Introduction

Dans un chapitre précédent, nous avons présenté les problématiques des Systèmes de Gestion de Bases de Données (SGBD) et le modèle relationnel qui est le principal modèle logique de structuration des données utilisé depuis les années 1970. Le langage SQL permet de définir et manipuler ces données, il a été développé chez IBM sur le SGBD System R.

Les exercices du cours peuvent être testés dans le fichier Capytale d'adresse :

`https://capytale2.ac-paris.fr/web/c/7a2b-610499`

Dans ce chapitre nous utiliserons les termes de table, ligne, colonne plutôt que leurs équivalents relation, nuplet, attribut.

Sources :

- *« Cours de bases de données – Modèles et langages » de Philippe Rigaux.*
- *« Bases de données – Modèles et langages » de Jean-Luc Hainaut aux éditions Dunod.*
- *« Manuel de NSI » de T. Balabonski, S. Conchon, JC. Filliâtre, K.Nguyen aux éditions Ellipse.*
- *« Cours de Terminale NSI » de Gilles Lassus :* `https://glassus.github.io/terminale_nsi/`.

1 Présentation du langage SQL et de la base `portables.db`

Définition 1 ***langage SQL***

Le langage Structured_Query_Language ou **SQL**, permet d'interagir avec un Système de Gestion de Bases de Données (**SGBD**) à travers des **requêtes**.

Le mode d'interaction suit en général *l'architecture client / serveur*, avec un client SQL interrogeant le serveur SGBD. Néanmoins le SGBD que nous utiliserons, SQLITE fonctionne sans serveur : la base de données est stockée dans un fichier d'extension `.db` en général et le SGBD et le client sont inclus directement dans le logiciel permettant de manipuler la base.

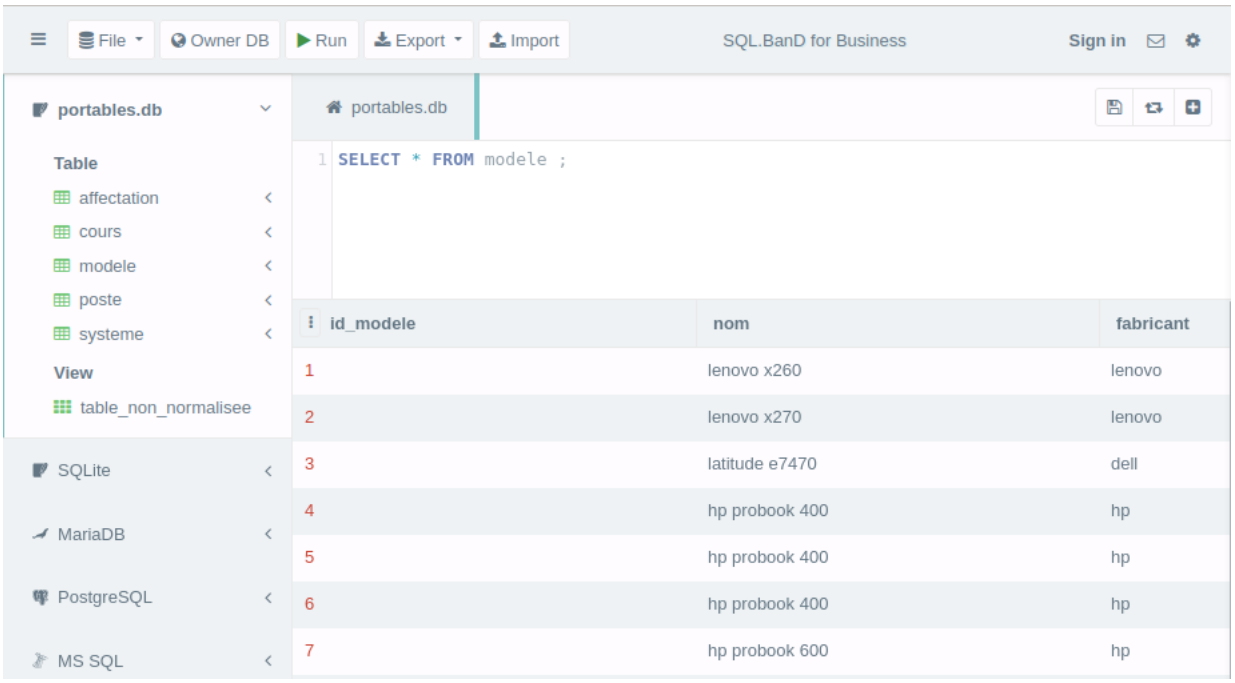
On distingue deux types de requêtes SQL :

- ☞ Les requêtes qui permettent de définir et modifier le schéma de la base de données, constituent l'aspect *Data Definition Language* de SQL. Cette partie n'est pas au programme de terminale NSI.
- ☞ Les requêtes qui permettent d'extraire des données ou de modifier les contenus des tables (mais pas leur schéma), constituent l'aspect *Data Modification Language* de SQL. Le programme de terminale NSI s'intéresse uniquement à cette partie.

Méthode *Utilisation d'un client SQLite*

Pour formuler des requêtes SQL sur une base de données, il faut utiliser un logiciel client. Pour les bases de données SQLite nous utiliserons trois clients :

☞ Un client en ligne `https://sqliteonline.com/`. Il suffit d'ouvrir la base puis de saisir les requêtes dans la console. Quand la session est terminée, on pense à sauvegarder ses requêtes dans un fichier texte, d'extension `.sql`.

