Lycée d'enseignement général et technologique international Victor Hugo
COLOMIERS

NSI
NUMÉRIQUE ET SCIENCES INFORMATIQUES

Les Bases De Données

Cours & Activité Pratique

Nom : Prénom : Date :

Table des matières

Les Bases De Données

Cours & Activité Pratique

1 Introduction

1.1 Qu'est-ce qu'une Base de Données ?

On pourrait définir le terme « base de données » en une phrase :

Une grande collection de données, avec des structures et une organisation, conservée sur un support persistant.

Pour gérer cette grande quantité de données, on va avoir besoin d'un logiciel. Ce logiciel est un système de gestion de bases de données, plus connu sous l'acronyme « SGBD ».

1.2 Caractéristiques des données

1.2.1 La persistance

La première caractéristique, c'est la persistance au cours du temps.

Lorsque vous lancez un programme que vous avez écrit, il construit ses données. