NSI (Numérique et Sciences Informatiques)

Classe : Terminale

Séquence 1

Mr BANANKO K.

LYCÉE INTERNATIONAL COURS LUMIÈRE

COURS

BASE DE DONNEES

Chap. 1

Contenus	Capacités attendues	Commentaires
Modèle relationnel : relation, attribut, domaine, clef primaire, clef étrangère, schéma relationnel.	Identifier les concepts définissant le modèle relationnel.	Ces concepts permettent d'exprimer les contraintes d'intégrité (domaine, relation et référence).
Base de données relationnelle.	Savoir distinguer la structure d'une base de données de son contenu. Repérer des anomalies dans le schéma d'une base de données.	La structure est un ensemble de schémas relationnels qui respecte les contraintes du modèle relationnel. Les anomalies peuvent être des redondances de données ou des anomalies d'insertion, de suppression, de mise à jour. On privilégie la manipulation de données nombreuses et réalistes.
Système de gestion de bases de données relationnelles.	Identifier les services rendus par un système de gestion de bases de données relationnelles : persistance des données, gestion des accès concurrents, efficacité de traitement des requêtes, sécurisation des accès.	Il s'agit de comprendre le rôle et les enjeux des différents services sans en détailler le fonctionnement
Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données.	Identifier les composants d'une requête. Construire des requêtes d'interrogation à l'aide des clauses du langage SQL : SELECT, FROM, WHERE, JOIN. Construire des requêtes d'insertion et de mise à jour à l'aide de : UPDATE, INSERT, DELETE.	On peut utiliser DISTINCT, ORDER BY ou les fonctions d'agrégation sans utiliser les clauses GROUP BY et HAVING.

A- INTRODUCTION

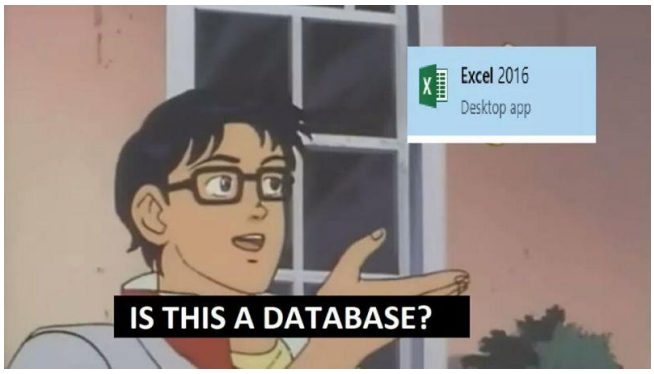
De la nécessité d'organiser le stockage de ses données

Lorsqu'une grande quantité de données doit être gérée, il faut savoir distinguer deux choses :

- La structure qui va sous-tendre l'ensemble de ces données, notamment les liens entre elles, les hiérarchies éventuelles, ...
- Le type de logiciel qui va aider à gérer ces données.

Par exemple, si je souhaite stocker toutes les températures relevées dans mon jardin tous les matins à 07h00, je sais que mes données seront des couples (date, température). Éventuellement ces dates seront regroupées par mois, ou par saison... mais la structure des données sera quand même simple et linéaire. Pour gérer ces données, je peux : les écrire à la main dans mon agenda, créer un feuille de tableur avec Excel ou LibreOffice, utiliser une liste dans un IDE Python,... Chaque méthode aura ses avantages et ses inconvénients.

- Si le nombre de données à stocker devient très grand, est-ce que ma solution choisie pourra les gérer? (on peut par exemple méditer sur le cas du Royaume-Uni dont le comptage des patients positifs au Covid est devenu faux car il a dépassé les limites de leur feuille Excel)



- Est-ce que d'autres personnes que moi sont susceptibles de consulter ou modifier ces données, éventuellement en même temps que moi ?
- Si une donnée se retrouve à plusieurs endroits dans mes données, devrais-je aller modifier cette donnée partout où elle se trouve ou bien une seule fois ?

L'étude des Bases de Données tente d'apporter des réponses à toutes ces questions.

1- Introduction

Le développement des traitements informatiques nécessite la manipulation de données de plus en plus nombreuses. Leur organisation et leur stockage constituent un enjeu essentiel de performance. Le recours aux bases de données relationnelles est aujourd'hui une solution très répandue. Ces bases de données permettent d'organiser, de stocker, de mettre à jour et d'interroger des données structurées volumineuses

utilisées simultanément par différents programmes ou différents utilisateurs. Cela est impossible avec les représentations tabulaires étudiées en classe de première. Des systèmes de gestion de bases de données (SGBD) de très grande taille (de l'ordre du pétaoctet) sont au centre de nombreux dispositifs de collecte, de stockage et de production d'informations. L'accès aux données d'une base de données relationnelle s'effectue grâce à des requêtes d'interrogation et de mise à jour qui peuvent par exemple être rédigées dans le langage SQL (Structured Query Language). Les traitements peuvent conjuguer le recours au langage SQL et à un langage de programmation.

Le terme base de données est apparu au début des années 60. C'est l'apparition des disques durs à la fin des années 50 qui a permis d'utiliser les ordinateurs pour stocker et manipuler des données. Avec l'apparition du Web, la quantité de données à stocker a littéralement explosé. Aujourd'hui, la plupart des sites internet (du petit site personnel au grand site d'e-commerce) utilisent au moins une base de données. Les bases de données jouent un rôle fondamental dans notre monde devenu numérique où il est extrêmement facile de dupliquer l'information. Voilà pourquoi nous allons cette année les étudier.

Activité 1 .1

Trouvez des exemples de domaines d'activité où les bases de données jouent un rôle primordial.

2- Les systèmes de gestion de base de données

Dans une base de données, l'information est stockée dans des fichiers, mais à la différence des fichiers au format CSV, il n'est pas possible de travailler sur ces données avec un simple éditeur de texte. Pour manipuler les données présentes dans une base de données (écrire, lire ou encore modifier), il est nécessaire d'utiliser un type de logiciel appelé "système de gestion de base de données" très souvent abrégé en SGBD. Il existe une multitude de SGBD : des gratuites, des payantes, des libres ou bien encore des propriétaires. Les SGBD permettent de grandement simplifier la gestion des bases de données :

- Les SGBD permettent de gérer la lecture, l'écriture ou la modification des informations contenues dans une base de données
- Les SGBD permettent de gérer les autorisations d'accès à une base de données. Il est en effet souvent nécessaire de contrôler les accès par exemple en permettant à l'utilisateur A de lire et d'écrire dans la base de données alors que l'utilisateur B aura uniquement la possibilité de lire les informations contenues dans cette même base de données.
- Les fichiers des bases de données sont stockés sur des disques durs dans des ordinateurs, ces ordinateurs peuvent subir des pannes. Il est souvent nécessaire que l'accès aux informations contenues dans une base de données soit maintenu, même en cas de panne matérielle. Les bases de données sont donc dupliquées sur plusieurs ordinateurs afin qu'en cas de panne d'un ordinateur A, un ordinateur B contenant une copie de la base de données présente dans A, puisse prendre le relais. Tout cela est très complexe à gérer, en effet toute modification de la base de données présente sur l'ordinateur A doit entrainer la même modification de la base de données présente sur l'ordinateur B. Cette synchronisation entre A et B doit se faire le plus rapidement possible, il est fondamental d'avoir des copies parfaitement identiques en permanence. C'est aussi les SGBD qui assurent la maintenance des différentes copies de la base de données.

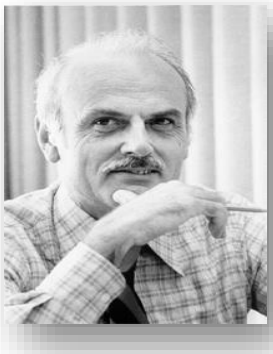
- Plusieurs personnes peuvent avoir besoin d'accéder aux informations contenues dans une base donnée en même temps. Cela peut parfois poser problème, notamment si les 2 personnes désirent modifier la même donnée au même moment (on parle d'accès concurrent). Ces problèmes d'accès concurrent sont aussi gérés par les SGBD.

Comme nous venons de la voir, les SGBD jouent un rôle fondamental. L'utilisation des SGBD explique en partie la supériorité de l'utilisation des bases de données sur des solutions plus simples à mettre en œuvre ; mais aussi beaucoup plus limitées comme les fichiers au format CSV.

B- BASE DE DONNEES RELATIONNELLE

1- Introduction

Il existe différents types de bases de données, par exemple, les bases de données hiérarchiques, les bases de données objet, les bases de données nosql ou bien encore les bases de données relationnelles. Les bases de données relationnelles sont le plus utilisées au monde, c'est ce type de base de données que nous allons étudier.

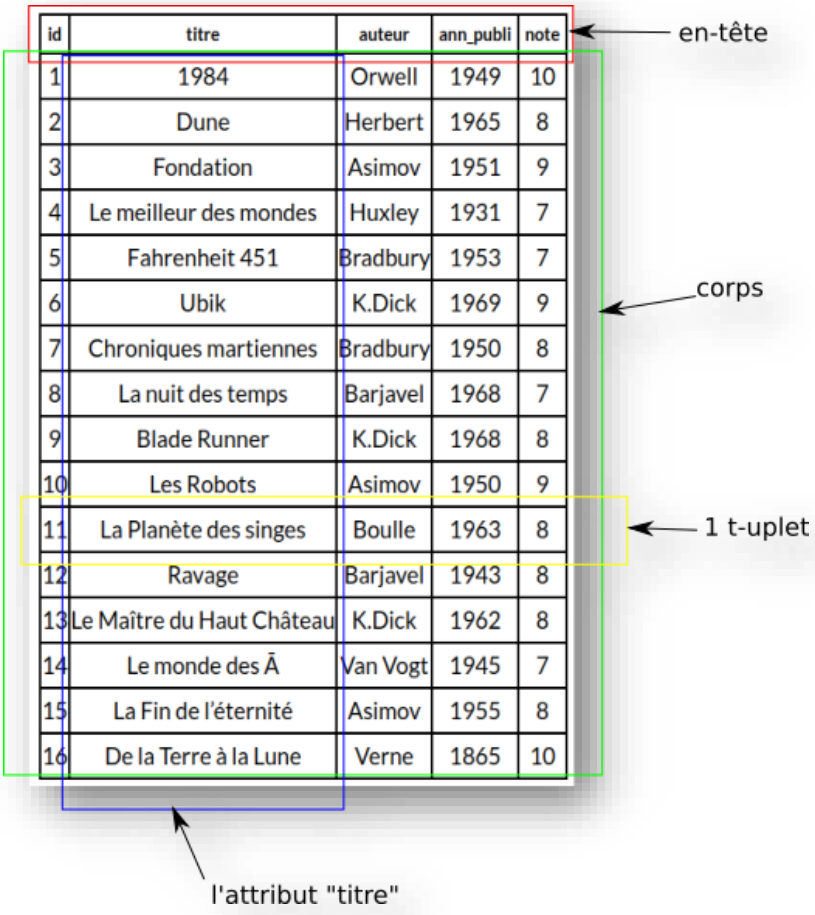


Les bases de données relationnelles ont été mises au point en 1970 par Edgar Franck Codd, informaticien britannique (1923-2003). Ces bases de données sont basées sur la théorie mathématique des ensembles.

2- Relation

La notion de relation est au cœur des bases de données relationnelles. Une relation peut être vue comme un tableau à 2 dimensions, composé d'un en-tête et d'un corps. Le corps est lui-même composé de t-uplets (lignes) et d'attributs (colonnes). L'en-tête contient les intitulés des attributs, le corps contient les données proprement dites. À noter que l'on emploie aussi le terme "**table**" à la place de "**relation**".

Voici un exemple de relation :



id	titre	auteur	ann_publi	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
3	Fondation	Asimov	1951	9
4	Le meilleur des mondes	Huxley	1931	7
5	Fahrenheit 451	Bradbury	1953	7
6	Ubik	K.Dick	1969	9
7	Chroniques martiennes	Bradbury	1950	8
8	La nuit des temps	Barjavel	1968	7
9	Blade Runner	K.Dick	1968	8
10	Les Robots	Asimov	1950	9
11	La Planète des singes	Boulle	1963	8
12	Ravage	Barjavel	1943	8
13	Le Maître du Haut Château	K.Dick	1962	8
14	Le monde des Ā	Van Vogt	1945	7
15	La Fin de l'éternité	Asimov	1955	8
16	De la Terre à la Lune	Verne	1865	10

Relation "LIVRES"

Le t-uplet encadré en jaune sur le schéma ci-dessus contient les éléments suivant : **11, La Planète des singes, Boulle, 1963 et 8.**

L'attribut "**titre**" est composé des éléments suivants : 1984, Dune, Fondation, Le meilleur des mondes, Fahrenheit 451, Ubik, Chroniques martiennes, La nuit des temps, Blade Runner, Les Robots, La Planète des singes, Ravage, Le Maître du Haut Château, Le monde des Ā, La Fin de l'éternité et De la Terre à la Lune.

Activité 1.2

Faites la liste des éléments appartenant à l'attribut "**auteur**".

3- Domaine

Pour chaque attribut d'une relation, il est nécessaire de définir un domaine : Le domaine d'un attribut donné correspond à un ensemble fini ou infini de valeurs admissibles.

Par exemple, le domaine de l'attribut "**id**" correspond à l'ensemble des entiers (noté INT) : la colonne "**id**" devra obligatoirement contenir des entiers.

Autre exemple, le domaine de l'attribut "**titre**" correspond à l'ensemble des chaînes de caractères (noté TEXT). Dernier exemple, le domaine de l'attribut "**note**" correspond à l'ensemble des entiers positifs.

Activité 1.3

Quel est, selon vous, le domaine de l'attribut "**auteur**"

Au moment de la création d'une relation, il est nécessaire de renseigner le domaine de chaque attribut. Le SGBD s'assure qu'un élément ajouté à une relation respecte bien le domaine de l'attribut correspondant : si par exemple vous essayez d'ajouter une note non entière (par exemple 8.5), le SGBD signalera cette erreur et n'autorisera pas l'écriture de cette nouvelle donnée.

4- Clé primaire

Autre contrainte très importante dans les bases de données relationnelles, une relation ne peut pas contenir 2 t-uplets identiques. Par exemple, la situation ci-dessous n'est pas autorisée (ici aussi c'est le SGBD qui veille au grain) :

id	titre	auteur	ann_publi	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
2	Dune	Herbert	1965	8
3	Fondation	Asimov	1951	9

Afin d'être sûr de respecter cette contrainte des t-uplets identiques, on définit la notion de "**clef primaire**".

Une clef primaire est un attribut dont la valeur permet d'identifier de manière unique un t-uplet de la relation.

Autrement dit, si un attribut est considéré comme clef primaire, on ne doit pas trouver dans toute la relation 2 fois la même valeur pour cet attribut.

Si on se réfère à l'exemple de la relation ci-dessous :

Relation LIVRES

id	titre	auteur	ann_publi	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
3	Fondation	Asimov	1951	9
4	Le meilleur des mondes	Huxley	1931	7
5	Fahrenheit 451	Bradbury	1953	7
6	Ubik	K.Dick	1969	9
7	Chroniques martiennes	Bradbury	1950	8
8	La nuit des temps	Barjavel	1968	7
9	Blade Runner	K.Dick	1968	8
10	Les Robots	Asimov	1950	9
11	La Planète des singes	Boulle	1963	8
12	Ravage	Barjavel	1943	8
13	Le Maître du Haut Château	K.Dick	1962	8
14	Le monde des Ā	Van Vogt	1945	7
15	La Fin de l'éternité	Asimov	1955	8
16	De la Terre à la Lune	Verne	1865	10

L'attribut "**note**" peut-il jouer le rôle de clef primaire ? **Non**, car il est possible de trouver 2 fois la même note.

L'attribut "**ann_publi**" peut-il jouer le rôle de clef primaire ? **Non**, car il est possible de trouver 2 fois la même année.

L'attribut "**auteur**" peut-il jouer le rôle de clef primaire ? **Non**, car il est possible de trouver 2 fois le même auteur.

L'attribut "**titre**" peut-il jouer le rôle de clef primaire ? **A priori oui**, car l'attribut "titre" ne comporte pas 2 fois le même titre de roman. Mais, ce n'est pas forcément une bonne idée, car il est tout à fait possible d'avoir un même titre pour 2 romans différents. Par exemple, en 2013, l'Américaine Jill McCorkle et l'Anglaise Kate Atkison publiaient avec seulement six jours d'écart un livre intitulé "**Life After Life**" !

Il nous reste donc l'attribut "**id**". En fait, l'attribut "**id**" ("**id**" comme "**identifiant**") a été placé là pour jouer le rôle de clef primaire. En effet, à chaque fois qu'un roman est ajouté à la relation, sont "**id**" correspond

à l'incrémentation de l'id (id du nouveau=id de l'ancien+1) du roman précédemment ajouté. Il est donc impossible d'avoir deux romans avec le même id. Ajouter un attribut "**id**" afin qu'il puisse jouer le rôle de clef primaire est une pratique courante (mais non obligatoire) dans les bases de données relationnelles. Dans le cas précis qui nous intéresse, il aurait été possible de ne pas utiliser d'attribut "**id**", car chaque livre édité possède un numéro qui lui est propre : l'ISBN, cet ISBN aurait donc pu jouer le rôle de clef primaire.

À noter qu'en toute rigueur, une clef primaire peut être constituée de plusieurs attributs, par exemple le couple "**auteur**" + "**titre**" pourrait jouer le rôle de clé primaire (à moins qu'un auteur écrive 2 romans différents, mais portant tous les deux le même titre), mais nous n'étudierons pas cet aspect des choses ici.

Activité 1.4

Voici un extrait d'une relation référençant des films :

Relation FILMS

id	titre	realisateur	ann_sortie	note_sur_10
1	Alien, le huitième passager	Scott	1979	10
2	Dune	Lynch	1985	5
3	2001 : l'odyssée de l'espace	Kubrick	1968	9
4	Blade Runner	Scott	1982	10

.....

Listez les différents attributs de cette relation. Donnez le domaine de chaque attribut. Pour chaque attribut dire si cet attribut peut jouer le rôle de clef primaire, vous n'oublierez pas de justifier vos réponses.

5- Clé étrangère

a- Duplication des données

Revenons à notre relation "**LIVRES**". Nous désirons maintenant un peu enrichir cette relation en ajoutant des informations supplémentaires sur les auteurs, nous obtenons alors :

Relation LIVRES_AUTEURS

id	titre	nom_auteur	prenom_auteur	date_nai_auteur	langue_ecriture_auteur	ann_publi	note
1	1984	Orwell	George	1903	anglais	1949	10
2	Dune	Herbert	Frank	1920	anglais	1965	8
3	Fondation	Asimov	Isaac	1920	anglais	1951	9
4	Le meilleur des mondes	Huxley	Aldous	1894	anglais	1931	7
5	Fahrenheit 451	Bradbury	Ray	1920	anglais	1953	7
6	Ubik	K.Dick	Philip	1928	anglais	1969	9
7	Chroniques martiennes	Bradbury	Ray	1920	anglais	1950	8
8	La nuit des temps	Barjavel	René	1911	français	1968	7
9	Blade Runner	K.Dick	Philip	1928	anglais	1968	8
10	Les Robots	Asimov	Isaac	1920	anglais	1950	9
11	La Planète des singes	Boulle	Pierre	1912	français	1963	8
12	Ravage	Barjavel	René	1911	français	1943	8
13	Le Maître du Haut Château	K.Dick	Philip	1928	anglais	1962	8
14	Le monde des Ā	Van Vogt	Alfred Elton	1912	anglais	1945	7
15	La Fin de l'éternité	Asimov	Isaac	1920	anglais	1955	8
16	De la Terre à la Lune	Verne	Jules	1828	français	1865	10

Nous avons ajouté 3 attributs ("**prenom_auteur**", "**date_nai_auteur**" et "**langue_ecriture_auteur**"). Nous avons aussi renommé l'attribut "auteur" en "**nom_auteur**".

Comme vous l'avez peut-être remarqué, il y a pas mal d'informations dupliquées, par exemple, on retrouve 3 fois "**K.Dick Philip 1928 anglais**", même chose pour "**Asimov Isaac 1920 anglais**"...Cette duplication est-elle indispensable ? Non ! Est-elle souhaitable ? Non plus ! En effet, dans une base de données, on évite autant que possible de dupliquer l'information (sauf à des fins de sauvegarde, mais ici c'est toute autre chose). Si nous dupliquons autant de données inutilement c'est que notre structure ne doit pas être la bonne ! Mais alors, comment faire pour avoir aussi des informations sur les auteurs des livres ?

b- Notion de clé étrangère

La solution est relativement simple : travailler avec 2 relations au lieu d'une seule et créer un "**lien**" entre ces 2 relations :

Une clé étrangère est une clé primaire d'une autre relation.

Relation AUTEURS

id	nom	prenom	ann_naissance	langue_ecriture
1	Orwell	George	1903	anglais
2	Herbert	Frank	1920	anglais
3	Asimov	Isaac	1920	anglais
4	Huxley	Aldous	1894	anglais
5	Bradbury	Ray	1920	anglais
6	K.Dick	Philip	1928	anglais
7	Barjavel	René	1911	français
8	Boulle	Pierre	1912	français
9	Van Vogt	Alfred Elton	1912	anglais
10	Verne	Jules	1828	français

Relation LIVRES

id	titre	id_auteur	ann_publi	note
1	1984	1	1949	10
2	Dune	2	1965	8

3	Fondation	3	1951	9
4	Le meilleur des mondes	4	1931	7
5	Fahrenheit 451	5	1953	7
6	Ubik	6	1969	9
7	Chroniques martiennes	5	1950	8
8	La nuit des temps	7	1968	7
9	Blade Runner	6	1968	8
10	Les Robots	3	1950	9
11	La Planète des singes	8	1963	8
12	Ravage	7	1943	8
13	Le Maître du Haut Château	6	1962	8
14	Le monde des Ā	9	1945	7
15	La Fin de l'éternité	3	1955	8
16	De la Terre à la Lune	10	1865	10

Nous avons créé une relation **AUTEURS** et nous avons modifié la relation **LIVRES** : nous avons remplacé l'attribut "**auteur**" par un attribut "**id_auteur**".

Comme vous l'avez sans doute remarqué, l'attribut "**id_auteur**" de la relation **LIVRES** permet de créer un lien avec la relation **AUTEURS**. "**id_auteur**" correspond à l'attribut "**id**" de la relation **AUTEURS**. L'introduction d'une relation AUTEURS et la mise en place de liens entre cette relation et la relation **LIVRES** permettent d'éviter la redondance d'informations.

Pour établir un lien entre 2 relations R_A et R_B, on ajoute à R_A un attribut x qui prendra les valeurs de la clé primaire de R_B. Cet attribut x est appelé clef étrangère (l'attribut correspond à la clé primaire d'une autre table, d'où le nom).

Dans l'exemple ci-dessus, l'attribut "**id_auteur**" de la relation LIVRES permet bien d'établir un lien entre la relation LIVRES et la relation AUTEURS, "**id_auteur**" correspond bien à la clef primaire de la relation **AUTEURS**, conclusion : "**id_auteur**" est une clef étrangère.

Pour préserver l'intégrité d'une base de données, il est important de bien vérifier que toutes les valeurs de la clef étrangère correspondent bien à des valeurs présentes dans la clef primaire (nous aurions un problème d'intégrité de la base de données si une valeur de l'attribut "**id_auteur**" de la relation **LIVRES** ne

correspondait à aucune valeur de la clef primaire de la relation **AUTEURS**). Certains SGBD ne vérifient pas cette contrainte (ne renvoie aucune erreur en cas de problème), ce qui peut provoquer des comportements erratiques.

Activité 1.5

En partant de la relation FILMS ci-dessous, créez une relation REALISATEURS (attributs de la relation REALISATEURS : id, nom, prenom et ann_naissance, vous trouverez toutes les informations nécessaires sur le Web).

Modifiez ensuite la relation FILMS afin d'établir un lien entre les relations FILMS et REALISATEURS. Vous préciserez l'attribut qui jouera le rôle de clef étrangère.

Relation FILMS

id	titre	realisateur	ann_sortie	note_sur_10
1	Alien, le huitième passager	Scott	1979	10
2	Dune	Lynch	1985	5
3	2001 : l'odyssée de l'espace	Kubrick	1968	9
4	Blade Runner	Scott	1982	10

6- SCHEMA RELATIONNEL

Dernière définition, on appelle schéma relationnel l'ensemble des relations présentes dans une base de données. Quand on vous demande le schéma relationnel d'une base de données, il est nécessaire de fournir les informations suivantes :

- Les noms des différentes relations
- Pour chaque relation, la liste des attributs avec leur domaine respectif
- Pour chaque relation, la clef primaire et éventuellement la clef étrangère

Voici un exemple pour les relations LIVRES et AUTEURS :

AUTEURS(id : INT, nom : TEXT, prenom : TEXT, ann_naissance : INT, langue_ecriture : TEXT)

LIVRES(id : INT, titre : TEXT, #id_auteur : INT, ann_publi : INT, note : INT)

Les attributs soulignés sont des clefs primaires, le # signifie que l'on a une clef étrangère.

Les clés étrangères, lorsqu'elles existent, peuvent être signalées par une astérisque * ou un dièse #.

Activité 1.6

Donnez le schéma relationnel de la base de données que vous avez défini dans le "**Activité 1.4**"

EXERCICE

On considérera une base de données constituée de la relation PRODUITS :

Relation PRODUITS

ref	**nom**	**prix_unitaire**	**fournisseur**
1278	X-2212	64	AMC-V
1580	Y-32	56	YAMEL
1665	BN6	57	ABS-united
1447	Z32	48	AMC-V
2568	Y-67	90	YAMEL
3558	OIP-78	90	ABS-united
2222	BN6	60	ABS-united

1. Expliquez en quelques lignes ce qu'est un SGBD (système de gestion de base de données).
2. Expliquez en quelques lignes l'avantage d'une base de données par rapport au stockage des données dans un fichier de type CSV
3. Donnez le nom d'un des attributs de la relation PRODUITS
4. Donnez un exemple de t-uplet de la relation PRODUITS
5. Parmi tous les attributs de la relation PRODUITS, déterminez-le(s) attribut(s) qui peut (peuvent) jouer le rôle de clé primaire ? (Justifiez votre réponse)

C- LANGAGE SQL

1- Introduction

Nous avons eu l'occasion d'étudier la structure d'une base de données relationnelle, nous allons maintenant apprendre à réaliser des requêtes, c'est-à-dire que nous allons apprendre à créer une base des données, créer des attributs, ajouter de données, modifier des données et enfin, nous allons surtout apprendre à interroger une base de données afin d'obtenir des informations.

Pour réaliser toutes ces requêtes, nous allons devoir apprendre un langage de requêtes : **SQL** (*Structured Query Language*)*(Langage de requêtes structurées)*. **SQL** est propre aux bases de données relationnelles, les autres types de bases de données utilisent d'autres langages pour effectuer des requêtes.

Les SGBD les plus utilisés sont basés sur le modèle relationnel. Parmi eux, citons Oracle, MySQL, Microsoft SQL Server, PostgreSQL, Microsoft Access, SQLite, MariaDB...

Mais de plus en plus de SGBD **non-relationnels** sont utilisés, spécialement adaptés à des données plus diverses et moins structurées. On les retrouve sous l'appelation **NoSQL** (pour *Not only SQL*). Citons parmi eux MongoDB, Cassandra (Facebook), BigTable (Google)...

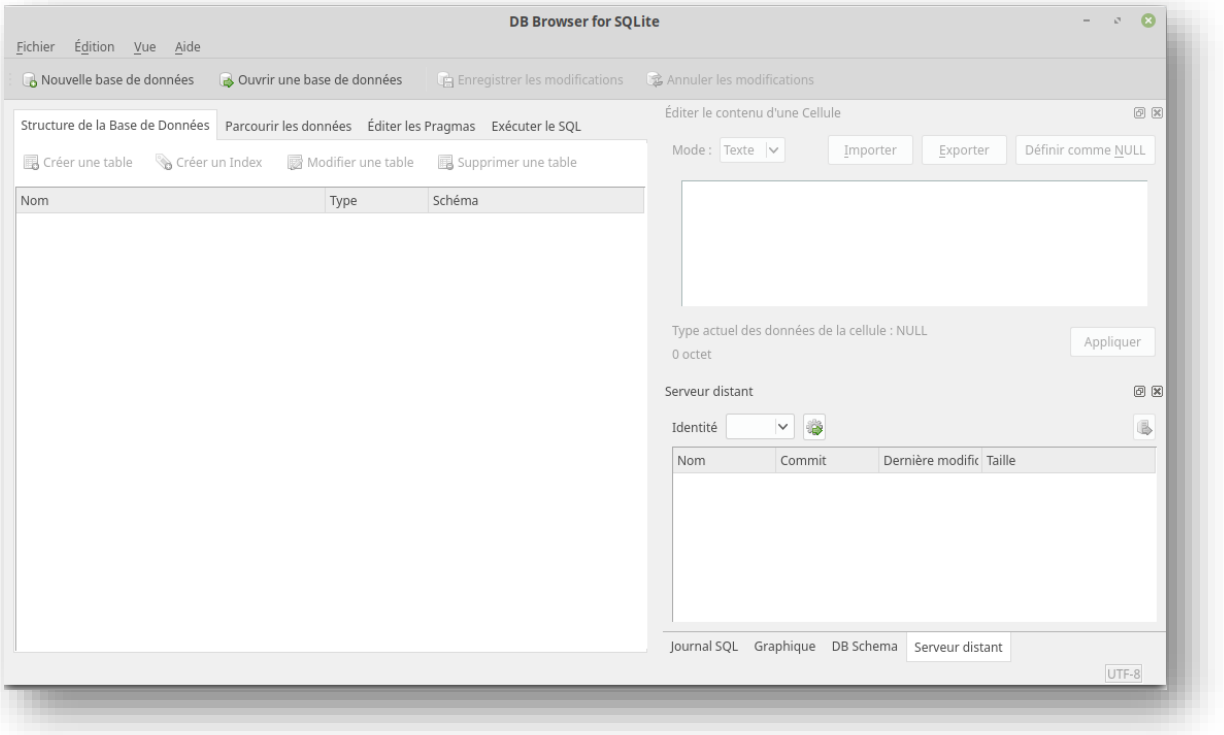
Nous allons travailler principalement avec le langage SQLite peut lui s'utiliser directement sans démarrer un serveur : la base de données est entièrement représentée dans le logiciel utilisant SQLite (dans notre cas, **DB Browser for SQLite**(https://sqlitebrowser.org/.)).

Sa simplicité d'utilisation en fera notre choix pour illustrer cette présentation du langage SQL.

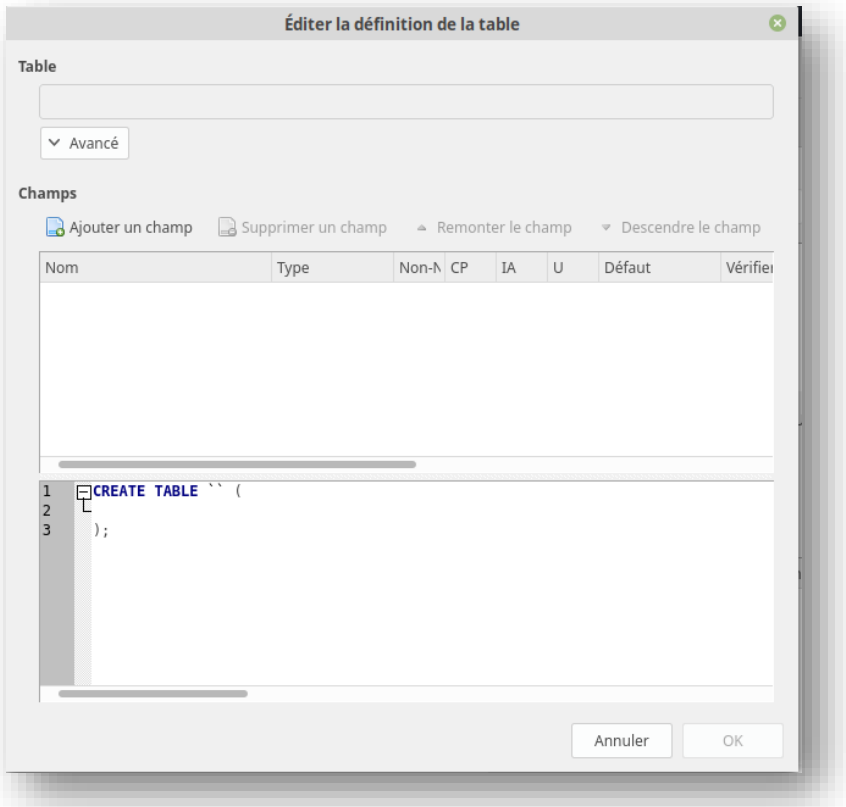
Nous allons commencer par créer notre base de données :

Activité 1.7

Après avoir lancé le logiciel "DB Browser for SQLite", vous devriez obtenir ceci :

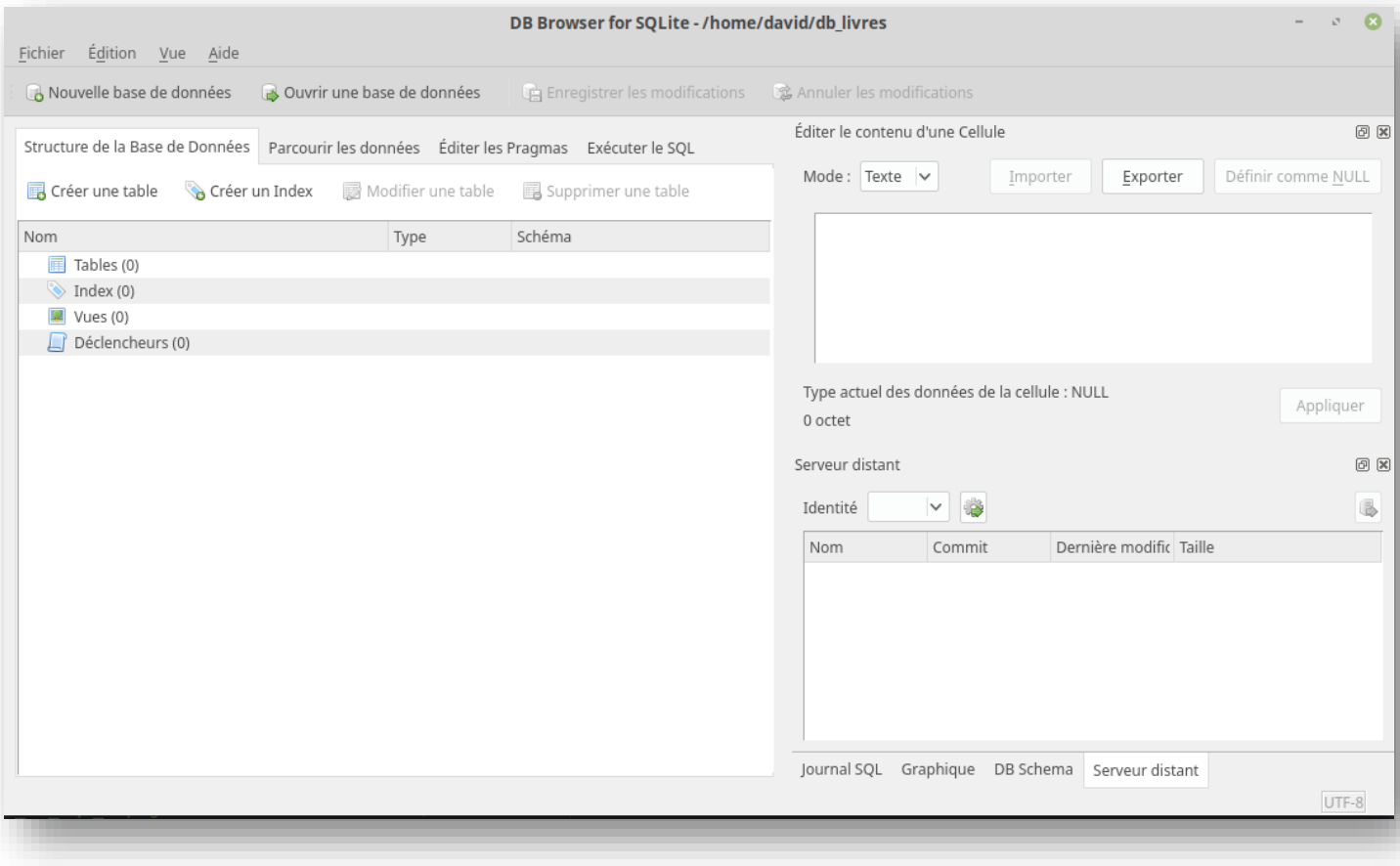


Cliquez sur Nouvelle base de données. Après avoir choisi un nom pour votre base de données (par exemple "**db_livres.db**"), vous devriez avoir la fenêtre suivante :

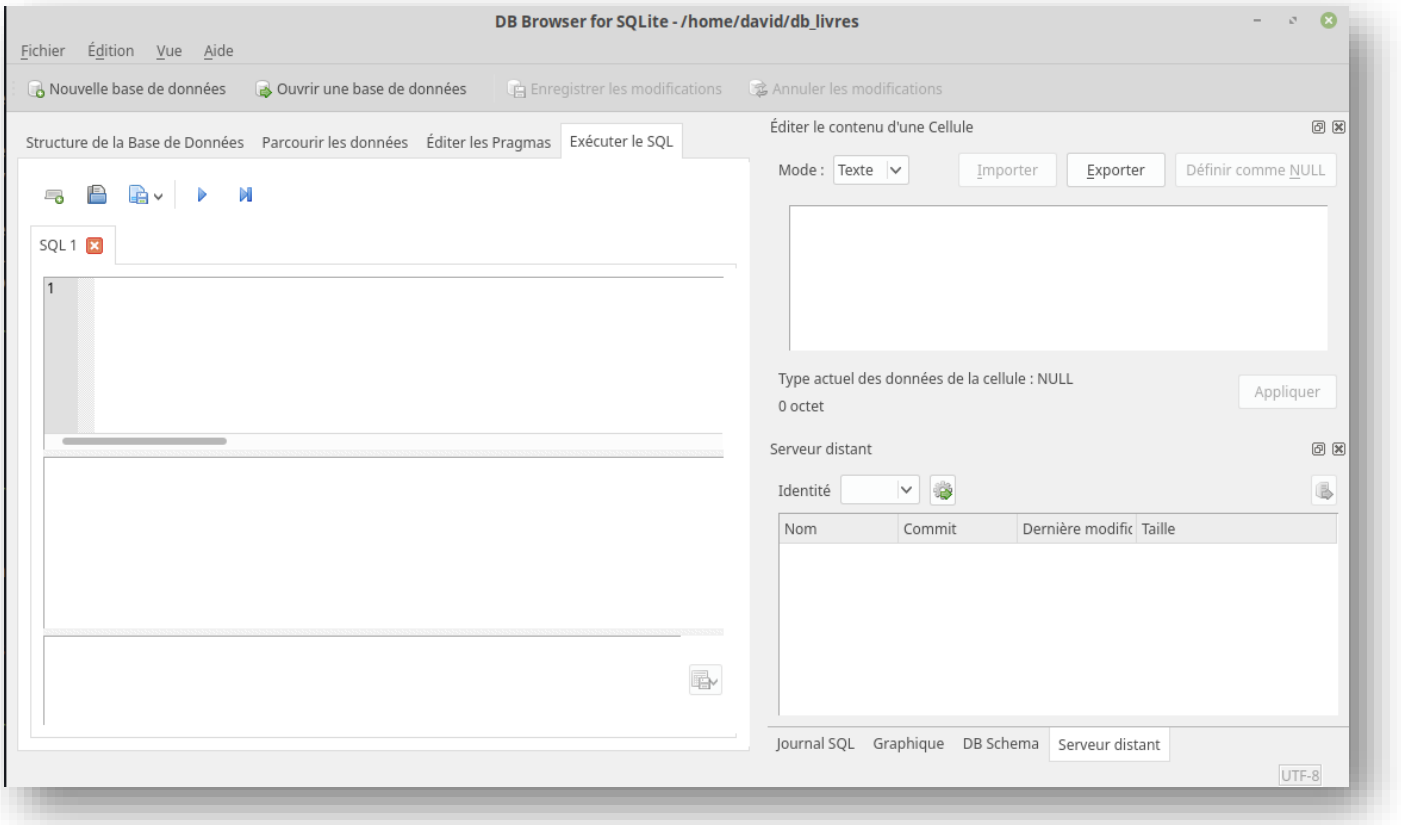


Cliquez alors sur Annuler

Notre base de données a été créée :



mais pour l'instant elle ne contient aucune table (aucune relation), pour créer une table, cliquez sur l'onglet "**Exécuter le SQL**". On obtient alors :