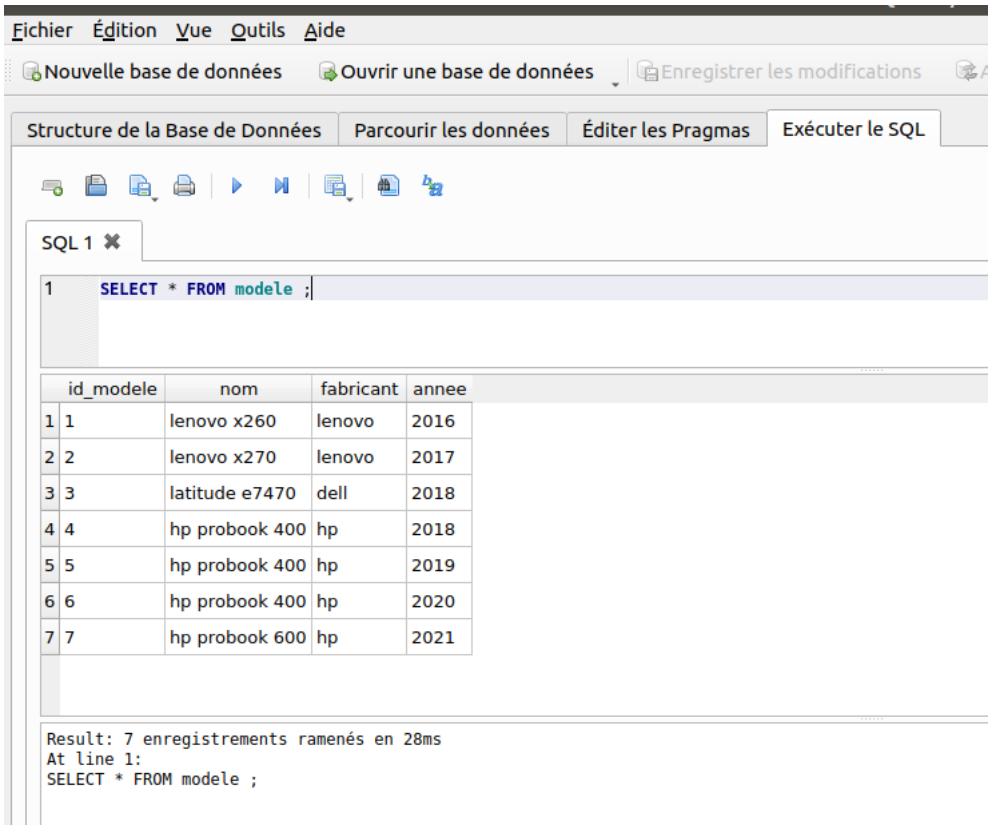


☞ Le client fourni dans le type d'activité SQL du service Capytale de votre ENT. L'avantage est de pouvoir enregistrer toutes les requêtes d'une session. Le fichier avec la base de données est attaché à l'activité.

```
Entrée[1]: SELECT * FROM modele ;
Sortie[1]:
```

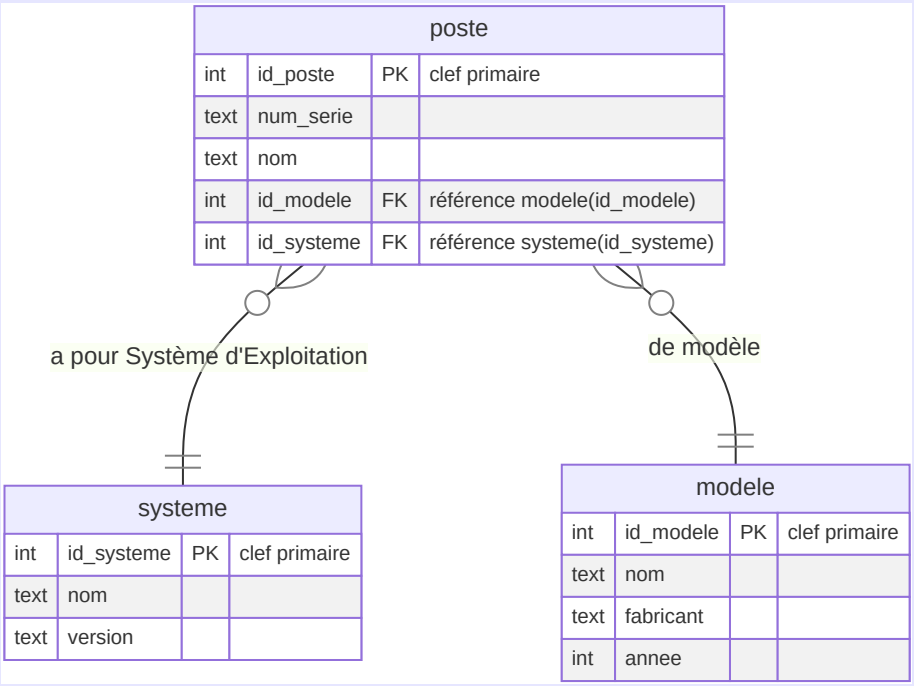
id_modele	nom	fabricant	annee
1	lenovo x260	lenovo	2016
2	lenovo x270	lenovo	2017
3	latitude e7470	dell	2018
4	hp probook 400	hp	2018
5	hp probook 400	hp	2019
6	hp probook 400	hp	2020
7	hp probook 600	hp	2021

☞ Le logiciel client DB Browser for SQLite, téléchargeable sur `https://sqlitebrowser.org/`. Le fonctionnement est le même que pour `https://sqliteonline.com/`.



Outil 1 *Base* `portables.db` *(1/2)*

La plupart des exemples et exercices de ce cours porteront sur une base `portables.db` représentant un parc d'ordinateurs portables dans un département informatique d'université. La base complète est constituée de 5 tables mais dans un premier temps on travaillera uniquement sur 3 tables en ignorant les deux autres. Le schéma relationnel de cette base restreinte est donné ci-dessous. Les clefs primaires sont repérées par **PK** (*Primary Key*) et les clefs étrangères par **FK** (*Foreign Key*).



graphique 1

On rappelle le lien vers l'activité Capytale permettant de manipuler cette base :
https://capytale2.ac-paris.fr/web/c/7a2b-610499.

