Informatique PT. 2024 - 2025

Mathématiques & Informatique - F. Gaunard
http://frederic.gaunard.com
Lycée Voltaire, Paris 11e.

5 Bases de données et requêtes SQL

Dans ce cours, nous allons parler de systèmes informatiques gérant des données structurées mais non hiérarchisées, en l'occurrence les bases de données (BDD). Les exemples sont innombrables : gestion des stocks (magasins, bibliothèques,...), gestion des réservations (trains, avions, spectacles,...), ou création et interrogation de listes diverses (vos films préférés...)

1 Introduction : Architecture et langage

Considérons, d'un côté, un serveur de données avec des disques durs contenant des données et de l'autre, un utilisateur. Entre les deux se trouve une application (appelée **Système de Gestion de Base de Données** ou **SGBD**) qui traduit les demandes de l'utilisateur, dans un langage spécifique (le langage standard est SQL pour *Structured Query Langage*).

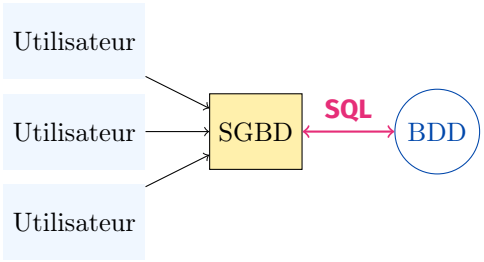

FIGURE 1.1. L'architecture client/serveur dans le cas d'une base de données

L'architecture ci-dessus est appelée *client/serveur*, car la base de données est gérée sur un serveur, et l'application tourne sur une autre machine, le client. Cette architecture contient un troisième *niveau* (ou *tiers*) : une machine qui gère l'interface, comme un navigateur Web sur un ordinateur. On parle alors d'architecture *trois-tiers* :

- ✗ le tiers utilisateur ;
- ✗ le tiers applicatif ;
- ✗ le tiers base de données.

Dans les faits, un même serveur de bases de données est souvent commun à une multitude d'architectures trois-tiers. La difficulté de conception d'un tel système vient du fait que plusieurs utilisateurs peuvent accéder simultanément à la base de données (des dizaines de milliers de personne peuvent consulter simultanément le catalogue d'un magasin en ligne). Ainsi, un utilisateur peut modifier des données existantes pendant qu'un second effectue une recherche. Une telle utilisation nécessite cependant de veiller à l'ordre dans lequel sont traitées les demandes.

Le langage SQL comporte des instructions relatives :

- ✗ à l'interrogation de bases de données ;
- ✗ à la création et la modification des données ;
- ✗ au contrôle d'accès des données.

☞ Dans ce cours, nous nous intéresserons uniquement aux instructions de la première catégorie.

Le langage SQL a pour vocation d'être utilisé par des non spécialistes : l'utilisateur pourra y écrire une requête, sans se préoccuper de la façon de coder celle-ci de façon optimale, c'est le gestionnaire de bases de données qui assume cette tâche.

Les systèmes de gestion de bases de données sont des logiciels complexes, résultats de dizaines d'années de recherche et de développement. Ils permettent à des individus ou des programmes d'exprimer des requêtes pour interroger ou modifier des bases de données. Nous nous focaliserons ici sur les plus répandus d'entre ces systèmes, l*es systèmes relationnels*, parmi lesquels nous trouvons des logiciels très répandus comme SQLite ou MySQL.

2 Structure d'une base de données et vocabulaire

2.1 Tables, attributs et enregistrements

Les bases de données sont décrites par ce que l'on appelle *le modèle relationnel*, dans lequel les données sont organisées en tableaux à deux dimensions que nous appellerons des *relations* ou *tables*. Chaque base de données est ainsi constituée de différentes tables ou relations.

Définition 1. **Relation**

Une **relation** est un tableau de données (penser à un fichier Excel). Chaque ligne représente un objet et tous les objets sont de même nature.
Une **base de donnée** (parfois abrégée en BDD) est un ensemble de relations.

Les en-têtes de chaque colonne sont appelés des *attributs* ; à chaque colonne est associé un *domaine*, qui correspond au type de données utilisé pour enregistrer l'attribut.

Définition 2. **Vocabulaire relationnel**

On a les définitions suivantes :

- ✗ Une *relation* est aussi appelée une **table**.
- ✗ Les termes *ligne*, *n-uplet*, *enregistrement* ou *vecteur* sont tous synonymes.
- ✗ De même les colonnes sont souvent appelées des **attributs**.
- ✗ Le **schéma** est l'ensemble des attributs d'une relation.
- ✗ Le *domaine* désigne l'ensemble des valeurs que peuvent prendre un attribut.

Exemple 1.

Construisons la table de relation des étudiants de cette prépa, en indiquant comme caractéristiques les dates de naissance, la ville de résidence, leur profil et l'université de rattachement en inscription cumulative.

Etudiants de Voltaire				
Identifiant	**Date de naissance**	**Ville de residence**	**Profil**	**Universite**
1	21.06.2006	Paris 11	3/2	Sorbonne
2	17.11.2005	Saint Maur	5/3	Sorbonne
3	24.03.2006	Bagnolet	3/2	Paris Cité
4	11.10.2005	Vincennes	5/2	Descartes
5	03.01.2005	Paris 20	5/2	Sorbonne
6	03.01.2006	Bondy	3/2	Sorbonne
...	...	...	...	...

Sur cet exemple, `Identifiant` est un attribut de la table `Etudiants de Voltaire`, dont le domaine est l'ensemble des nombres entiers, `Ville de residence` est un autre attribut dont le domaine est l'ensemble des chaînes de caractères et `Date de naissance` est un attribut de la même table dont le domaine est l'ensemble des chaînes de caractères au format date JJ.MM.AA.

`Etudiants de Voltaire` est une relation de schéma

{`identifiant, date de naissance, ville de residence, profil, universite` }.

☞ Dans une relation, l'ordre des lignes n'a pas d'importance.

2.2 Clé primaire, clé étrangère

Reprenons l'exemple précédent des étudiants de notre prépa. Il est tout à fait possible que deux étudiants aient exactement la même date de naissance et étudient les mêmes langues. Dans une table, il est très important d'identifier de **manière unique** chaque ligne. C'est pour résoudre ce problème d'identification qu'interviennent les clés.

Définition 3. **Clé**

Une **clé** est un groupe minimum d'attributs caractérisant chaque enregistrement de manière unique.

Exercice 1.

On considère la relation **conserves** décrite par la table ci-dessous.

code barre	produit	marque	prix unitaire	quantité	peremption
3083680020763	Maïs	Budget+	1,19	12	2026
3017800207901	Maïs	GourmetBio	2,69	9	2025
3038359008504	Tomates	PizzaLovers	2,99	6	2025
303087439871	Haricots Verts	Budget+	1,89	10	2025
3093439818758	Pêches	GourmetBio	2,99	6	2030
3023249825326	Pois chiches	Budget+	0,99	4	2026

1. Le groupe d'attributs {**code barre**} est-il une clé pour notre relation ?
2. Le groupe d'attributs {**code barre**, **produit**} est-il une clé pour notre relation ?
3. Le groupe d'attributs {**produit**, **marque**, **prix**} est-il une clé pour notre relation ?
4. Le groupe d'attributs {**produit**} est-il une clé pour notre relation ?
5. Le groupe d'attributs {**marque**, **prix**} est-il une clé pour notre relation ?

Définition 4. **Clé Primaire ou Primary Key**

Dans une relation, les groupes d'attributs qui constituent des clés sont appelés des *clés candidates*. Dans la pratique, il sera indispensable d'en choisir une et nous l'appellerons la **clé primaire**.
En anglais, clé primaire se dit Primary Key et s'abrège en **PK**.

Mais une clé primaire ne correspond pas nécessairement à un unique attribut : cela peut être un ensemble de plusieurs attributs.

Remarque 1.

Le choix d'une clé primaire pourrait très bien être arbitraire (parmi les clés candidates). Cependant, on peut prendre en compte certains critères. En effet, une clé primaire est généralement choisie de façon à ce qu'elle soit simple, c'est-à-dire qu'elle ne contienne le moins d'attributs possibles.

De plus, on préfère généralement les attributs "basiques" (par exemple des entiers ou des chaînes de caractères courtes).

☞ Parfois aucune des clés candidates ne contient un nombre raisonnable d'attributs et ne semble "simple". On peut alors fabriquer une clé simple :

Définition 5.

Une **clé artificielle** est un attribut que l'on ajoute à la relation. Cet attribut n'a pas de réelle signification dans le domaine que l'on modélise et sa seule fonction est d'identifier de manière unique les n-uplets de la relation.

☞ Pour des questions de performance (en termes de temps de calcul), les clés non artificielles ne sont souvent pas optimisées lorsqu'il est demandé au SGBDD de retrouver une ligne dans une table.

☞ Un bon programmeur prend en fait l'habitude d'utiliser comme clés primaires des clés artificielles.

Une base de données est en fait bien souvent constituée de plusieurs tables ; et ces tables ne sont pas indépendantes.

Dans notre exemple fil rouge de la base de données des étudiants de notre prépa, nous pouvons nous intéresser à la ville de provenance des étudiants et renseigner dans une relation séparée certaines caractéristiques des villes qui y décrivent

l'ambiance de travail. Même si cette seconde table est d'un intérêt indépendant, ces deux relations sont liées car chaque étudiant provient d'une des villes décrites dans le seconde table.

Villes autour de Voltaire			
Ville (PK)	**Temps de trajet max**	**Avantages**	**Inconvénients**
Paris 11	5	proche	cher
Bagnolet	15	bon marché	pas de bibliothèque
Saint-Maur	75	calme et agréable	loin
Vincennes	35	sport au Bois	super cher
Bondy	50	possib. de res. univ.	pas facile pour le sport
Paris 20	12	pas cher, pas loin et vivant	Bruyant
...	...	...	...

Ainsi, si on veut connaître le temps de trajet quotidien d'un étudiant, on doit déjà connaître sa ville de provenance, puis regarder dans la table des villes, quel est le temps de trajet pour venir à Voltaire.

☞ **Retrouver une ligne dans une table, c'est justement le rôle des clés !**

Définition 6. **Clé étrangère ou Foreign Key**

Un attribut d'une table qui est une clé pour une autre table de la base de données s'appelle une **clé étrangère**. En anglais, clé étrangère se dit Foreign Key et s'abrège en **FK**.

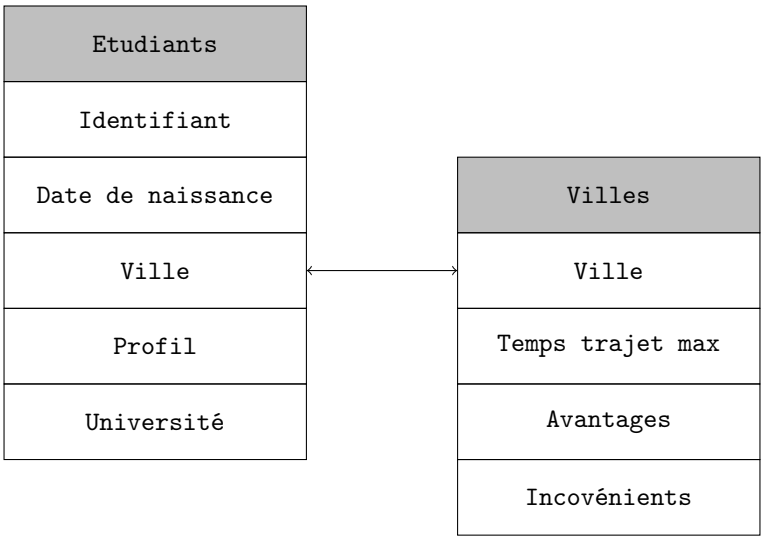

FIGURE 2.1. Schéma de la base de données

On dira que l'attribut `Ville de residence` est une clé étrangère pour la table `Etudiants de Voltaire`, car il correspond à la clé primaire d'une autre table (en l'occurrence `Ville`). Le schéma d'une base de données fait donc apparaître les tables, les attributs, ainsi que les liens existants entre clés primaires et clés étrangères.

2.3 Les tables d'association

Considérons la situation suivante. On dispose de la relation des **écoles** ci-dessous

Ecoles		
Identifiant (PK)	**Nom de l'école**	**Rang**
1	ENS	1
2	Polytechnique	2
3	Mines Paris	3
4	CentraleSupélec	4
5	Ponts ParisTech	5
6	SupAero Toulouse	6
7	Telecom Paris	6 ex-aequo
...	...	...

(Observons que nous avons pris pour **PK** une clé artificielle, le rang n'étant pas une bonne PK car il peut y avoir des ax-aequo.)

C'est maintenant la fin de l'année dans notre prépa et nous voulons ajouter à notre base de données les informations sur les différentes admissibilités de nos étudiants dans les différentes écoles. Nous pouvons donc ajouter une colonne dans notre table `Etudiants de Voltaire` avec l'attribut **admissible à** qui est une FK vers la table **Ecoles**.

Et si un étudiant est admissible dans 2 écoles, comment fait on ?

On pourrait mettre deux colonnes avec les attributs **admissible à 1** et **admissible à 2** qui seraient toutes les deux des clés étrangères vers la table **écoles**. Mais ce raisonnement n'est pas très bon : on ne peut pas savoir à l'avance dans combien d'écoles un étudiant va être admissible.

La stratégie inverse ne fonctionne pas non plus. On pourrait envisager de mettre dans la table **écoles** un attribut **étudiant admissible** qui serait une clé étrangère vers la table **étudiants**. Mais cela voudrait dire qu'une école ne peut sélectionner qu'un seul étudiant admissible !

☞ Ce problème motive la définition ci-dessous.

Définition 7. **Table d'association**

Une **table d'association** est une relation qui contient comme attributs au moins deux clés étrangères vers d'autres relations de la base de données.

Exemple 2. **Table d'association**

Construisons donc la table d'association des étudiants et des écoles dans lesquelles ils sont admissibles.
Nous pouvons renseigner efficacement les différentes admissibilités des étudiants de la prépa, sans nous limiter, ni en nombre d'écoles par étudiant, ni en nombre d'étudiants admissibles par école.

Admissibilites	
Étudiant	**École**
1	1
1	2
1	4
2	4
2	5
...	...

☞ Chacun des attributs de cette table est FK mais la PK est constituée des deux attributs {**Étudiant, École**}.

Remarque 2.

Rien ne nous empêche d'ajouter des attributs dans notre table d'association (par exemple la date de passage pour les oraux dans la situation précédente) : tous les attributs de la table d'association ne sont donc pas nécessairement des clés étrangères pour une relation de la base de données.

Cela pose d'ailleurs la question du choix de la clé primaire dans une table d'association... mais c'est une autre histoire.

3 Algèbre relationnelle

Dans la partie précédente, nous avons vu comment représenter des données. Nous passons maintenant à la partie manipulation. Comme convenu au début de ce cours, nous décrivons tout d'abord de manière abstraite les différentes opérations que l'on peut appliquer à une base de données, puis nous verrons comment cela se concrétise dans la section suivante.

3.1 Projection et Restriction

Définition 8. **Projection**

La **projection** consiste à sélectionner les attributs que l'on souhaite et éliminer les autres.

Exemple 3.

Si l'on s'intéresse uniquement aux temps de trajet, on peut faire une projection pour ne conserver que l'attribut **Temps de trajet max** et on obtient alors la table suivante.

Villes autour de Voltaire	
Ville (PK)	**Temps de trajet max**
Paris 11	5
Bagnolet	15
Saint-Maur	75
Vincennes	35
Bondy	50
Paris 20	12
...	...

Nous voudrions maintenant faire la même opération mais sur les lignes, c'est-à-dire ne conserver que certaines des lignes de notre relation. C'est légèrement plus compliqué car les colonnes de notre table portent un nom mais pas les lignes. Les lignes d'une base de données sont amenées à varier car les informations contenues dans la table varient au cours du temps.

Comme les lignes à garder ou à enlever sont *dynamiques*, il nous faut donc une condition qui nous permette de **restreindre les lignes.**

Définition 9. **Restriction**

Une **restriction** dans une table de données est une condition binaire (de type vrai ou faux) qui porte sur un ou plusieurs des attributs de la relation.
Une restriction s'appelle aussi parfois une *Sélection*.

Exemple 4.

La restriction de la table `Etudiants de Voltaire` sous la condition `Universite = 'Sorbonne'` produit la table suivante:

Etudiants de Voltaire				
Identifiant	**Date de naissance**	**Ville**	**Profil**	**Universite**
1	21.06.2006	Paris 11	3/2	Sorbonne
2	17.11.2005	Saint Maur	5/3	Sorbonne
5	03.01.2005	Paris 20	5/2	Sorbonne
6	03.01.2006	Bondy	3/2	Sorbonne
...	...	...	...	...

Exercice 2.

Quelle table obtient-on à partir de la table **conserves** de l'**Exercice 1** après restriction sous la condition

```
peremption <2026
```

3.2 Union, Différence, Intersection

Définition 10. **Union de relations**

L'**union** de deux relations R_1 et R_2 de même schéma est une nouvelle relation, également de même schéma, qui contient l'ensemble des lignes de R_1 et R_2

☞ Attention, dans la pratique avec le langage SQL, l'union de deux tables qui contiennent une ligne en commun produira une table avec la même ligne répétée. Mais ce n'est pas très grave, il existe une commande pour éliminer les doublons d'une table.

Définition 11. **Différences de relations**

La **différence** de deux relations R_1 et R_2 de même schéma donne une relation R_3 de même schéma qui contient toutes les lignes de R_1 qui ne sont pas dans R_2.

Définition 12. **Intersection de relations**

L'**intersection** de deux relations R_1 et R_2 de même schéma donne une relation R_3 de même schéma qui contient toutes les lignes qui sont à la fois dans R_1 et dans R_2.

3.3 Produit cartésien

Au conseil de classe, chaque professeur doit donner une appréciation sur chaque étudiant de la prépa. Pour cela, nous disposons d'une relation qui contient les informations sur les professeurs de la classe (pour simplifier, nous n'entrerons pas dans le détail du choix des options), et (une projection et restriction du) tableau des étudiants.

identifiant prof	Nom du professeur
prof1	M. Boucher
prof2	M. Chabot
prof3	M. Gaunard
prof4	Mme Pichon
prof5	M. Robert

identifiant étudiant
1
2
3

Pour pouvoir indiquer toutes les appréciations dans une relation, nous devons former le *produit cartésien* des deux tables des enseignants et des étudiants.

Définition 13. **Produit cartésien de deux relations**

Le **produit cartésien** de deux relations R_1 et R_2 est une table qui contient toutes les combinaisons possibles des lignes de R_1 et des lignes de R_2 et qui contient les colonnes de R_1 ainsi que les colonnes de R_2.

Exemple 5.

Dans l'exemple entamé en début de sous-section, le produit cartésien des relations professeurs et étudiants renvoie :

identifiant prof	Nom du professeur	identifiant étudiant
prof1	M. Boucher	1
prof1	M. Boucher	2
prof1	M. Boucher	3
prof2	M. Chabot	1
prof2	M. Chabot	2
prof2	M. Chabot	3
prof3	M. Gaunard	1
prof3	M. Gaunard	2
prof3	M. Gaunard	3
prof4	Mme Pichon	1
prof4	Mme Pichon	2
prof4	Mme Pichon	3
prof5	M. Robert	1
prof5	M. Robert	2
prof5	M. Robert	3

à laquelle il est ensuite facile d'ajouter un attribut **Appreciation**.

3.4 Jointure

La *jointure* est le concept fondamental de l'algèbre relationnelle. Jusqu'à présent, nous avons vu que les clés étrangères (FK) servent à lier des relations. Mais nous ne savons pas encore comment exploiter ces liaisons. C'est justement le but de l'opération de jointure.

Si nous voulons connaître le temps de trajet pour venir à Voltaire d'un étudiant de la prépa, nous devons regarder d'abord dans la table des étudiants sa ville de provenance, puis aller chercher le temps de trajet dans la table des villes voisines. La jointure sert à coller les deux tables pour faire apparaître l'information sur une seule table. Plus précisément,

Définition 14. **Jointure**

On considère deux relations R_1 et R_2 et on suppose que dans la relation R_1, le groupe d'attributs G est une clé externe (FK) pour la relation R_2. La **jointure** de R_1 et R_2 selon G est la table qui contient le même nombre de lignes que R_1 et les attributs de R_1 et de R_2. Chaque ligne est construite en commençant par la ligne de R_1, puis en ajoutant à sa suite la ligne de R_2 caractérisée par la valeur de sa clé dans G.
Le groupe d'attributs G s'appelle **la condition de jointure**.

Exemple 6.

Reprenons notre relation préférée **étudiants**

Etudiants				
Identifiant (PK)	**Date de naissance**	**Ville de residence (FK)**	**Profil**	**Universite**
1	21.06.2006	Paris 11	3/2	Sorbonne
2	17.11.2005	Saint Maur	5/3	Sorbonne
3	24.03.2006	Bagnolet	3/2	Paris Cité
4	11.10.2005	Vincennes	5/2	Descartes
5	03.01.2005	Paris 20	5/2	Sorbonne
6	03.01.2006	Bondy	3/2	Sorbonne
...	...	...	...	...

et une relation **Villes** (obtenue après une projection d'une relation ci-avant):

Villes	
Ville (PK)	**Temps de trajet max**
Paris 11	5
Bagnolet	15
Saint-Maur	75
Vincennes	35
Bondy	50
Paris 20	12
...	...

La jointure de ces deux relations selon la condition `Ville de residence = Ville` donne la table suivante

Etudiants. Identifiant	**Etudiants. Date de naissance**	**Etudiants. Ville de residence**	**Etudiants. Profil**	**Etudiants. Universite**	**Villes. Ville**	**Villes. Temps de trajet max**
1	21.06.2006	Paris 11	3/2	Sorbonne	Paris 11	5
2	17.11.2005	Saint Maur	5/3	Sorbonne	Saint Maur	75
3	24.03.2006	Bagnolet	3/2	Paris Cité	Bagnolet	15
4	11.10.2005	Vincennes	5/2	Descartes	Vincennes	35
5	03.01.2005	Paris 20	5/2	Sorbonne	Paris 20	12
6	03.01.2006	Bondy	3/2	Sorbonne	Bondy	50

3.5 Agrégation

L'*agrégation* (qui n'est en fait pas une opération de l'algèbre relationnelle) est utilisée lorsqu'on veut faire un calcul qui porte sur plusieurs ligne d'une table. Elle sert par exemple à répondre à la question *Quel est le temps de trajet moyen des*

étudiants de chaque profil ?

L'agrégation est donc une opération en deux étapes : une étape de *partitionnement* et une étape de calcul où on applique une fonction à chacun des blocs de la partition.

Définition 15. **Agrégat**

On considère une table et un groupe d'attributs.
Un **agrégat** est une partition des lignes d'une table selon les différentes valeurs que peuvent prendre les attributs.
Dans cette situation, on dit que les attributs sont des **attributs de partitionnement**.

Une fois les agrégats formés, on leur applique une fonction d'agrégation.

Définition 16. **Fonction d'agrégation**

Une **fonction d'agrégation** est une fonction définie sur les éléments de la partition (elle rend une unique valeur pour chaque bloc de la partition).

☞ Les exemples classiques de fonctions d'agrégation que nous manipulerons dans la pratique sont les fonctions de *décompte* (compter le nombre d'éléments d'un bloc), des fonctions de *moyenne*, de *minimum*, de *maximum*.

4 Commandes et requêtes SQL

Une fois les données représentées sous forme relationnelle, il faut pouvoir les exploiter pour y rechercher des informations précises. Pour cela, on doit tout d'abord être capable d'exprimer ces recherches sous une forme précise. On va donner ici un aperçu de ce qui sera formellement défini plus tard : le langage de requêtes SQL.

Considérons la requête SQL suivante :

```
SELECT Identifiant FROM Etudiants WHERE Profil="5/2"
```

Cette requête correspond à une traduction en langage SQL de la demande suivante : "obtenir les identifiants des étudiants de la classe qui sont des 5/2".

Cette requête peut se décomposer en les opérations élémentaires suivantes :

- ✗ **sélectionner** dans la relation `Etudiants` les valeurs dont l'attribut `Profil` correspond à la valeur (en tant que chaine de caractère ici) '5/2' ;
- ✗ regrouper ces deux ensembles de valeurs ;
- ✗ extraire de chacune des valeurs l'attribut `Nom`.

4.1 Les commandes correspondant aux opérations relationnelles

Projection

Pour **projeter** une table `Table` selon des attributs $(A1, A2, \ldots, An)$ de cette table, on utilise la syntaxe suivante :

```
SELECT A1,A2,...,An FROM Table
```

☞ Il peut exister plusieurs enregistrements ayant les mêmes valeurs sur tous les attributs sur lesquels on projette, suite à quoi on aura deux enregistrements identiques après la projection. Si on souhaite supprimer tous les doublons, il suffit d'utiliser la commande

```
SELECT DISTINCT
```

Restriction

La restriction permet d'extraire d'une table `Table` les seuls enregistrements respectant une condition `Condition`, en utilisant la syntaxe suivante :

```
SELECT * FROM Table WHERE Condition
```

Remarque 3.

Attention aux faux-amis : le mot-clé `SELECT` permet de faire une projection, et non une restriction (sélection). La restriction s'effectue grâce au mot clé `WHERE`.
C'est notamment pour ça qu'on lui préfère la terminologie Restriction.

☞ `Condition` est exprimée à l'aide :

- ✗ d'opérateurs logiques : `AND, OR, NOT`
- ✗ de comparateurs de chaîne : `IN, BETWEEN, LIKE`
- ✗ d'opérateurs arithmétiques : `+, -, *, /, %, ^`
- ✗ et de comparateurs arithmétiques : `=, !=, >, <, >=, <=, <>`.

Exemple 7.

Il est possible de comparer plusieurs attributs dans une sélection ou d'effectuer des opérations sur ceux-ci lorsque les données sont de type entier ou flottant. Considérons la table suivante :

Stock				
Ref	Achetes	Vendus	Jetes	EnStock
12530	50	47	1	2
28596	50	15	0	34
35841	50	39	5	6
47123	100	99	1	0
50239	70	60	3	7

La requête suivante, qui utilise l'opérateur `<>` qui teste la différence, permet de détecter une erreur de saisie.

```
SELECT * FROM Stock WHERE Achetes - Vendus - Jetes - EnStock <> 0
```

Ref	Achetes	Vendus	Jetes	EnStock
28596	50	15	0	34

Par ailleurs, la commande `ORDER BY` permet de trier les lignes dans le résultat d'une requête. Il est ainsi possible de trier les données en fonction d'un ou plusieurs attributs, soit par ordre croissant (en ajoutant `ASC` pour ascendant), soit par ordre décroissant (avec `DESC` pour descendant).

Exemple 8.

Par exemple, la requête

```
SELECT * FROM Stock ORDER BY EnStock DESC
```

renvoie

Ref	Achetes	Vendus	Jetes	EnStock
28596	50	15	0	34
50239	70	60	3	7
35841	50	39	5	6
12530	50	47	1	2
47123	100	99	1	0

Enfin, lorsque l'on utilise `ORDER BY`, les commandes `LIMIT` et `OFFSET` permettent d'afficher seulement une portion des lignes générées par le reste de la requête :

- ✗ `LIMIT` indique le nombre maximal de lignes qui sera renvoyé (et la machine renverra les premières du classement) ;
- ✗ `OFFSET` indique le nombre de lignes à omettre avant de renvoyer les lignes restantes.

Exemple 9.

Par exemple, la requête

```
SELECT * FROM Stock ORDER BY EnStock DESC LIMIT 3
```

renvoie

Ref	Achetes	Vendus	Jetes	EnStock
28596	50	15	0	34
50239	70	60	3	7
35841	50	39	5	6

et la requête

```
SELECT * FROM Stock ORDER BY EnStock DESC LIMIT 3 OFFSET 2
```

renvoie

Ref	Achetes	Vendus	Jetes	EnStock
35841	50	39	5	6
12530	50	47	1	2
47123	100	99	1	0

☞ Plus généralement, si R est une relation, $A1, A2$ des attributs de celle-ci et n, p des entiers, la requête suivante :

```
SELECT * FROM R ORDER BY A1 ASC, A2 DESC LIMIT n OFFSET p
```

va trier les lignes de la table, d'abord en fonction de la colonne $A1$ de façon croissante, puis de la colonne $A2$ de façon décroissante, et renverra n lignes en commençant par la $(p+1)$-ème du classement.

Fonctions d'agrégation

Le langage SQL permet d'effectuer certains calculs directement sur les données numériques d'une base de données. On dispose des cinq fonctions *d'agrégation* suivantes :

- ✗ `COUNT` : renvoie le nombre de valeurs.
- ✗ `MAX` : renvoie la valeur maximale.
- ✗ `MIN` : renvoie la valeur minimale.
- ✗ `SUM` : renvoie la somme des valeurs.
- ✗ `AVG` : renvoie la valeur moyenne (*average* en anglais).

Pour calculer la fonction d'agrégation f sur un attribut A d'une relation `R`, on utilise la requête suivante :

```
SELECT f(A) FROM R
```

Exemple 10.

Considérons par exemple la relation suivante :

DSInfo1		
Classe	Eleve	Note
PT2	Alice	14
PT1	Bob	3
PT1	Charles	11
PT2	David	7
PT2	Nadera	10
PT1	Florian	16
PT2	Gael	18

☞ Pour calculer la moyenne sur les deux classes, on utilise la requête suivante :

```
SELECT AVG(Note) FROM DSInfo1
```

On obtient alors le résultat :

AVG(Note)
11.2857142857143

Il est aussi possible de coupler ce calcul à une sélection :

```
SELECT AVG(Note) FROM DSInfo1 WHERE Classe="PT1"
```

AVG(Note)
10.0

Le résultat renvoyé par l'agrégation étant de type numérique, il est possible de l'utiliser dans des comparaisons.

☞ Pour obtenir les élèves ayant obtenu plus que la moyenne de la classe, on écrit :

```
SELECT * FROM DSInfo1 WHERE Note>= (SELECT AVG(Note) FROM DSInfo1)
```

Classe	Eleve	Note
PT2	Alice	14
PT1	Florian	16
PT2	Gael	18

Plus généralement, il est possible de regrouper les données avant de leur appliquer une fonction d'agrégation. Considérons une relation R, des attributs $A1, \dots, An, B_1, \dots, Bk$ de celle-ci et des fonctions d'agrégation $f1, \dots, fk$. La requête suivante :

```
SELECT A1, ..., An, f1(B1), ..., fk(Bk) FROM R GROUP BY A1,\ldots,An
```

permet de regrouper les valeurs de R ayant les mêmes attributs $A1, \dots, An$ et d'appliquer les fonctions d'agrégation $f1, \dots, fk$ sur les attributs $B1, \dots, Bk$.

Exemple 11.

Par exemple, pour calculer les notes minimale, maximale et moyenne par classe, on peut effectuer la requête suivante :

```
SELECT Classe,MIN(Note),MAX(Note),AVG(Note) FROM DSInfo1 GROUP BY Classe
```

Classe	MIN(Note)	MAX(Note)	AVG(Note)
PT1	3	16	10
PT2	7	18	12.25

On peut enfin coupler ce calcul à :

- ✗ une projection ;
- ✗ une restriction en amont, en utilisant une condition `WHERE` avant l'instruction `GROUP BY`;
- ✗ une restriction en aval. Pour cela, il faut utiliser une syntaxe spécifique de condition en utilisant l'instruction `HAVING` après l'instruction `GROUP BY`.

La syntaxe la plus générale est donc la suivante, où $C1$ et $C2$ sont deux conditions :

```
SELECT A1, ..., An, f1(B1), ... , fk(Bk) FROM R GROUP BY A1,... ,An HAVING C2
```

Exemple 12.

La requête

```
SELECT Classe,MIN(Note),MAX(Note) FROM DSInfo1 WHERE (Note>8 AND Note<17)
GROUP BY Classe HAVING MAX(Note)-MIN(Note)<=4
```

renvoie

Classe	MIN(Note)	MAX(Note)
PT2	10	14

Remarque 4.

Dans la mesure du possible, il faut toujours réaliser les restrictions en amont à l'aide d'une instruction `WHERE`, car cela limite le nombre de valeurs à considérer dans l'agrégation. Cependant, lorsque l'on désire sélectionner en fonction du résultat des fonctions d'agrégation, il est obligatoire de le faire en aval à l'aide d'une instruction `HAVING`.

Renommage

Il est possible de renommer un attribut. Pour cela, on ajoute lors de la projection le mot-clé `AS`. Par exemple, si R est une relation de schéma $(A1, \dots, An)$ où l'on souhaite renommer $A1$ en B, on effectue :

```
SELECT A1 AS B, A2 , ... , An FROM R
```

Produit Cartésien

Pour effectuer le produit cartésien de deux tables `Table1` et `Table2`, on utilise la syntaxe suivante :

```
SELECT * FROM Table1,Table2
```

Jointure

Pour effectuer la jointure symétrique de deux tables $R1$ et $R2$ selon les attributs (A, B), on utilise la syntaxe SQL suivante :

```
SELECT * FROM R1 JOIN R2 ON A=B
```

Attention. Dans le cas où on souhaite recoller deux tables selon des attributs qui portent le même nom, il faut préciser le nom de table de chaque attribut dans la requête `SQL` afin d'éviter toute confusion. Ainsi, si A est un attribut des tables R et R', pour calculer la jointure symétrique de R et R' selon (A, A), on utilise la syntaxe suivante :

```
SELECT * FROM R1 JOIN R2 ON R1.A=R2.B
```

Exemple 13.

Dans notre exemple fil rouge, la jointure de la table Etudiants et Villes selon la condition `Ville de residence = Ville` s'écrit :

```
SELECT * FROM Etudiants JOIN Villes ON Etudiants.Ville=Villes.Ville
```

Remarque 5.

La présence des deux attributs A et A' est redondante après jointure. On peut éliminer cette redondance en effectuant une projection.

Remarque 6.

La jointure peut également se faire à l'aide d'un produit cartésien, avec la syntaxe

```
SELECT * FROM R1, R2 WHERE A=B
```

Il serait possible de ne jamais utiliser opérateur de jointure, puisqu'il s'exprime comme une composition d'un produit cartésien et d'une sélection. Concrètement, ce serait une très mauvaise idée de programmer les jointures par le biais de ces deux autres opérations : si les relations $R1$ et $R2$ contiennent respectivement n_1 et n_2 valeurs, le produit cartésien $R1 \times R2$ construit une relation de $n \times n_2$ valeurs, qu'il faut ensuite parcourir pour effectuer la sélection, d'où un coût quadratique. Avec les tailles courantes des bases de données, un tel coût est impraticable et, de plus, la relation $R1 \times R2$ a peu de chances de tenir dans la mémoire vive disponible. Les gestionnaires de bases de données disposent d'algorithmes efficaces pour effectuer la jointure de deux tables de taille n avec une complexité en $O(n \ln n)$.

4.2 Remarques générales sur les requêtes SQL

Il est bon de retenir les remarques suivantes :

- ✗ L'optimisation des requêtes est laissée à la charge du gestionnaire de bases de données.
- ✗ Les requêtes SQL ne sont pas sensibles à la casse, mais il est d'usage de mettre les mots-clés en majuscules et les attributs en minuscules. Les valeurs des attributs non numériques sont écrites entre guillemets simples ou doubles.
- ✗ Toutes les requêtes de recherche commencent par le mot-clé `SELECT`, qui effectuera une projection en fin de requête.
- ✗ La relation sur laquelle on opère est précisée par `FROM`.
- ✗ Les requêtes SQL se terminent systématiquement par un point-virgule, mais dans la plupart des gestionnaires de bases de données ceux-ci ne sont pas indispensables.

Exercice 3.

Une entreprise de vente de calendriers en ligne dispose d'une base de données concernant son activité, notamment une table `Clients` et une table `Commandes`.

Clients				
Id	Prenom	Nom	eMail	Ville
1	Marion	Macheral	marion69@caramail.com	Lyon
2	Emmanuel	Moncra	allezOM@gmail.com	Paris
3	Jean-Luc	Mechenlon	JLpresident@wanadoo.fr	Marseille
4	Philippe	Pata	fifi33@gmx.fr	Bordeaux
5	Jordan	Berdalle	jordyBG@outlook.com	Strasbourg

Commandes			
Utilisateur_id	date_achat	num_facture	prix_total
5	15-05-2018	FR18-52	4.39
1	21-06-2018	FR18-59	5.24
4	17-11-2018	FR18-85	11.20
3	03-01-2019	FR19-01	14.50
3	24-03-2019	FR19-22	3.14

Que renvoie la commande ci-dessous ?

```
SELECT Id, Prenom, Nom, date_achat, num_facture, prix_total
FROM Clients
INNER JOIN Commande ON Clients.Id = Commandes.Utilisateur_id
```

5 Manipuler SQL sur sa machine

☞ **[SQLite]** Pour communiquer (et modifier) avec une base de données, on utilise le langage SQL (*Structured Query Language*). Pour 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 trouve ça un peu austère. On a donc décidé d'utiliser un gestionnaire de base de données avec une interface intuitive; il sera 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, utilisateurs de Mac que nous sommes, le logiciel SQLite Studio, disponible gratuitement ici.

Pour nos amis sous Windows, Cubrid semble être une alternative satisfaisante. On confesse ne l'avoir pour l'instant pas testée.

6 Exercices

Exercice 4.

On considère la relation :

NotesEleves					
Nom	Prenom	Classe	Maths	Phy	SI
Meyer	Romain	1	9,25	8,0	14,0
Martin	Paul	2	7,75	12,0	8,0
Robert	Marie	4	12,0	5,0	10,5
Michel	Lucile	2	11,5	11,0	13,5
Bernard	Sylvie	1	17,5	15,5	13,0
Martin	Romain	3	14,0	10,5	9,0
Meyer	Pierre	1	9,25	14,0	16,0
Dubois	Camille	3	11,5	6,0	8,5

Traduire les requêtes suivantes en langage SQL.

