour servir de fil rouge.

Commentaire

- Une société peut être domiciliée dans un pays, mais être enregistrée dans un autre. Dans ce cas, cette société répondra à la juridiction dans laquelle elle est enregistrée, même si son adresse officielle n'est pas dans cette juridiction.
- Souvent, les termes *juridiction* et *pays* sont confondus. En général, les lois sont les mêmes à l'intérieur d'un même pays. Mais parfois, un pays possède plusieurs juridictions : c'est souvent le cas des états fédéraux, dans lesquels chaque état possède des lois différentes. Par exemple, l'état du Delaware aux USA est souvent considéré comme un paradis fiscal, car les lois y sont plus avantageuses pour les sociétés que dans les autres états des USA.

Un TP dont vous êtes le héros

On se propose de se mettre dans la peau d'un enquêteur qui enquête sur le financement d'un réseau criminel.

Vous avez au cours de votre enquête intercepté une facture émise par une mystérieuse société qui s'appelle *Big Data Crunchers Limited*. Sur cette facture, l'adresse de cette société n'est pas indiquée. Vous ne savez pas qui se cache derrière cette société, mais vous pensez qu'elle peut être une société écran. Une société écran ne se crée pas si facilement que cela. En général, il faut demander de l'aide à des services spécialisés. On les appellera ici des intermédiaires.

Vous allez donc enquêter sur cette mystérieuse société, mais aussi sur les intermédiaires qui ont aidé à la créer, car vous pensez qu'il sera peut-être possible de les accuser de complicité.

IV. Un peu de vocabulaire

Définition

En économie, une **société** est la forme juridique la plus répandue des entreprise. Elle est créée dans un but marchand, à savoir, produire des biens ou des services pour le marché, qui peut être une source de profit ou d'autres gains financiers pour son ou ses propriétaires ; elle est la propriété collective de ses actionnaires, qui ont le pouvoir de désigner les administrateurs responsables de sa direction générale. (Source : INSEE)

Définition

Une **société écran** est une société fictive, créée pour dissimuler les transactions financières d'une ou de plusieurs autres sociétés. (Source : Wikipedia)

Définition

Un intermédiaire est dans la plupart des cas une personne ou un cabinet d'avocats agissant pour des clients recherchant un fournisseur de services offshores ou demandant la création d'une société offshore. (Source : ICIJ)

Définition

Un fournisseur de services offshore (en anglais : *offshore service provider* ou *agent*) est une société qui fournit des services dans une juridiction offshore, sur demande d'un client. Ces services peuvent être la création, l'enregistrement ou la gestion de sociétés offshores. (Source : ICIJ)

Définition

Un bénéficiaire (en anglais : *beneficial owner* ou *beneficiary*) est la personne réellement propriétaire de la société. Dans le monde offshore, l'identité du bénéficiaire est souvent gardée secret. (Source : ICIJ)

Commentaire

L'ICIJ tient à préciser à toute personne souhaitant utiliser la base de données les points suivants.

1. L'utilisation de sociétés offshores et de trusts n'est pas toujours illégale. Les personnes, sociétés ou autres entités citées dans la base de données n'ont donc pas forcément enfreint la loi ou agi de manière illégitime.
2. Beaucoup de personnes ou entités ont des noms similaires. Avant de conclure que deux noms correspondent à la même personne ou entité, il est conseillé de vérifier leurs adresses respectives ou toute autre information pertinente.
3. En cas d'erreur dans la base de données, prendre contact avec l'ICIJ.

V. Préambule : création et modification d'une table

- On commence par lancer SQLite Studio, charger la base de données panamapapers.sqlite3 et s'y connecter.
- Les tables sur lesquelles nous allons travailler sont déjà créées dans la BDD. Néanmoins, il faut savoir créer une table et y ajouter des entrées.
  - La commande CREATE TABLE permet de créer une table et de renseigner son nom (sur la première ligne) et les différents attributs.
  - On peut spécifier le type de chaque attribut, une chaîne de caractères (TEXT), un nombre entier (INTEGER), un réel (FLOAT), une date (DATE, au format AAAA-MM-JJ), ou aussi un booléen (un objet qui prend 2 valeurs, TRUE ou FALSE - BOOLEAN).
  - Le mot-clé NOT NULL nous empêche de créer une ligne sans renseigner l'attribut correspondant, indispensable notamment pour la PK.

  Commentaire

  Les mots-clés SQL sont insensibles à la casse mais ils sont souvent écrits en majuscules.

  Commençons par créer une table (que l'on effacera ensuite car elle ne nous servira pas pour autre chose que manipuler la commande introduite ci-dessus) intitulée TP_SQL.

```
1  CREATE TABLE TP_SQL (
2          id INTEGER,
3          nom_etudiant TEXT NOT NULL,
4          exp_sql BOOLEAN,
5          khube BOOLEAN,
6          date_TP DATE,
7          note FLOAT,
8          PRIMARY KEY(id)
9  )
```

  Après avoir exécuté la requête, on voit la table TP_SQL dans le schéma de la BDD.
- La table est alors vide. On va la remplir.
  - Pour insérer une ligne dans une table, on utilise la commande INSERT INTO suivie du nom de la table, puis, entre des premières parenthèses la liste des attributs que l'on souhaite compléter, ensuite, le mot clé VALUES suivi d'une autre paire de parenthèses qui contiennent les valeurs que l'on souhaite insérer.
  - Ces valeurs doivent être dans le même ordre que le nom des attributs.
  - Si certaines valeurs sont laissées vides, elles ne contiennent aucune valeur.

Insérer alors une ligne avec l'identifiant 0, votre nom, si vous avez déjà fait du sql, si vous êtes khube ou non et la date.

```sql
INSERT INTO TP_SQL (id, nom_etudiant, exp_sql, khube, date_TP)
        VALUES (0, 'Duroux', TRUE, TRUE, '2023-03-31') ;
```

- Insérer une autre ligne avec les informations relatives à votre voisin.

```sql
INSERT INTO TP_SQL (id, nom_etudiant, exp_sql, khube, date_TP)
        VALUES (1, 'Legendre', FALSE, FALSE, '2023-03-31') ;
```

- On veut maintenant supprimer des lignes (puis la table).
  - La commande DELETE FROM nom_de_la_table WHERE condition permet de supprimer toutes les lignes de la table où la condition est vérifiée.

  Supprimer la ligne de votre voisin. Vérifier qu'elle a disparu.

```sql
DELETE FROM TP_SQL WHERE nom_etudiant = 'Legendre'
```

- Supprimer la ligne vous concernant.

```sql
DELETE FROM TP_SQL WHERE nom_etudiant = 'Duroux'
```

- La table ne contient plus aucune ligne mais est encore présente dans le schéma de la BDD.
  - La commande DROP TABLE nom_de_la_table permet de supprimer une table du schéma de la BDD.

  Supprimer la table TP_SQL.

```sql
DROP TABLE TP_SQL
```

VI. La partie pratique

- La commande SELECT sert à dialoguer avec la BDD. À chaque exécution d'une requête avec SELECT, le logiciel renvoie une table.
  Exécuter la requête suivante.

```sql
SELECT * FROM entity ;
```

  À quoi sert le caractère * derrière SELECT ?

> Le caractère * derrière SELECT signifie que l'on souhaite obtenir toutes les lignes et toutes les colones disponibles de la table `entity`.

- Comparer la requête précédente avec la suivante.

```
1  SELECT DISTINCT * FROM entity ;
```

  À quoi sert le mot-clé DISTINCT ?

  - Le mot-clé DISTINCT sert à ne pas sélectionner les doublons de la BDD.
  - Dans notre cas, les deux commandes fournissent le même résultat car la table `entity` ne possède pas de doublons.

- Exécuter la requête suivante.

```
1  SELECT id, name, status FROM entity ;
```

  Que voyez-vous ? Quelle est la méthode SQL pour obtenir une projection ?

  - Seules les colonnes `id`, `name` et `status` de la table `entity` ont été sélectionnées.
  - Pour obtenir une projection en SQL, on utilise la commande SELECT suivie des attributs que l'on souhaite conserver.

- Commençons maintenant notre enquête ! Nous allons chercher cette mystérieuse société dont le nom est *Big Data Crunchers Limited*. Il s'agit donc de fabriquer une restriction de la relation **entity**. C'est l'affaire du mot clé WHERE. Exécuter la requête suivante.

```
1  SELECT * FROM entity WHERE name = 'Big Data Crunchers Ltd.' ;
```

  Qu'apprend-on sur la société que l'on recherche ?

  On apprend que cette société est sous la juridiction des Bahamas et active depuis le 2 avril 2007.

  Pour trouver la société *Big Data Crunchers Limited*, nous avons utilisé l'opérateur de comparaison =. D'autres opérateurs de comparaison existent :

  × $A = B$: A est égal à B.
  × $A <> B$: A est différent de B.
  × $A > B$ et $A < B$: A est supérieur/inférieur à B.
  × $A >= B$ et $A =< B$: A est supérieur/inférieur ou égal à B.
  × A BETWEEN A AND C : A est compris entre B et C.
  × A LIKE 'chaine de caractère' : pour comparer A à une chaine de caractère donnée.
  × A IN $(B_1, B_2, \cdots)$: A est présent dans la liste $(B_1, B_2, \cdots)$.
  × A IS NULL : A n'a pas de valeur.

- Les opérateurs logiques OR, AND et NOT signifient respectivement OU, ET et NON. Grâce à ces opérateurs, on peut complexifier un peu nos conditions.
  Exécuter la requête suivante.

```
SELECT * FROM entity
WHERE (id < 10000004 AND (NOT id < 10000000)) OR (name = 'Big Data Crunchers Ltd.') ;
```

  Interpréter.

  On sélectionne dans la table `entity` les lignes dont l'identifiant est strictement inférieur à 10000004 et n'est pas strictement inférieur à 10000000, où dont le nom est *Big Data Crunchers Ltd.*.
  Autrement dit, on sélectionne dans la table `entity`, les lignes dont l'identifiant est compris entre 10000000 et 10000003 ou dont le nom est *Big Data Crunchers Ltd.*

- Le produit cartésien s'obtient facilement grâce à la commande SELECT.

```
SELECT * FROM entity, address ;
```

  Attention, le temps de calcul est parfois long.

- Nous voulons maintenant savoir si *Big Data Crunchers Limited* a servi d'intermédiaire. Les intermédiaires peuvent être soit des personnes physiques, soit des sociétés. Il y a donc peut-être des sociétés qui sont à la fois dans la table intermediary et dans entity.
  Pour utiliser un opérateur binaire (intersection, union, différence) il faut que les tables aient le même schéma, ce qui n'est pas le cas ici.
  Pour avoir la liste des sociétés de entity et de intermediary, on utilise le mot clé UNION. Exécuter la requête suivante.

```
SELECT name, id_address FROM entity
UNION
SELECT name, id_address FROM intermediary ;
```

- Utiliser le mot clé EXCEPT pour trouver les sociétés qui en sont pas des intermédiaires.

```
SELECT name, id_address FROM entity
EXCEPT
SELECT name, id_address FROM intermediary ;
```

- Utiliser enfin le mot clé INTERSECT pour trouver les sociétés qui sont aussi des intermédiaires. Chercher si *Big Data Crunchers Limited* en fait partie.

```
SELECT name, id_address FROM entity
INTERSECT
SELECT name, id_address FROM intermediary ;
```

  Pour savoir si *Big Data Crunchers Ltd.* en fait partie, on peut ajouter la ligne suivante :

```
WHERE name = 'Big Data Crunchers Ltd.' ;
```

  Aucune ligne n'apparaît, cette société n'est donc pas un intermédiaire.