Complément 1 *Création d'une base de données en SQL*

Cette partie n'est pas au programme de Terminale NSI, mais il vaut mieux connaître la syntaxe de la clause CREATE car elle peut apparaître dans certains énoncés.

La clause SQL `CREATE` permet de créer les relations/tables d'une base de données en précisant pour chacune son **schéma relationnel** avec ses *contraintes d'intégrité* :

- *contrainte de domaine* associant à chaque attribut/colonne son domaine ;
- *contrainte de relation* avec déclaration d'une **clef primaire** pour chaque relation ;
- *contrainte d'intégrité référentielle* dans la déclaration des éventuelles **clefs étrangères** avec les références aux clefs primaires d'autres relations qui leur sont liées.

Voici un exemple de requêtes permettant de créer le schéma relationnel de la base `portables.db` représentée sur le **graphique 1** page 4. On peut noter que les relations/tables `modele` et `systeme` sont créées avant la relation `poste` qui leur fait référence.

```
CREATE TABLE modele(
      id_modele INT PRIMARY KEY,
      nom TEXT,
      fabricant TEXT,
      annee INT
);

CREATE TABLE systeme(
      id_systeme INT PRIMARY KEY,
      nom TEXT,
      version TEXT
);

CREATE TABLE poste(
      id_poste INT PRIMARY KEY,
      num_serie TEXT UNIQUE NOT NULL,
      nom TEXT,
      id_modele INT,
      id_systeme INT,
      FOREIGN KEY (id_modele) REFERENCES modele(id_modele),
      FOREIGN KEY (id_systeme) REFERENCES systeme(id_systeme)
);
```

Exercice 1

1. Afficher le contenu de la table `poste` dans le fichier Capytale avec `SELECT * FROM poste ;`

   Quelles autres clefs primaires aurait-on pu choisir dans la table `poste` ?

2. D'après vous quel est le rôle de la clause `UNIQUE` ? et de `NOT NULL` ?

2 Requêtes d'interrogation sur une seule table

Dans cette section, on considère une table d'une base de données dont on veut extraire certaines informations. Lors de nos tests, on affichera simplement les données extraites sur la sortie de la console mais en TP on pourra utiliser un programme Python pour récupérer ces données et les traiter.

2.1 Extraction simple de lignes

Méthode

Pour extraire le contenu de *certaines colonnes* d'une table, la syntaxe minimale est constituée de la clause `SELECT` suivie des noms de colonnes séparés par des virgules puis la clause `FROM` suivie du nom de la table.

```
SELECT colonne1, colonne2, ..., colonnep
FROM table ;
```

Pour extraire le contenu de *toutes les colonnes* d'une table, on utilise l'attrape-tout `*` :

```
SELECT *
FROM table ;
```

Remarque 1

- Une requête `SELECT` renvoie une table dite calculée.
- En considérant une table comme un ensemble, on parle de **projection** sur la ou les colonnes après le `SELECT`.
- La fin d'une requête SQL est marquée par un point virgule, facultatif si la requête est isolée.
- SQL n'est pas sensible à la casse des caractères pour les mots clefs, les noms des tables, des colonnes (mais la casse est importante pour les valeurs de type TEXT, VARCHAR ...)

  Ainsi `SELECT Nom FROM SysTeme ;` donne le même résultat que `SELECT nom FROM systeme ;`.

  Néanmoins il est préférable de rester homogène dans son style : on recommande d'écrire les mots clefs en majuscule et de respecter la casse sur les noms de table ou de colonnes.

Exemple 1

Considérons notre base `portables.db` dont le schéma relationnel est donné dans le graphique 1 page 4.

1. Pour extraire les colonnes avec le nom du système d'exploitation de la table `systeme`, on écrira :

```
SELECT nom
FROM systeme ;
```

nom
debian
debian
fedora linux
fedora linux
ubuntu
ubuntu

2. Pour extraire toutes les colonnes de la table `modele`, on écrira :

```
SELECT *
FROM modele ;
```

id_modele	nom	fabricant	annee
1	lenovo x260	lenovo	2016
2	lenovo x270	lenovo	2017
3	latitude e7470	dell	2018
4	hp probook 400	hp	2018
5	hp probook 400	hp	2019
6	hp probook 400	hp	2020
7	hp probook 600	hp	2021

3. Pour extraire les colonnes avec le fabricant et l'année de livraison de la table `modele`, on écrira :

```sql
SELECT  fabricant, annee
FROM modele ;
```

fabricant	annee
lenovo	2016
lenovo	2017
dell	2018
hp	2018
hp	2019
hp	2020
hp	2021

On peut noter avec ce dernier exemple que la table calculée, comporte des doublons. Pour les éliminer, on peut utiliser la clause `DISTINCT`.

Méthode

On élimine les doublons dans les lignes du résultat d'une requête en préfixant la liste des colonnes par la clause `DISTINCT` :

```sql
SELECT DISTINCT nom
FROM systeme ;
```

nom
debian
fedora linux
ubuntu

Exercice 2

Considérons notre base `portables.db` dont le schéma relationnel est donné dans le graphique 1 page 4.

1. Écrire une requête qui extrait les colonnes `id_modele` et `id_systeme` de la table poste, en éliminant les doublons sur les lignes du résultat.
2. Écrire une requête qui extrait les colonnes `id_poste`, `id_modele` et `id_systeme` de la table poste. Est-il nécessaire d'éliminer les doublons sur les lignes du résultat ?

2.2 Extraction avec sélection de lignes, requête SFW

Exemple 2

Nous avons vu comment extraire certaines colonnes d'un table mais comment sélectionner des lignes ?

1. La requête pour sélectionner toutes les colonnes de la table `modele` mais seulement les lignes dont le fabricant est `'hp'` peut s'écrire :

```
SELECT *
FROM modele
WHERE fabricant = 'hp' ;
```

id_modele	nom	fabricant	annee
4	hp probook 400	hp	2018
5	hp probook 400	hp	2019
6	hp probook 400	hp	2020
7	hp probook 600	hp	2021

2. La requête pour sélectionner toutes les colonnes de la table `modele` mais seulement les lignes dont l'année de livraison est inférieure à 2020 peut s'écrire :

```
SELECT *
FROM modele
WHERE annee < 2020 ;
```

id_modele	nom	fabricant	annee
1	lenovo x260	lenovo	2016
2	lenovo x270	lenovo	2017
3	latitude e7470	dell	2018
4	hp probook 400	hp	2018
5	hp probook 400	hp	2019

Définition 2 *Requête SFW*

Une requête SQL permettant d'effectuer une projection sur les colonnes d'une table et une sélection sur les lignes, comporte trois parties :

- la clause `SELECT` précise les noms des colonnes sur lesquelles on projette les lignes du résultat ;

- ☞ la clause `FROM` précise la table dont on tire le résultat ;
- ☞ la clause `WHERE` spécifie la condition de sélection que doivent vérifier les lignes du résultat.

On désigne ces requêtes par l'acronyme `SFW`.

Méthode *condition de sélection*

La condition de sélection associée à une clause `WHERE` est une condition booléenne.
Une condition simple peut être construite à partir d'opérateurs de comparaison. Dans le paramètre du filtre de motif textuel, le symbole `'?'` désigne un caractère quelconque et le symbole `'%'` désigne zéro ou plusieurs caractères quelconques.

Opérateur	Sémantique	Exemple
=	égal à	SELECT * FROM modele WHERE fabricant = 'hp' ;
<> ou !=	différent de	SELECT * FROM systeme WHERE fabricant <> 'hp' ;
<	plus petit que	SELECT * FROM modele WHERE annee < 2019 ;
<=	plus petit ou égal à	SELECT * FROM modele WHERE annee <= 2019 ;
>	plus grand que	SELECT * FROM modele WHERE annee > 2019 ;
>=	plus grand ou égal à	SELECT * FROM modele WHERE annee >= 2019 ;
`LIKE`	filtre de motif	SELECT * FROM modele WHERE nom LIKE '%probook%' ;

On peut composer plusieurs conditions simples avec les opérateurs booléens `NOT`, `AND` et `OR` pour créer des conditions complexes. Il est recommandé d'utiliser des parenthèses.

Opérateur	Sémantique	Exemple
`OR`	ou	SELECT * FROM modele WHERE (annee < 2019) OR (nom LIKE '%probook%') ;
`AND`	et	SELECT * FROM modele WHERE (annee < 2019) AND (fabricant <> 'dell') ;
`NOT`	négation	SELECT * FROM modele WHERE NOT((annee < 2021) AND (fabricant = 'hp')) ;

Exercice 3

On donne le contenu de la table `modele` :

```
SELECT *
FROM modele ;
```

id_modele	nom	fabricant	annee
1	lenovo x260	lenovo	2016
2	lenovo x270	lenovo	2017
3	latitude e7470	dell	2018
4	hp probook 400	hp	2018
5	hp probook 400	hp	2019
6	hp probook 400	hp	2020
7	hp probook 600	hp	2021

Donner les tables calculées pour chacune des requêtes listées dans le point de méthode précédent.

2.3 Ordonner les lignes

Méthode *ordre sur les lignes du résultat*

L'ordre des lignes du résultat d'une requête SFW est a priori arbitraire. Cependant on peut trier les lignes par ordre croissant ou décroissant selon une ou plusieurs colonnes (par ordre lexicographique) avec la clause `ORDER BY` combinée avec `ASC` (*croissant*) ou `DESC` (*décroissant*).

La clause `ORDER BY` doit être toujours en dernière position dans une requête et c'est la dernière opération effectuée.

Pour afficher la colonne `nom` de la table `systeme` par ordre alphabétique *croissant* on écrira :

```
SELECT DISTINCT nom
FROM systeme
ORDER BY nom ASC ;
```

nom
debian
fedora linux
ubuntu

Pour afficher la colonne `nom` de la table `systeme` par ordre alphabétique *décroissant* on écrira :

```sql
SELECT DISTINCT nom
FROM systeme
ORDER BY nom DESC ;
```

nom
ubuntu
fedora linux
debian

Exercice 4 *Centres-Étrangers 2022 sujet 1*

Dans le cadre d'une étude sur le réchauffement climatique, un centre météorologique rassemble des données. On considère que la base de données contient deux relations (tables). La relation `Centres` est un ensemble de nuplets constitués d'un identifiant de centre météorologique, la ville, la latitude, la longitude et l'altitude du centre. La relation `Mesures` est un ensemble de nuplets constitués d'un identifiant de mesure, l'identifiant du centre, la date de la mesure, la température, la pression et la pluviométrie mesurées.

Le schéma relationnel de la relation `Centres` est :

```
Centres(id_centre: INT, nom_ville: VARCHAR, latitude: FLOAT, longitude:
    FLOAT, altitude: FLOAT)
```

Le schéma relationnel de la relation `Mesures` est :

```
Mesures(id_mesure: INT, id_centre: INT, date: DATE, temperature: FLOAT,
    pression: INT, pluviometrie: FLOAT)
```

Relation `Centres`

id_centre	nom_ville	latitude	longitude	altitude
213	Amiens	49.894	2.293	60
138	Grenoble	45.185	5.723	550
263	Brest	48.388	-4.49	52
185	Tignes	45.469	6.909	2594
459	Nice	43.706	7.262	260
126	Le Puy-en-Velay	45.042	3.888	744
317	Gérardmer	48.073	6.879	855

Relation `Mesures`

id_mesure	id_centre	date	temperature	pression	pluviometrie
1566	138	2021-10-29	8.0	1015	3
1568	213	2021-10-29	15.1	1011	0
2174	126	2021-10-30	18.2	1023	0
2200	185	2021-10-30	5.6	989	20
2232	459	2021-10-31	25.0	1035	0
2514	213	2021-10-31	17.4	1020	0
2563	126	2021-11-01	10.1	1005	15
2592	459	2021-11-01	23.3	1028	2
3425	317	2021-11-02	9.0	1012	13
3430	138	2021-11-02	7.5	996	16
3611	263	2021-11-03	13.9	1005	8
3625	126	2021-11-03	10.8	1008	8

1. Proposer une clé primaire pour la relation Mesures. Justifier.

2. Qu'affiche la requête suivante ? `SELECT * FROM Centres WHERE altitude>500;`
3. On souhaite récupérer le nom de la ville des centres météorologiques situés à une altitude comprise entre 700 m et 1200 m. Ecrire la requête SQL correspondante.
4. On souhaite récupérer la liste des longitudes et des noms des villes des centres météorologiques dont la longitude est supérieure à 5. La liste devra être triée par ordre alphabétique des noms de ville. Ecrire la requête SQL correspondante.
5. Qu'affiche la requête suivante ? `SELECT * FROM Mesures WHERE date="2021-10-30";`

2.4 Valeurs calculées et alias de colonnes

Méthode

En plus des colonnes de la table, il est possible de préciser comme paramètres de la clause `SELECT` des constantes ou des valeurs calculées à partir de valeurs de la table.
On définit ainsi de nouvelles colonnes qu'il est possible de nommer avec `AS alias`.
Par exemple pour calculer l'âge des modèles d'ordinateur disponibles dans la table `modele` et nommer cette nouvelle colonne, on peut écrire :

```
SELECT nom, fabricant, 2022 - annee AS age
FROM modele ;
```

nom	fabricant	age
lenovo x260	lenovo	6
lenovo x270	lenovo	5
latitude e7470	dell	4
hp probook 400	hp	4
hp probook 400	hp	3
hp probook 400	hp	2
hp probook 600	hp	1

2.5 Fonctions d'agrégation

Méthode

On peut passer comme paramètres de la clause `SELECT`, des valeurs obtenues à l'aide de **fonctions d'agrégation** : elles calculent des valeurs agrégées à partir de l'ensemble des lignes sélectionnées.

- ☞ `COUNT(*)` donne le nombre de lignes dans la table ;
- ☞ `COUNT(nom_colonne)` donne le nombre de valeurs (doublons compris) dans la colonne ;
- ☞ `COUNT(DISTINCT nom_colonne)` donne le nombre de valeurs distinctes dans la colonne ;
- ☞ `MAX(nom_colonne)` donne la valeur maximale de la colonne ;

- ☞ `MIN(nom_colonne)` donne la valeur minimale de la colonne ;
- ☞ `SUM(nom_colonne)` donne la somme des valeurs de la colonne (si la somme est possible !) ;
- ☞ `AVG(nom_colonne)` donne la valeur moyenne de la colonne (si la moyenne est possible !).

Par exemple pour calculer l'année de livraison minimale, l'année maximale et l'âge moyen des modèle d'ordinateur disponibles dans la table `modele`, on peut écrire :

```
SELECT MIN(annee) AS annee_min, MAX(annee) AS annee_max, AVG(2022 - ANNEE) AS age_moyen
FROM modele ;
```

annee_min	annee_max	age_moyen
2016	2021	3.5714285714285716

⚠ *Bien entendu, on aimerait calculer ces statistiques sur la table `poste` contenant les informations sur les postes, on le fera un peu plus tard après avoir joint les informations des tables `poste` et `modele`.*

Exercice 5

Considérons notre base `portables.db` dont le schéma relationnel est donné dans le graphique 1 page 4.

1. Écrire une requête SQL qui permet d'obtenir le nombre de postes dans la table `poste` pour lesquels la valeur de l'attribut `id_systeme` est 5 (système `ubuntu` en version `18.04` installé).
2. Écrire une requête SQL qui permet d'obtenir la valeur minimale de l'attribut `id_poste` pour les postes dont le numéro de série comporte au moins un 4.

3 Requêtes d'interrogation sur plusieurs tables, jointures

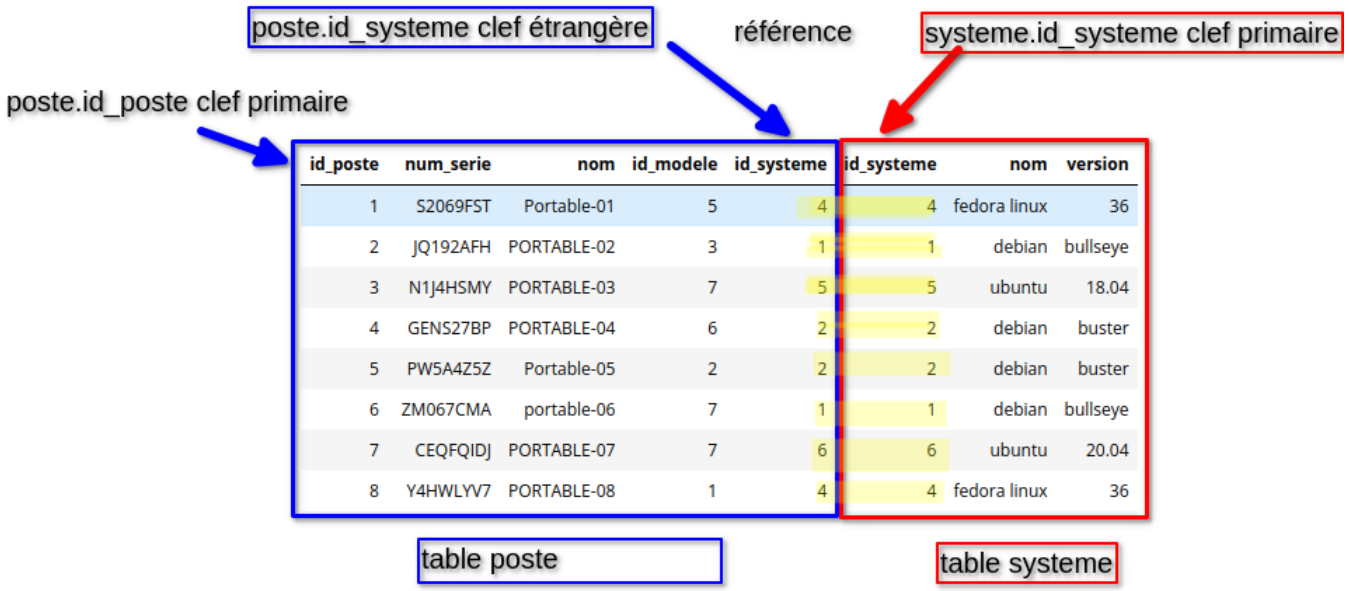
3.1 Jointure entre deux tables

Définition 3

Examinons l'extrait de table ci-dessous obtenu en rapprochant les informations contenues dans les tables `poste` et `systeme` de notre base `portables.db`.
On peut observer que le rapprochement des lignes s'est effectué sous la condition d'égalité entre la colonne de nom `id_systeme` de la table `poste` et la colonne de même nom de la table `systeme`.
Fixons le vocabulaire :

- ☞ La table rassemblant les informations correspondantes des tables `poste` et `systeme` a été obtenue par **jointure** de ces deux tables.
- ☞ La **condition de jointure** pour sélectionner les lignes est l'égalité entre la valeur d'une clef étrangère de la table `poste` et de la clef primaire référencée dans la table `systeme`. On peut repérer cette dépendance entre clef étrangère et primaire dans le schéma relationnel de la base page 4.

*La **jointure** est donc l'opération qui permet de reconstituer l'information séparée entre différentes relations/tables lors de la normalisation du schéma d'une base de données (voir chapitre SGBD et modèle relationnel).*



id_poste	num_serie	nom	id_modele	id_systeme
1	S2069FST	Portable-01	5	4
2	JQ192AFH	PORTABLE-02	3	1
3	N1J4HSMY	PORTABLE-03	7	5
4	GENS27BP	PORTABLE-04	6	2
5	PW5A4Z5Z	Portable-05	2	2
6	ZM067CMA	portable-06	7	1
7	CEQFQIDJ	PORTABLE-07	7	6
8	Y4HWLYV7	PORTABLE-08	1	4

id_systeme	nom	version
4	fedora linux	36
1	debian	bullseye
5	ubuntu	18.04
2	debian	buster
2	debian	buster
1	debian	bullseye
6	ubuntu	20.04
4	fedora linux	36

graphique 2

3.2 Jointure en SQL

Méthode *Jointure entre deux tables en SQL*

On peut interpréter ainsi la jointure entre la table `poste` et la table `systeme` :

- On construit une table associant chaque ligne de la table `poste` à chaque ligne de la table `systeme` (*produit cartésien*) :

```
FROM poste, systeme
```

- On sélectionne uniquement les lignes pertinentes c'est-à-dire celles où on a une égalité entre la clef étrangère `poste.id_systeme` et la clef primaire référencée `systeme.id_systeme` car ce sont les attributs qui font un lien logique entre `poste` et `systeme` :

```
WHERE poste.id_systeme = systeme.id_systeme
```

- On projette sur les colonnes (ici toutes) qui nous intéressent après le `SELECT`.

```
SELECT *
```

Les attributs intervenant dans la condition de jointure peuvent porter, comme dans cet exemple, les mêmes noms. Pour lever toute ambiguïté, on préfixe chacun par le nom de sa table.

Une première requête de jointure entre les tables `poste` et `systeme` serait donc :

```
SELECT *
FROM poste, systeme
WHERE poste.id_systeme = systeme.id_systeme ;
```

Néanmoins si on veut opérer une sélection sur les lignes après la jointure, par exemple ne garder que les postes de système `'debian'`, sur les lignes de la table obtenue par jointure, il faut rajouter une condition dans le `WHERE` et on a une confusion entre les conditions de jointure et de sélection. Au passage on remarque qu'on préfixe systématiquement les attributs par le nom de la table s'il peut y avoir une ambiguité entre des noms partagés par des attributs des deux tables :

```
SELECT *
FROM poste, systeme
WHERE (poste.id_systeme = systeme.id_systeme) AND (systeme.nom='ubuntu');
```

On préfère donc une syntaxe spécifique pour la jointure qui est incluse dans le `FROM` avec une clause `ON` pour préciser la condition de jointure :

```
SELECT *
FROM poste JOIN systeme ON poste.id_systeme = systeme.id_systeme ;
```

On peut alors ajouter une sélection sur les lignes après jointure avec la clause `WHERE` :

```
SELECT *
FROM poste JOIN systeme ON poste.id_systeme = systeme.id_systeme
WHERE systeme.nom = 'ubuntu' ;
```

On peut cependant noter qu'avec un `SELECT *`, les colonnes `poste.id_systeme` et `systeme.id_systeme` ont le même nom `id_systeme` dans la table obtenue par jointure. Pour corriger ce problème, on peut en renommer une avec un alias de colonne, ou n'en conserver qu'une.

Exercice 6

Considérons notre base `portables.db` dont le schéma relationnel est donné dans le graphique 1 page 4.