1. Sélectionner les prénoms des élèves des classes 1 et 3.
2. Sélectionner les noms et les notes des élèves ayant eu une note inférieure à 10.
3. Classer les élèves en fonction de leur note en Maths (dans l'ordre décroissant).
4. Classer les élèves en fonction de leur note en Maths puis en Physique (dans l'ordre décroissant).
5. Sélectionner les moyennes de classe dans chaque matière.
6. Sélectionner les prénoms des élèves dont la moyenne des notes est inférieure à 10.
7. Sélectionner les classes dont la moyenne est supérieure ou égale à douze en Maths.
8. Sélectionner les classes dont la moyenne est supérieure ou égale à douze dans les trois matières.
9. Sélectionner les classes dont la moyenne est supérieure ou égale à douze dans au moins une matière.

Exercice 5.

On s'intéresse ici à une table de données cinématographiques.

Plus précisément, on veut manipuler :

✗ Une table `Comedians`, avec les attributs suivants :
- `id_act`, un nombre entier (de type `INTEGER`) identifiant le ou la comédien.ne;
- `Name`, de type `TEXT`;
- `Country_act`, de type `TEXT` précisant le pays de provenance;
- `Age`, de type `INTEGER`;
- `Sex`, de type `TEXT`, (vaudra M ou F);
- `Size`, de type `INTEGER`, (exprimée en cm);

✗ Une table `Films`, avec les attributs suivants :
- `id_film`, de type `INTEGER`, un nombre identifiant le film ;
- `Title`, de type `TEXT`;
- `Main_role`, de type `INTEGER`, qui contiendra le numéro identifiant le ou la comédien.ne dans la table `Comedians`;
- `Year`, de type `INTEGER`;
- `Length`, de type `INTEGER`, (exprimée en minutes);
- `Country_Film`, de type `TEXT`, précisant le pays d'origine du film.

1. Quelles clés primaires et clés étrangères doit-on déclarer sur ces tables ? (on précisera vers quoi doivent pointer les clés étrangères).

2. Donner une requête SQL permettant de lister les titres des films français réalisés avant 1985.

3. Donner une requête SQL permettant de calculer la durée moyenne des films norvégiens.

4. Donner une requête SQL permettant d'afficher les titres et durées des films français et des films japonais, triés par ordre chronologique, du plus ancien au plus récent.

5. Interpréter la requête SQL suivante

```
SELECT AVG(Length) as avg_length FROM Films by Country_film
ORDER BY avg_length desc
```

On utilise maintenant les deux tables.

7. Donner une requête SQL permettant d'afficher les titres et années des films italiens dont le rôle principal est tenu par un ou une comédien.ne français.e.

8. Donner une requête SQL permettant d'afficher le nombre de films qui ont un acteur principal de sexe féminin ?

Exercice 6.

On considère un ensemble de relations utiles pour gérer un complexe hôtelier. Celui-ci est composé de différents bâtiments, identifiés par leur nom et leur nombre d'étoiles. Chaque chambre comporte un numéro et fait partie d'un bâtiment. Certaines chambres possèdent deux lits. On sépare donc les lits dans une autre relation, où ils comportent chacun un identifiant numérique unique au sein du complexe. Enfin, les nuitées sont identifiées par le nom du client, l'identifiant du lit et par la date.

Batiments	
NomBatiment	Etoiles
Rose	3
Jasmin	2
Lys	3

Chambres		
NumeroChambre	NomBatiment	Fenetres
1	Rose	2
2	Rose	1
3	Rose	1
1	Jasmin	1
2	Jasmin	0
3	Jasmin	1
4	Jasmin	1
1	Lys	3
2	Lys	2
3	Lys	2

Lits		
IdLit	NumeroChambre	NomBatiment
1	1	Rose
2	1	Rose
3	2	Rose
4	3	Rose
5	1	Jasmin
6	2	Jasmin
7	2	Jasmin
8	3	Jasmin
9	3	Jasmin
10	4	Jasmin
11	1	Lys
12	2	Lys
13	3	Lys

Nuitees		
Client	IdLit	Date
Bob	1	15-08-2023
Martin	8	18-08-2023
Dupond	1	03-07-2023
Dupont	2	03-07-2023
Simon	10	05-08-2023
Jones	1	13-08-2023
Smith	11	05-08-2023
Bill	7	02-08-2023
Sam	1	08-08-2023

On suppose qu'aux attributs `NomBatiment`, `Client`, `Date` sont associées des valeurs de type `str`, tous les autres attributs étant associés au type `int`.

On suppose encore que:

- ✗ l'attribut `NomBatiment` est une clé pour la table `Batiments`;
- ✗ l'ensemble d'attributs `{NumeroChambre,NomBatiment}` est une clé pour la table `Chambres`;
- ✗ l'attribut `IdLit` est une clé primaire pour la table `Lits`;
- ✗ l'ensemble d'attributs `{Client,IdLit,Date}` est une clé pour la table `Nuitees`;
- ✗ l'attribut `NomBatiment` est une clé étrangère pour les tables `Chambre` et `Lits`, et l'attribut `IdLit` est une clé étrangère pour la table `Nuitees`.

Pour chacune des recherches suivantes, donner la requête SQL correspondante ainsi que le résultat obtenu.

1. Obtenir le nom des clients ayant séjourné dans l'hôtel le 3 juillet 2023.
2. Obtenir le nom des clients ayant séjourné dans le bâtiment Jasmin.
3. Obtenir le nom des clients ayant séjourné dans un bâtiment 3 étoiles.
4. Obtenir le nom des clients ayant séjourné dans une chambre ayant au moins 2 fenêtres.
5. Obtenir les dates de réservation pour chaque chambre.
6. Obtenir le nombre de lits dans chaque bâtiment.
7. Obtenir le nombre de lits dans chaque chambre.
8. Obtenir pour chaque chambre, le nombre de clients à avoir réservé.