- Nous voulons maintenant trouver l'adresse de la mystérieuse société *Big Data Crunchers Limited*. Pour cela il va falloir faire une *jointure*.
  - Considérons deux tables table1 (de clé étrangère `fk`) et table2 (de clé primaire `pk`). Pour obtenir la jointure de la table table1 avec la table table2 sous la condition `fk` = `pk`, on utilise la syntaxe :

```
SELECT * FROM table1
JOIN table2 ON (table1.fk = table2.pk) ;
```

  Quelle requête permet alors de faire la jointure de la table entity avec la table address sous la condition id_address = id_address ?

```
SELECT * FROM entity
JOIN address ON (entity.id_address = address.id_address) ;
```

- Expliquer pourquoi la requête suivante a le même effet que la jointure.

```
SELECT * FROM entity, address
WHERE entity.id_address = address.id_address ;
```

Cette requête permet d'enchaîner le produit cartésien de `entity` par `address`, puis la restriction aux lignes dont l'attribut `id_address` est identique dans les tables `entity` et `address`. Cela revient bien à faire la jointure de nos 2 tables selon `id_address`.

- Retrouver maintenant l'adresse de la société mystère.

Par lecture dans la table, on obtient que lL'adresse de la société *Big Data Crunchers Ltd.* est le 504 rue de la requête Hesscuhel, Paris, France.

- Retrouvons maintenant les intermédiaires qui ont participé à la création de la société *Big Data Crunchers Limited*.
  Quel est le rôle de la table assoc_inter_entity ?

La table assoc_inter_entity associe chaque intermédiaire aux sociétés avec lesquelles elle a eu une interaction. De manière pratique, chaque ligne contient l'identifiant d'un intermédiaire (idientifiant contenu dans la table intermediary) et l'identifiant d'une société avec laquelle elle a interagi (identifiant contenu dans la table entity). Cette table peut donc présenter plusieurs lignes identiques si une société et un intermédiaire sont entrés en contact plusieurs fois.

- Exécuter et interpréter la requête suivante.

Commentaire

On notera que le mot-clé AS permet de renommer les attributs en quelque chose de plus lisible. De même, les lettres a, e et i dans le FROM sont aussi des alias.

```sql
SELECT
    i.id AS intermediary_id,
    i.name AS intermediary_name,
    e.id AS entity_id,
    e.name AS entity_name,
    e.status AS entity_status
FROM
    intermediary i,
    assoc_inter_entity a,
    entity e
WHERE
    e.name = 'Big Data Crunchers Ltd.'
    AND a.entity = e.id
    AND a.inter = i.id ;
```

La requête précédente effectue le produit cartésien des 3 tables intermediary, assoc_inter_entity et entity. Il sélectionne ensuite :

- dans la table intermediary, les attributs id et name,
- dans la table entity, les attributs id, name et status,

puis extrait seulement les lignes dont :

- le nom de la société est *Big Data Crunchers Ltd.*
- ET l'identifiant de société est le même dans la table assoc_inter_entity et dans la table entity
- ET l'identifiant d'intermédiaire est le même dans la table assoc_inter_entity et dans la table intermediary

Il sélectionne donc seulement les intermédiaires de la société *Big Data Crunchers Ltd.*

▶ Comment obtenir le résultat précédent à l'aide de jointures ?

```sql
SELECT e.id, e.name, e.status, i.id, i.name
FROM entity e
JOIN assoc_inter_entity a ON (a.entity = e.id)
JOIN intermediary i ON (a.inter = i.id)
WHERE e.name = 'Big Data Crunchers Ltd.' ;
```

▶ Conclure sur les intermédiaires qui ont servi à la création de *Big Data Crunchers Limited.*

Les intermédiaires ayant servi à la création de *Big Data Crunchers Limited* sont :

× *Pacher Banking S.A.*,

× *Plouf Financial Services Corp.*

- Nous voulons maintenant incriminer les sociétés qui ont bénéficié des services des deux intermédiaires que nous avons trouvés, dans chacune des juridictions. Nous devons donc faire des agrégations.
  Exécuter la requête suivante.

```
1  SELECT status, COUNT(*) FROM entity GROUP BY status ;
```

  Ici nous avons placé l'attribut de partitionnement status derrière le mot-clé GROUP BY et la fonction d'agrégation COUNT() dans le SELECT. Que renvoie cette requête ?

  Cette commande renvoie un tableau à 2 colonnes.
  - La 1ère colonne contient toutes les différentes valeurs présentes dans la colonne status de la table entity.
  - La 2ème colonne liste l'effectif de chacune de ces valeurs.

- Exécuter et interpréter la ligne suivante.

```
1  SELECT MAX(incorporation_date) AS maxi FROM entity ;
```

  La fonction max est une autre fonction d'agrégation.

  La ligne précédente renvoie la plus grande valeur de la colonne incorporation_date de la table entity.

- Exécuter et interpréter la requête suivante.

```
1  SELECT
2      i.id AS intermediary_id,
3      i.name AS intermediary_name,
4      e.jurisdiction,
5      COUNT(*)
6  FROM intermediary i,
7  JOIN assoc_inter_entity a ON (a.inter = i.id)
8  JOIN entity e ON (a.entity = e.id)
9  WHERE
10     (i.id = 5000 OR i.id = 5001)
11 GROUP BY
12     i.id, i.name, e.jurisdiction ;
```

  La requête précédente :
  1. joint la table assoc_inter_entity (sous l'alias a) à la table intermediary (sous l'alias i) sous la condition `a.inter = i.id`,
  2. adjoint à cette prémière jointure la table entity (sous l'alias e) sous la condition `a.entity = e.id`,
  3. ne sélectionne dans la table ainsi créée que les attributs i.id (renommé en intermediary_id), i.name (renommé en intermediary_name) et e.jurisdiction,
  4. ne sélectionne parmi la projection obtenue que les lignes dont l'attribut i.id est égal à 5000 ou 5001 (identifiants des 2 intermédiaires de la société étudiée)
  5. affiche les colonnes des 3 attributs sélectionnés et une 4ème colonne contenant l'effectif de des différents triplets d'attributs possibles.

- Comment obtenir le résultat précédent à l'aide d'un produit cartésien ?

```sql
SELECT
    i.id AS intermediary_id,
    i.name AS intermediary_name,
    e.jurisdiction,
    count(*)
FROM
    intermediary i,
    assoc_inter_entity a,
    entity e
WHERE
    (i.id = 5000 OR i.id = 5001)
    AND a.inter = i.id
    AND a.entity = e.id
GROUP BY
    i.id, i.name, e.jurisdiction ;
```

- Quelle requête permet d'obtenir la table des noms des juridictions existantes, et du nombre de sociétés enregistrées dans chacune d'entre elles ?

```sql
SELECT
    e.jurisdiction_description AS jurisdiction,
    SUM(e.id)
FROM entity e
GROUP BY jurisdiction ;
```

- Par ailleurs, le mot-clé HAVING permet d'imposer des conditions sur les groupes à qui on applique la fonction d'agrégation. Sachant cela, que fait la requête suivante ?

```sql
SELECT
    e.jurisdiction_description AS jurisdiction,
    SUM(e.id) AS somme
FROM entity e
GROUP BY jurisdiction HAVING (somme > 10000000000);
```

Cette requête permet :

- × de sélectionner l'attribut `jurisdiction_description` de la table `entity` (en le renommant sous l'alias `jurisdiction`),
- × d'ajouter l'attribut d'alias `somme` comptant, pour chaque juridiction, le nomdre d'entreprise en dépendant. Pour cela on utilise la fonction d'agrégation SUM (et le mot-clé GROUP BY),
- × de ne conserver que les lignes pour lesquelles ce nombre d'entreprises est supérieur strict à 10000000000 (avec le mot-clé HAVING).

VII. Filtrage des résultats

À la toute fin d'une requête, il est possible de trier les résultats et de n'en renvoyer qu'une partie.

- Pour trier les lignes d'une table selon le critère que l'on choisit, on utilise le mot clé ORDER BY. Que constatez-vous lorsque vous exécutez la requête ci-dessous ?

```
1  SELECT * FROM entity ORDER BY lifetime DESC ;
```

La table obtenue après exécution de cette requête est triée selon l'attribut lifetime. Plus précisément, les lignes sont classées par valeurs décroissantes de l'attribut lifetime.
Pour obtenir un classement selon des valeurs croissantes, on écrirait :

```
1  SELECT * FROM entity ORDER BY lifetime ASC ;
```

- Exécuter la requête ci-dessous. Interpréter.

```
1  SELECT
2      i.id AS intermediary_id,
3      i.name AS intermediary_name,
4      e.jurisdiction,
5      e.jurisdiction_description,
6      COUNT(*) AS cnt
7  FROM intermediary i,
8  JOIN assoc_inter_entity a ON (a.inter = i.id)
9  JOIN entity e ON a.entity = e.id)
10 WHERE
11     (i.id = 5000 OR i.id = 5001)
12 GROUP BY
13     i.id, i.name, e.jurisdiction, e.jurisdiction_description
14 ORDER BY
15     cnt DESC ;
```

Cette requête renvoie le nombre d'interaction des intermédiaires *Pacher Banking S.A.* et *Plouf Financial Services Corp.* dans chaque juridiction où elle a opéré.

- Que permet de faire la modification suivante par rapport à la requête précédente ?

```
15      cnt DESC LIMIT 5 ;
```

Cette requête permet d'obtenir la même table que la requête précédente mais où l'on a seulement gardé les 5 premières lignes.

- Même question pour la modification suivante.

```
15      cnt DESC LIMIT 5 OFFSET 5 ;
```

Cette requête permet d'obtenir la même table que la requête précédente mais où l'on a seulement gardé les 5 dernières lignes.

VIII. Conclusion : les commandes à retenir

- La commande CREATE TABLE.
- Les mots clés PRIMARY KEY et FOREIGN KEY avec la méthode pour pointer vers une clé candidate d'une autre table.
- Les projections, restrictions et produits cartésiens avec SELECT ... FROM ... WHERE ...
- Pour la restriction, les opérateurs de comparaison.
- Les opérateurs logiques OR, AND et NOT.
- Le mot clé DISTINCT.
- Les opérations binaires sur les tables données par les mots clé UNION, INTERSECT, EXCEPT.
- La méthode de jointure avec l'opérateur JOIN ... ON ...
- Les mots-clés GROUP BY ($\cdots$ HAVING $\cdots$) pour faire un agrégat.
- Les mots-clés de filtrage des résultats :
  - × ORDER BY `attribut` ASC / DESC pour trier suivant l'`attribut` choisi par ordre croissant / décroissant,
  - × LIMIT n pour limiter la sorties à n lignes,
  - × OFFSET n pour débuter à partir de la $n^{\text{ème}}$ ligne.
- Les fonctions d'agrégation :
  - × COUNT(*) : compter le nombre d'occurence d'une valeur d'un attribut ou d'un groupement d'attributs,
  - × MIN : trouver le minimum des valeurs d'un attribut,
  - × MAX : trouver le maximum des valeurs d'un attribut,
  - × AVG : effectuer la moyenne des valeurs d'un attribut,
  - × SUM : effectuer la somme des valeurs d'un attribut.