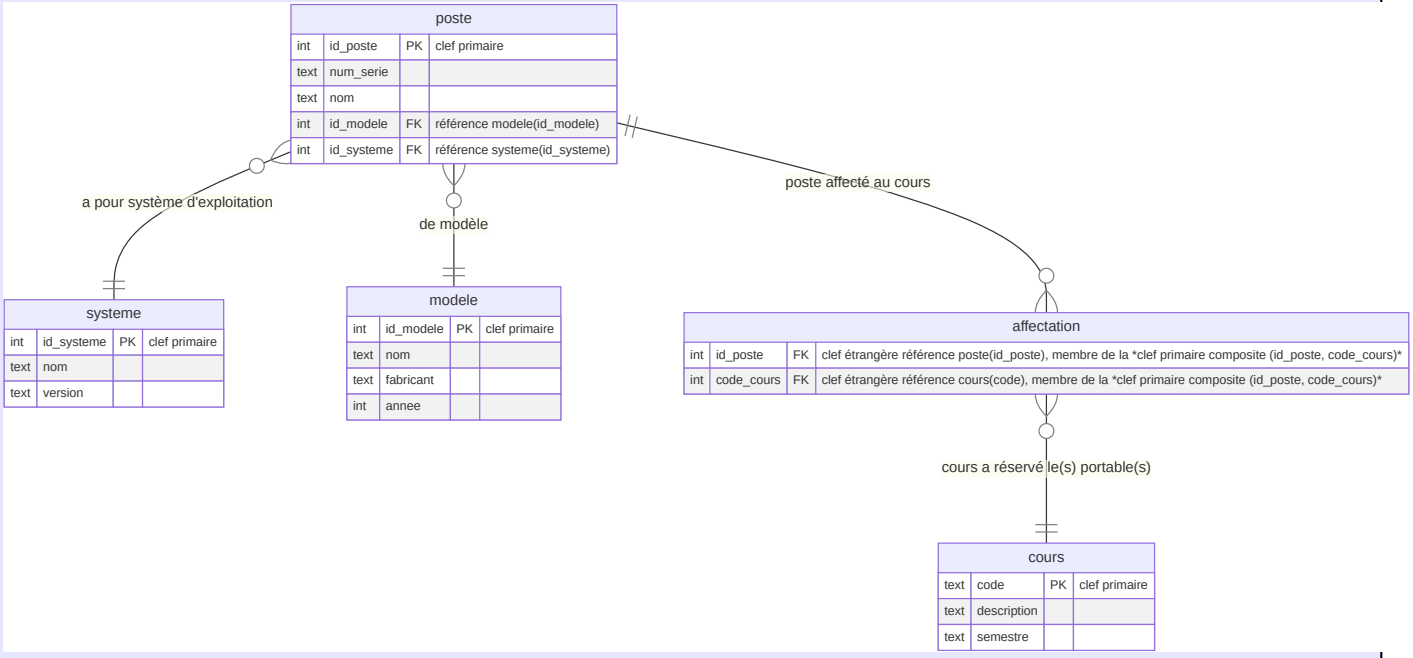
1. Écrire une requête SQL qui affiche une nouvelle table qui contient pour chaque poste son identifiant `id_poste`, son numéro de série, son nom, son modèle, son fabricant et l'année de livraison.
2. Modifier la requête précédente pour ne sélectionner dans le résultat de la jointure que les modèles de fabricant `'hp'`.
3. Modifier la requête précédente pour ordonner les résultats par ordre décroissant d'année de livraison.

3.3 Jointures multiples

Outil 2 *Base* `portables.db` *(2/2)*

On travaille désormais sur la version complète de la base `portables.db` avec deux nouvelles tables `affectation` et `cours` :

- ☞ chaque ligne de la table `cours` correspond à un cours de licence d'informatique avec son `code` (clef primaire), sa description et son semestre d'enseignement ;
- ☞ chaque ligne de la table `affectation` associe à un poste informatique de la table `poste` identifié par la clef étrangère `id_poste`, un cours identifié par la clef étrangère `code_cours`.



graphique 3

On rappelle le lien vers l'activité Capytale permettant de manipuler cette base : *https://capytale2.ac-paris.fr/web/c/7a2b-610499*.

Exercice 7 *Jointures multiples*

1. Déterminer la clef primaire et les clefs étrangères de la table `affectation`.
2. Que représente le résultat de la requête suivante ?

```
SELECT poste.nom, description, semestre
FROM poste JOIN affectation ON poste.id_poste=affectation.id_poste
    JOIN cours ON affectation.code_cours = cours.code ;
```

3. Écrire une requête dont le résultat est une table de colonnes `nom_poste, nom_modele,fabricant, nom_systeme, version` dont on donne un extrait. On renommera certaines colonnes avec `AS`.

nom_poste	nom_modele	fabricant	nom_systeme	version
Portable-01	hp probook 400	hp	fedora linux	36
PORTABLE-02	latitude e7470	dell	debian	bullseye
PORTABLE-03	hp probook 600	hp	ubuntu	18.04
PORTABLE-04	hp probook 400	hp	debian	buster
Portable-05	lenovo x270	lenovo	debian	buster
portable-06	hp probook 600	hp	debian	bullseye
PORTABLE-07	hp probook 600	hp	ubuntu	20.04

4. Écrire une requête dont le résultat est une table avec pour tous les postes affectés au cours `"UNIX"`, leur nom, celui de leur système d'exploitation et sa version.
5. Écrire une requête qui calcule, pour le cours `"Algorithmique et programmation récursive"`, le nombre de postes, leur âge moyen, l'âge minimal et l'âge maximal.
6. a. Écrire d'abord une requête qui détermine l'ensemble des postes qui ont été affectés au cours `"UNIX"` avec leur nom de poste et leur numéro de série.
   b. Écrire ensuite une requête qui détermine l'ensemble des postes qui ont été affectés au cours `"Algorithmique et programmation récursive"` avec leur nom de poste et leur numéro de série.
   c. Déduire des questions précédentes une requête qui détermine l'ensemble des postes qui ont été affectés à la fois au cours `"UNIX"` et au cours `"Algorithmique et programmation récursive"`, avec leur nom de poste et leur numéro de série.