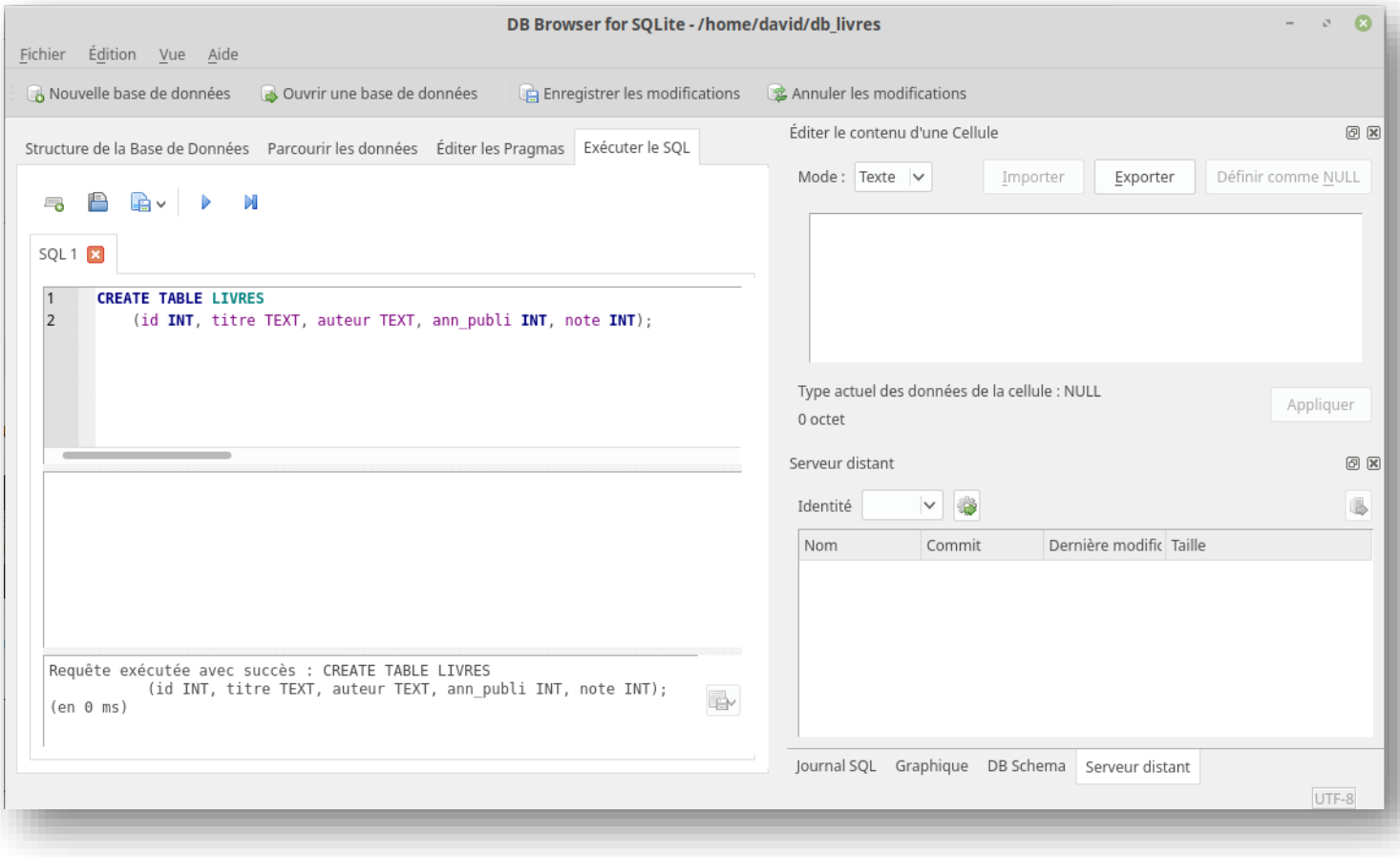


➢ Création de données

Copiez-collez le texte ci-dessous dans la fenêtre "**SQL 1**"

```
CREATE TABLE LIVRES

(id INT, titre TEXT, auteur TEXT, ann_publi INT, note INT);
```

Cliquez ensuite sur le petit triangle situé au-dessus de la fenêtre SQL 1 (ou appuyez sur F5), vous devriez avoir ceci :



Comme indiqué dans la fenêtre, "**Requête exécutée avec succès**" !

Vous venez de créer votre première table.

Revenons sur cette première requête :

Le *CREATE TABLE LIVRES* ne devrait pas vous poser de problème : nous créons une nouvelle table nommée "LIVRES".

La suite est à peine plus complexe :

nous créons ensuite les attributs :

- id
- titre
- auteur
- ann_pulbi
- note

Nous avons pour chaque attribut précisé son domaine : id : entier (INT), titre : chaîne de caractères (TEXT), auteur : chaîne de caractères, ann_publi : entier et note : entier

L'attribut "**id**" va jouer ici le rôle de clé primaire. On peut aussi, par souci de sécurité (afin d'éviter que l'on utilise 2 fois la même valeur pour l'attribut "**id**"), modifier le l'instruction SQL vue ci-dessus, afin de préciser que l'attribut "**id**" est bien notre clé primaire :

```
CREATE TABLE LIVRES

(id INT, titre TEXT, auteur TEXT, ann_publi INT, note INT, PRIMARY KEY (id));
```

Notre système de gestion de base de données nous avertira si l'on tente d'attribuer 2 fois la même valeur à l'attribut "**id**".

Remarque :
C'est l'utilisateur qui spécifie, éventuellement, quel attribut sera une clé primaire.

Nous allons maintenant ajouter des données :

➢ Insertion de donnés

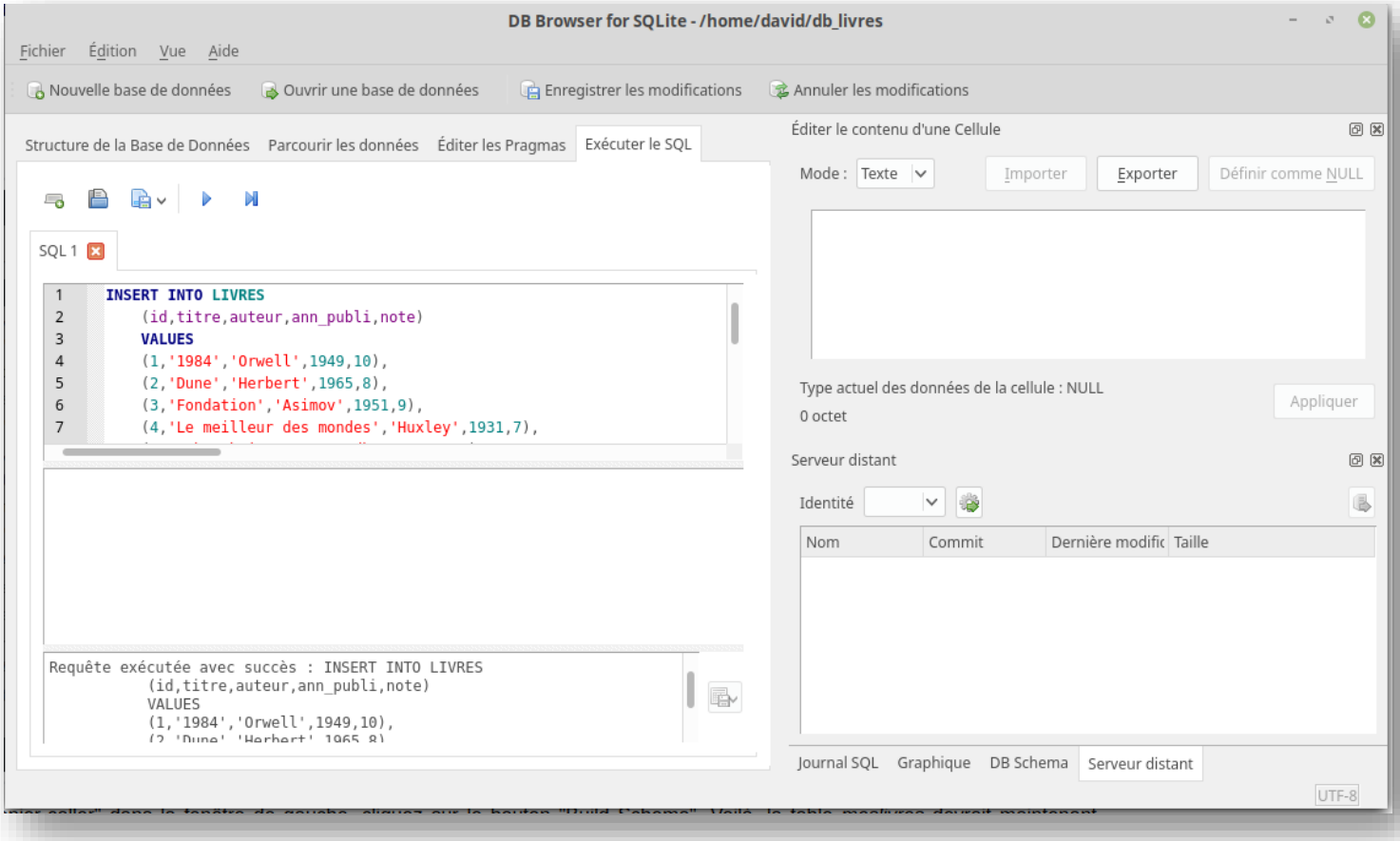
Activité 1.7

Toujours dans l'onglet "**Exécuter le SQL**", après avoir effacé la fenêtre SQL 1, copiez-collez dans cette même fenêtre la requête ci-dessous :