   Lorsqu'une table intervient plusieurs fois dans une série de jointures, on utilise des alias de table avec la syntaxe `AS alias` comme pour les colonnes.
   Voir https://www.tutorialspoint.com/sqlite/sqlite_alias_syntax.htm

4 Requêtes d'insertion, de mise à jour, de suppression

4.1 Requête d'insertion

Méthode *Ajout de lignes*

Pour insérer une nouvelle ligne dans une table, on utilise la clause `INSERT` avec la syntaxe :

```
INSERT INTO table VALUES (val1, ..., valn) ;
```

Deux remarques importantes :

☞ *L'ordre des valeurs est celui des colonnes lors de la création de la table.*

☞ *L'insertion sera refusée par le SGBD si elle ne respecte pas les contraintes d'intégrité (relation, domaine, référentielle).*

Si on ne veut pas tenir compte de l'ordre, il faut expliciter le changement sur l'ordre des attributs. Par exemple, si on veut inverser l'ordre des attributs, on écrira :

```
INSERT INTO table(attributn, ..., attribut2, attribut1)
      VALUES   (valn,........., val2, val1) ;
```

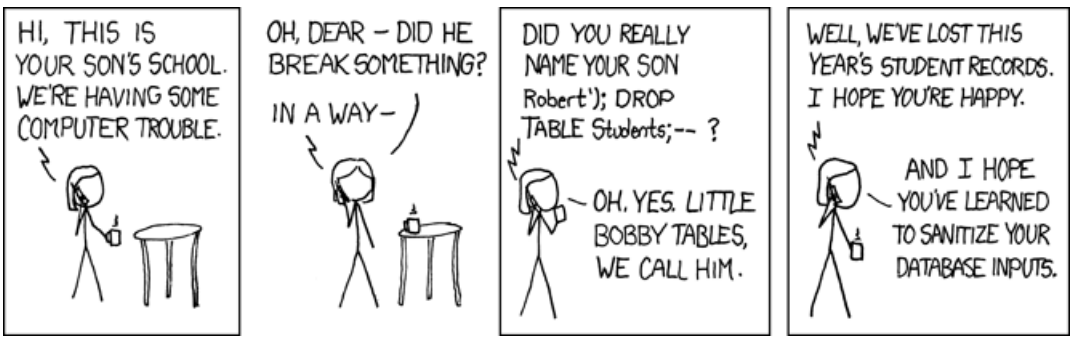
Exercice 8

On donne une liste de requêtes d'insertion dans la table `poste` de la base `portables.db`.

```
Entrée[10]: INSERT INTO poste VALUES(30, "ABCDE666", "PORTABLE-30", 6, 5) ;
            Error: UNIQUE constraint failed: poste.id_poste

Entrée[17]: INSERT INTO poste VALUES(31, "ABCDE666", "PORTABLE-31", 6, 7) ;
            Error: FOREIGN KEY constraint failed

Entrée[19]: INSERT INTO poste VALUES(31, "ABCDE666", "PORTABLE-31", 8, 5) ;
            Error: FOREIGN KEY constraint failed

Entrée[20]: INSERT INTO poste VALUES(31, "ABCDE666", "PORTABLE-31", 6, 5) ;

Entrée[21]: SELECT * FROM poste WHERE id_poste = 31 ;
Sortie[21]:
```

id_poste	num_serie	nom	id_modele	id_systeme
31	ABCDE666	PORTABLE-31	6	5

1. Expliquez pourquoi certaines requêtes échouent en inspectant la base sur Capytale.
2. À propos du problème de sécurité posé par l'injection de code SQL dans une requête d'insertion, résumer l'explication donnée sur https://explainxkcd.com/wiki/index.php/327:_Exploits_of_a_Mom de la planche 327 dessinée par XKCD.



4.2 Requête de suppression de ligne ou de table

Méthode

Pour supprimer un ensemble de lignes dans une table, on utilise la clause `DELETE` avec une condition de sélection pour spécifier la ou les lignes ciblée(s) :

```
DELETE FROM table WHERE condition ;
```

Pour supprimer une table de la base, on utilise la clause `DROP TABLE`.

```
DROP TABLE table;
```

Toute suppression doit laisser la base dans un état qui respecte les contraintes d'intégrité référentielle.

Exercice 9

1. Écrire une requête qui supprime les postes de la table `'poste'` dont le numéro de série contient un `'Z'`.
2. La requête `DELETE FROM systeme WHERE nom = 'ubuntu' AND version = '18.04' ;` est-elle acceptée par le SGBD ? Expliquez.
3. Écrire une requête qui supprime de la table `affectation` toutes les affectations de postes au cours `'UNIX'`.

   Il faudra utiliser une sous-requête : dans une condition de sélection :

   `WHERE affectation.code_cours = valeur` , valeur peut être le résultat d'une requête, placé entre parenthèses : `WHERE affectation.code_cours = (SELECT ..... )`
4. À partir du schéma relationnel de la base `portables.db`, page 16, déterminer quelle(s) table(s) pourraient-être supprimée(s) sans risque d'une violation de contrainte d'intégrité référentielle.

4.3 Requête de mise à jour

Méthode

Pour mettre à jour un attribut d'une table avec une nouvelle valeur pour un ensemble de lignes, on utilise la clause `UPDATE` avec une condition de sélection sur la ou les lignes ciblées :

```
UPDATE table SET attribut = valeur WHERE condition ;
```

Toute mise à jour doit laisser la base dans un état qui respecte les contraintes d'intégrité référentielle : on ne peut pas mettre à jour une valeur référencée par une clef étrangère dans un autre table.

Exercice 10

On reprend la base `portables.db`.

1. On a mis à jour tous les postes équipés du système `"ubuntu"` en version `"20.04"` avec la version `"22.04"`.

   Écrire une requête SQL qui enregistre cette mise à jour dans la base.

2. Expliquez pourquoi les requêtes ci-dessous échouent en inspectant la base sur Capytale.

```
Entrée[6]: UPDATE systeme SET id_systeme = 6 WHERE version = "18.04" ;
           Error: UNIQUE constraint failed: systeme.id_systeme

Entrée[5]: UPDATE systeme SET id_systeme = 7 WHERE version = "18.04" ;
           Error: FOREIGN KEY constraint failed
```

Table des matières