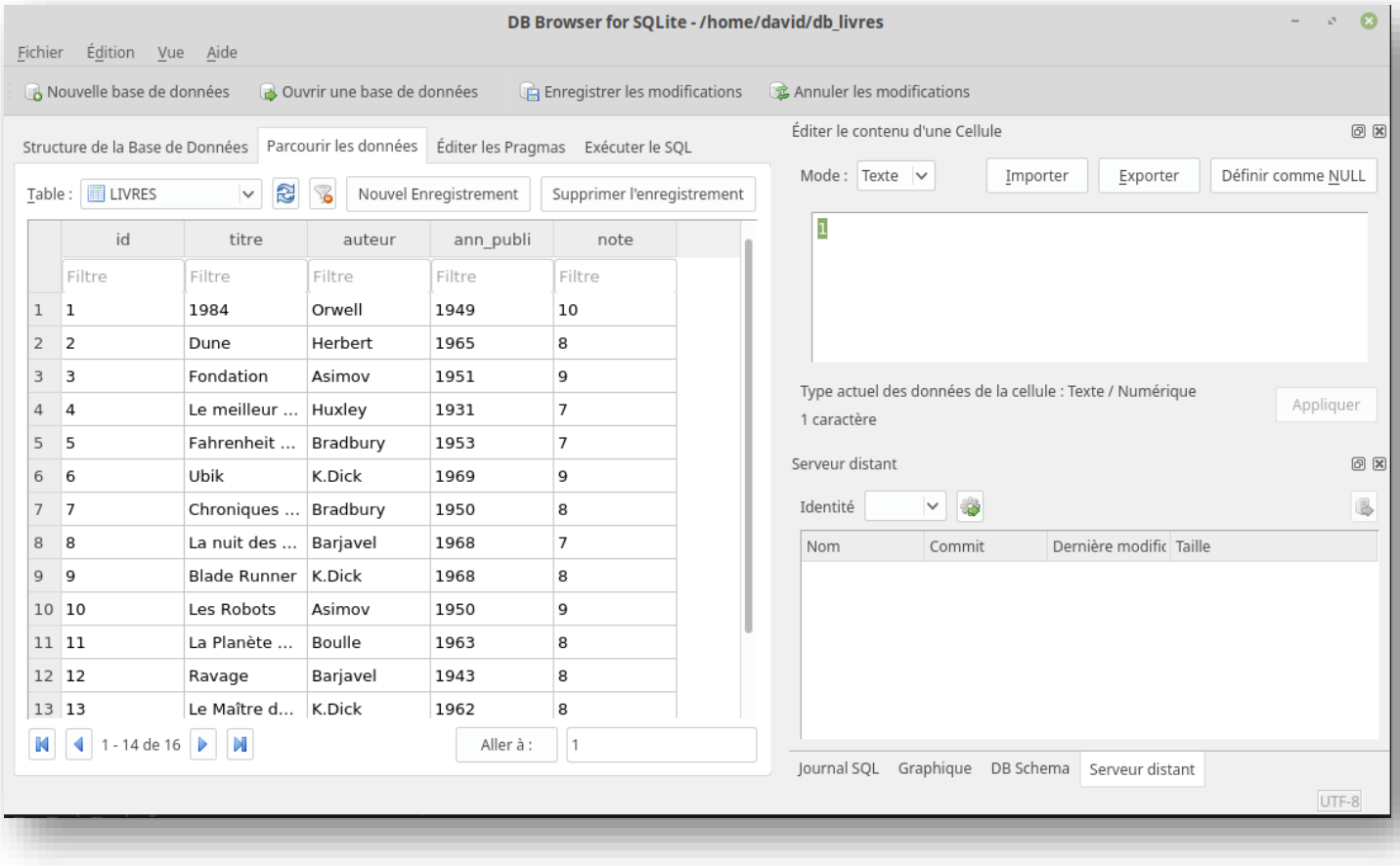
```
        INSERT INTO LIVRES

        (id,titre,auteur,ann_publi,note)

        VALUES

        (1,'1984','Orwell',1949,10),

        (2,'Dune','Herbert',1965,8),

        (3,'Fondation','Asimov',1951,9),

        (4,'Le meilleur des mondes','Huxley',1931,7),

        (5,'Fahrenheit 451','Bradbury',1953,7),

        (6,'Ubik','K.Dick',1969,9),

        (7,'Chroniques martiennes','Bradbury',1950,8),

        (8,'La nuit des temps','Barjavel',1968,7),

        (9,'Blade Runner','K.Dick',1968,8),

        (10,'Les Robots','Asimov',1950,9),
```

```
        (11,'La Planète des singes','Boulle',1963,8),
        (12,'Ravage','Barjavel',1943,8),
        (13,'Le Maître du Haut Château','K.Dick',1962,8),
        (14,'Le monde des Ā','Van Vogt',1945,7),
        (15,'La Fin de l'éternité','Asimov',1955,8),
        (16,'De la Terre à la Lune','Verne',1865,10);
```

Ici aussi, aucun problème, la requête a bien été exécutée :



La table LIVRES contient bien les données souhaitées (onglet "Parcourir les données") :



2- Requêtes d'interrogation

a- Requêtes d'interrogation "simples"

Nous allons apprendre à effectuer des requêtes d'interrogation sur la base de données que nous venons de créer.

Toutes les requêtes se feront dans la fenêtre SQL 1 de l'onglet "**Exécuter le SQL**"

➢ Sélection de données

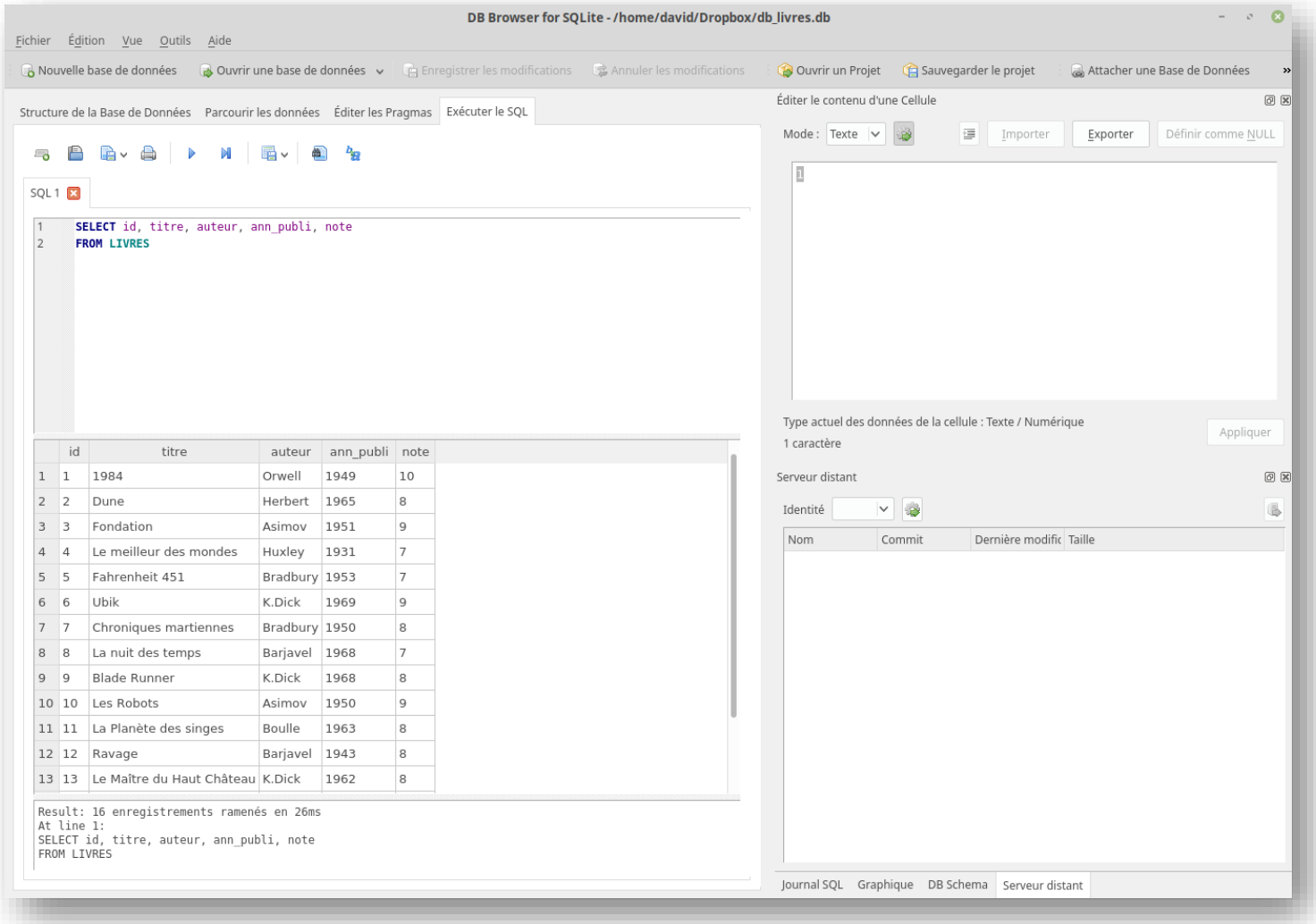
Activité1.8

Saisissez la requête SQL suivante :

```
SELECT id, titre, auteur, ann_publi, note

FROM LIVRES
```

Puis appuyez sur le triangle (ou la touche F5)

Après un temps plus ou moins long, vous devriez voir s'afficher ceci :



Comme vous pouvez le constater, notre requête SQL a permis d'afficher tous les livres. Nous avons ici 2 mots clés du langage SQL *SELECT* qui permet de sélectionner les attributs qui devront être "affichés" (je mets "affichés" entre guillemets, car le but d'une requête sql n'est pas forcément d'afficher les données) et *FROM* qui indique la table qui doit être utilisée.
Il est évidemment possible d'afficher seulement certains attributs (ou même un seul) :

Activité 1.9

Saisissez la requête SQL suivante :

```
SELECT titre, auteur

FROM LIVRES
```

Vérifiez que vous obtenez bien uniquement les titres et les auteurs des livres

Activité 1.10

Écrivez et testez une requête permettant d'obtenir uniquement les titres des livres.

N.B. Si vous désirez sélectionner tous les attributs, vous pouvez écrire :

```
SELECT *
FROM LIVRES
```

à la place de :

```
SELECT id, titre, auteur, ann_publi, note
FROM meslivres
```

*L'astérisque * est un joker (wildcard en anglais).*

b- Sélectionner certaines lignes : la clause WHERE

Pour l'instant nos requêtes affichent tous les livres, il est possible d'utiliser la clause ***WHERE*** afin d'imposer une (ou des) condition(s) permettant de sélectionner uniquement certaines lignes.

Le mot-clé **WHERE** doit être suivi d'un booléen. Les opérateurs classiques = , !=, >, >=, <, <= peuvent être utilisés, mais aussi le mot-clé **IN** :

La condition doit suivre le mot-clé ***WHERE*** :

Activité 1.11

Saisissez et testez la requête SQL suivante :

```
SELECT titre, ann_publi
FROM LIVRES
WHERE auteur='Asimov' ;
```

Vérifiez que vous obtenez bien uniquement les livres écrits par Isaac Asimov.

Activité 1 .12

Écrivez et testez une requête permettant d'obtenir uniquement les titres des livres écrits par Philip K.Dick.

Activité

Saisissez et testez la requête SQL suivante :

```
SELECT titre
FROM LIVRES
WHERE ann_publi IN (1950,1968,1943) ;
```

➢ *WHERE ….OR…. AND*

Il est possible de combiner les conditions à l'aide d'un ***OR*** ou d'un ***AND***

Activité 1.13

Saisissez et testez la requête SQL suivante :

```
SELECT titre, ann_publi
FROM LIVRES
WHERE auteur='Asimov' AND ann_publi>1953 ;
```

Vérifiez que nous obtenons bien le livre écrit par Asimov publié après 1953 (comme vous l'avez sans doute remarqué, il est possible d'utiliser les opérateurs d'inégalités).

Activité 1.14

D'après vous, quel est le résultat de cette requête :

```
SELECT titre
FROM LIVRES
WHERE auteur='K.Dick' OR note>=8 ;
```

Activité 1.15

Écrire une requête permettant d'obtenir les titres des livres publiés après 1945 qui ont une note supérieure ou égale à 9.

➢ Requête approchée : LIKE

Commande :

```
SELECT titre

FROM LIVRES

WHERE titre LIKE '%nuit%';
```

On veut les titres de la table «**LIVRES**» dont le titre contient la chaîne de caractères "**nuit**". Le symbole % est un joker qui peut symboliser n'importe quelle chaîne de caractères.

Remarque :

- *Le ; signale la fin de l'instruction. Il peut donc être omis s'il n'y a pas d'instructions enchaînées (ce qui sera toujours notre cas).*
- *L'indentation n'est pas syntaxique (pas comme en Python). On peut faire des retours à la ligne et des indentations pour rendre le code plus lisible.*

c- Mettre dans l'ordre les réponses : la clause ORDER BY

Il est aussi possible de rajouter la clause SQL ***ORDER BY*** afin d'obtenir les résultats classés dans un ordre précis.

Activité 1.16

Saisissez et testez la requête SQL suivante :

```
SELECT titre

FROM LIVRES

WHERE auteur='K.Dick' ORDER BY ann_publi ;
```

Nous obtenons les livres de K.Dick classés du plus ancien ou plus récent.

Il est possible d'obtenir un classement en sens inverse à l'aide de la clause **DESC**

Activité 1.17

Saisissez et testez la requête SQL suivante :

```
SELECT titre

FROM LIVRES

WHERE auteur='K.Dick' ORDER BY ann_publi DESC ;
```

Nous obtenons les livres de K.Dick classés du plus récent au plus ancien.

Activité 1.18

Que se passe-t-il quand la clause ORDER BY porte sur un attribut de type TEXT ?

d- La clause DISTINCT

Vous pouvez constater qu'une requête du type :

Titre

```
SELECT auteur
FROM LIVRES
```

Affiche plusieurs fois certains auteurs (les auteurs qui ont écrit plusieurs livres présents dans la base de données)

Il est possible d'éviter les doublons grâce à la clause DISTINCT

Activité 1.19

Saisissez et testez la requête SQL suivante :

```
SELECT DISTINCT auteur
FROM LIVRES
```

e- Les jointures

Nous avons vu précédemment qu'une base de données peut contenir plusieurs relations (plusieurs tables).

Activité 1.20

Créez une nouvelle base de données que vous nommerez par exemple db_livres_auteurs.db

Activité 1.21

Créez une table AUTEURS à l'aide de la requête SQL suivante :

```
CREATE TABLE AUTEURS
(id INT, nom TEXT, prenom TEXT, ann_naissance INT, langue_ecriture TEXT, PRIMARY KEY (id));
```

Activité 1.22

Créez une table LIVRES à l'aide de la requête SQL suivante :

```
CREATE TABLE LIVRES
(id INT, titre TEXT, id_auteur INT, ann_publi INT, note INT, PRIMARY KEY (id));
```

Activité 1.23

Ajoutez des données à la table AUTEURS à l'aide de la requête SQL suivante :

```
INSERT INTO AUTEURS
(id,nom,prenom,ann_naissance,langue_ecriture)
VALUES
(1,'Orwell','George',1903,'anglais'),
(2,'Herbert','Frank',1920,'anglais'),
(3,'Asimov','Isaac',1920,'anglais'),
(4,'Huxley','Aldous',1894,'anglais'),
(5,'Bradbury','Ray',1920,'anglais'),
(6,'K.Dick','Philip',1928,'anglais'),
(7,'Barjavel','René',1911,'français'),
(8,'Boulle','Pierre',1912,'français'),
(9,'Van Vogt','Alfred Elton',1912,'anglais'),
(10,'Verne','Jules',1828,'français');
```

Activité 1.24

Ajoutez des données à la table LIVRES à l'aide de la requête SQL suivante :

```
INSERT INTO LIVRES
(id,titre,id_auteur,ann_publi,note)
VALUES
(1,'1984',1,1949,10),
(2,'Dune',2,1965,8),
(3,'Fondation',3,1951,9),
```

```
(4,'Le meilleur des mondes',4,1931,7),
(5,'Fahrenheit 451',5,1953,7),
(6,'Ubik',6,1969,9),
(7,'Chroniques martiennes',5,1950,8),
(8,'La nuit des temps',7,1968,7),
(9,'Blade Runner',6,1968,8),
(10,'Les Robots',3,1950,9),
(11,'La Planète des singes',8,1963,8),
(12,'Ravage',7,1943,8),
(13,'Le Maître du Haut Château',6,1962,8),
(14,'Le monde des Ā',9,1945,7),
(15,'La Fin de l'éternité',3,1955,8),
(16,'De la Terre à la Lune',10,1865,10);
```

Nous avons 2 tables, grâce aux jointures nous allons pouvoir associer ces 2 tables dans une même requête.

En général, les jointures consistent à associer des lignes de 2 tables. Elles permettent d'établir un lien entre 2 tables. Qui dit lien entre 2 tables dit souvent clef étrangère et clef primaire.

Dans notre exemple l'attribut "id_auteur" de la tables LIVRES est bien une clé étrangère puisque cet attribut correspond à l'attribut "id" de la table "AUTEURS".

À noter qu'il est possible de préciser au moment de la création d'une table qu'un attribut jouera le rôle de clé étrangère. Dans notre exemple, à la place d'écrire :

```
CREATE TABLE LIVRES
(id INT, titre TEXT, id_auteur INT, ann_publi INT, note INT, PRIMARY KEY (id));
```

On pourra écrire :

```
CREATE TABLE LIVRES
(id INT, titre TEXT, id_auteur INT, ann_publi INT, note INT, PRIMARY KEY (id), FOREIGN KEY (id_auteur
) REFERENCES AUTEURS(id));
```

grâce à cette précision, sqlite sera capable de détecter les anomalies au niveau de clé étrangère : essayez par exemple d'ajouter un livre à la table LIVRES avec l'attribut "id_auteur" égal à 11 !

Passons maintenant aux jointures :

Activité 1.25

Saisissez et testez la requête SQL suivante :

```
SELECT *

FROM LIVRES

INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id ;
```

Explication

1. **SELECT * :**
    - L'astérisque (*) signifie que nous souhaitons sélectionner toutes les colonnes des tables impliquées dans la requête.
2. **FROM LIVRES :**
    - Nous commençons par spécifier la table principale d'où nous allons sélectionner les données. Ici, la table principale est LIVRES.
3. **INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id :**
    - INNER JOIN est une clause qui permet de joindre deux tables en fonction d'une condition spécifique.
    - Nous « invitons » la table AUTEURS à se joindre à la table LIVRES.
    - La condition de jointure est spécifiée par ON LIVRES.id_auteur = AUTEURS.id.
        - Cela signifie que nous faisons correspondre les lignes de la table LIVRES avec les lignes de la table AUTEURS où la colonne id_auteur de LIVRES est égale à la colonne id de AUTEURS.

Contexte et Importance :

- **Jointure (JOIN)** : Une jointure permet de combiner des lignes de deux ou plusieurs tables en fonction d'une condition de relation entre elles. Dans notre cas, nous utilisons une jointure interne (INNER JOIN), ce qui signifie que seules les lignes ayant une correspondance dans les deux tables seront incluses dans le résultat.

- **Clé primaire et clé étrangère** : id dans la table AUTEURS est une clé primaire, c'est-à-dire un identifiant unique pour chaque auteur. id_auteur dans la table LIVRES est une clé étrangère, utilisée pour établir une relation entre un livre et son auteur.
- **Pourquoi spécifier la condition de jointure** : Spécifier la condition de jointure (LIVRES.id_auteur = AUTEURS.id) est crucial car elle détermine comment les tables sont liées. Sans cette condition, la base de données ne saurait pas quelles lignes de LIVRES doivent être associées à quelles lignes de AUTEURS.

Cette notion de jointure n'est pas évidente, prenez votre temps pour bien réfléchir et surtout n'hésitez pas à poser des questions.

Activité 1.26

Saisissez et testez la requête SQL suivante :

```
SELECT *
FROM AUTEURS
INNER JOIN LIVRES ON LIVRES.id_auteur = AUTEURS.id ;
```

Comme vous pouvez le constater, le résultat est différent, cette fois-ci ce sont les lignes de la table LIVRES qui viennent se greffer sur la table AUTEURS.

Dans le cas d'une jointure, il est tout à fait possible de sélectionner certains attributs et pas d'autres :

Activité 1.27

Saisissez et testez la requête SQL suivante :

```
SELECT nom, prenom, titre
FROM AUTEURS
INNER JOIN LIVRES ON LIVRES.id_auteur = AUTEURS.id ;
```

Activité 1.28

Saisissez et testez la requête SQL suivante :

```
SELECT titre,nom, prenom
FROM LIVRES
INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id ;
```

Si un même nom d'attribut est présent dans les 2 tables (par exemple ici l'attribut id), il est nécessaire d'ajouter le nom de la table devant afin de pouvoir les distinguer (AUTEURS.id et LIVRES.id)

Activité 1.29

Saisissez et testez la requête SQL suivante :

```
SELECT titre,AUTEURS.id,nom, prenom
FROM LIVRES
INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id
```

f- Utilisation du WHERE dans les jointures

Il est possible d'utiliser la clause **WHERE** dans le cas d'une jointure :

Activité 1.30

Saisissez et testez la requête SQL suivante :

```
SELECT titre,nom, prenom
FROM LIVRES
INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id
WHERE ann_publi>1950
```

g- Les jointures plus complexes

Enfin, pour terminer avec les jointures, vous devez savoir que nous avons abordé la jointure la plus simple (INNER JOIN). Il existe des jointures plus complexes (CROSS JOIN, LEFT JOIN, RIGHT JOIN), ces autres jointures ne seront pas abordées ici.

Nous en avons terminé avec les requêtes d'interrogation, intéressons-nous maintenant aux requêtes de mise à jour (INSERT, UPDATE, DELETE).

Nous allons repartir avec une base de données qui contient une seule table :

Activité 1 .31

Créez une nouvelle base de données que vous nommerez par exemple db_livres.db

Activité 1.32

Créez une table LIVRES à l'aide de la requête SQL suivante :

```
CREATE TABLE LIVRES
(id INT, titre TEXT, auteur TEXT, ann_publi INT, note INT, PRIMARY KEY (id));
```

Activité 1.33

Ajoutez des données à la table LIVRES à l'aide de la requête SQL suivante :

```
INSERT INTO LIVRES
(id,titre,auteur,ann_publi,note)
VALUES
(1,'1984','Orwell',1949,10),
(2,'Dune','Herbert',1965,8),
(3,'Fondation','Asimov',1951,9),
(4,'Le meilleur des mondes','Huxley',1931,7),
(5,'Fahrenheit 451','Bradbury',1953,7),
(6,'Ubik','K.Dick',1969,9),
(7,'Chroniques martiennes','Bradbury',1950,8),
(8,'La nuit des temps','Barjavel',1968,7),
(9,'Blade Runner','K.Dick',1968,8),
(10,'Les Robots','Asimov',1950,9),
(11,'La Planète des singes','Boulle',1963,8),
(12,'Ravage','Barjavel',1943,8),
(13,'Le Maître du Haut Château','K.Dick',1962,8),
(14,'Le monde des Ā','Van Vogt',1945,7),
```

```
(15,'La Fin de l'éternité','Asimov',1955,8),
(16,'De la Terre à la Lune','Verne',1865,10);
```

3- Requêtes d'insertion

Nous avons déjà eu l'occasion de voir la requête permettant d'ajouter une entrée (utilisation d'INSERT)

Activité 1.34

Que va faire cette requête ? Vérifiez votre réponse en l'exécutant et en faisant une requête "SELECT * FROM LIVRES".

```
INSERT INTO LIVRES
(id,titre,auteur,ann_publi,note)
VALUES
(17,'Hypérion','Simmons',1989,8);
```

Activité 1.35

Écrivez et testez une requête permettant d'ajouter le livre de votre choix à la table LIVRES.

4- Requêtes de mise à jour

"**UPDATE**" va permettre de modifier une ou des entrées. Nous utiliserons "**WHERE**", comme dans le cas d'un "**SELECT**", pour spécifier les entrées à modifier.

Voici un exemple de modification :

Activité 1.36

Que va faire cette requête ? Vérifiez votre réponse en l'exécutant et en faisant une requête "SELECT * FROM LIVRES".

```
UPDATE LIVRES
SET note=7
WHERE titre = 'Hypérion'
```

Activité 1.37

Écrivez une requête permettant d'attribuer la note de 10 à tous les livres écrits par Asimov publiés après 1950. Testez cette requête.

5- Requêtes de suppression

a- Suppression d'un enregistrement : DELETE

"**DELETE**" est utilisée pour effectuer la suppression d'une (ou de plusieurs) entrée(s). Ici aussi c'est le "**WHERE**" qui permettra de sélectionner les entrées à supprimer.

Activité 1.38

Que va faire cette requête ? Vérifiez votre réponse en l'exécutant et en faisant une requête "SELECT * FROM LIVRES".

```
DELETE FROM LIVRES
WHERE titre='Hypérion' ;
```

Cette requête permet de supprimer le(les) livre(s) ayant pour titre Hypérion

Attention à l'utilisation de cette requête DELETE notamment si on oublie le WHERE. Un :

```
DELETE FROM LIVRES ;
```

supprimerait toutes les entrées de la table LIVRES

Activité1.39

Écrivez une requête permettant de supprimer les livres publiés avant 1945. Testez cette requête.

b- Suppression totale d'une table : DROP TABLE

Pour supprimer totalement et décisivement la table :

```
DROP TABLE LIVRES;
```

Là encore, si une autre table est reliée à « LIVRES » par une clé étrangère, la suppression sera bloquée par le SGBD.

6- Orders SQL COUNT()

En SQL, la fonction d'agrégation **COUNT**() permet de compter le nombre d'enregistrement dans une table. Connaître le nombre de lignes dans une table est très pratique dans de nombreux cas, par exemple pour savoir combien d'utilisateurs sont présents dans une table ou pour connaître le nombre de commentaires sur un article.

➢ Syntaxe

Pour connaître le nombre de lignes totales dans une table, il suffit d'effectuer la requête SQL suivante :

```
SELECT COUNT(*) FROM table
```

Il est aussi possible de connaitre le nombre d'enregistrement sur une colonne en particulier. Les enregistrements qui possèdent la valeur nulle ne seront pas comptabilisés. La syntaxe pour compter les enregistrements sur la colonne "**nom_colonne**" est la suivante :

```
SELECT COUNT(nom_colonne) FROM table
```

Enfin, il est également possible de compter le nombre d'enregistrement distinct pour une colonne. La fonction ne comptabilisera pas les doublons pour une colonne choisie. La syntaxe pour compter le nombre de valeur distincte pour la colonne "nom_colonne" est la suiante :

```
SELECT COUNT(DISTINCT nom_colonne) FROM table
```

➢ Utilisation de COUNT(*)

Pour compter le nombre d'utilisateurs total depuis que le site existe, il suffit d'utiliser COUNT(*) sur toute la table :

```
SELECT COUNT(*) FROM utilisateur
```

➢ Utilisation de COUNT(*) avec WHERE

Pour compter le nombre d'utilisateur qui ont effectué au moins un achat, il suffit de faire la même chose mais en filtrant les enregistrements avec WHERE :

```
SELECT COUNT(*) FROM utilisateur WHERE nombre_achat > 0
```

➢ Utilisation de COUNT(colonne)

Une autre méthode permet de compter le nombre d'utilisateurs ayant effectué au moins un achat. Il est possible de compter le nombre d'enregistrement sur la colonne "id_dernier_achat". Sachant que la valeur est nulle s'il n'y a pas d'achat, les lignes ne seront pas comptées s'il n'y a pas eu d'achat. La requête est donc la suivante :

```
SELECT COUNT(id_dernier_achat) FROM utilisateur
```

Révision

Ce qu'il faut savoir

- Pour consulter des données, ajouter une entrée, modifier une entrée ou supprimer une entrée dans une base de données relationnelle, il est nécessaire d'effectuer des "requêtes SQL" (utilisation du langage SQL)
- Pour ajouter des entrées à une table, on utilisera "INSERT" (exemple : INSERT INTO LIVRES (id,titre,auteur,ann_publi,note) VALUES (1,'1984','Orwell',1949,10);)
- Pour interroger une table, on utilisera "SELECT" (exemple : SELECT titre FROM LIVRES WHERE auteur='Asimov')
- Pour modifier une entrée, on utilisera "UPDATE" (exemple : UPDATE LIVRES SET note=7 WHERE titre = 'Hypérion')
- Pour supprimer une entrée, on utilisera "DELETE" (exemple : DELETE FROM LIVRES WHERE titre='Hypérion')
- Pour réaliser une jointure, il est possible d'utiliser "INNER JOIN" (exemple : SELECT FROM LIVRES INNER JOIN AUTEURS ON LIVRES.id_auteur = AUTEURS.id)

Ce qu'il faut savoir faire

- Vous devez être capable d'effectuer des requêtes SQL simples (utilisation de "INSERT", "SELECT", "UPDATE" et "DELETE")
- Vous devez être capable d'effectuer une jointure entre 2 tables (utilisation de "INNER JOIN")

Exercices du bac

- Sujet 1 2021 Exercice 4
- Sujet 2 2021 Exercice 3
- Sujet 3 2021 Exercice 2
- Sujet 4 2021 Exercice 1
- Sujet 5 2021 Exercice 1
- Sujet 7 2021 Exercice 4
- Sujet 8 2021 Exercice 1
- Sujet 10 2021 Exercice 3
- Sujet 1 2022 Exercice 3
- Sujet 2 2022 Exercice 2
- Sujet 3 2022 Exercice 4
- Sujet 4 2022 Exercice 1
- Sujet 5 2022 Exercice 3
- Sujet 6 2022 Exercice 4
- Sujet 7 2022 Exercice 3
- Sujet 8 2022 Exercice 2
- Sujet 9 2022 Exercice 4
- Sujet 10 2022 Exercice 3
- Sujet 11 2022 Exercice 2
- Sujet 12 2022 Exercice 3
- Sujet 13 2022 Exercice 1
- Sujet 14 2022 Exercice 3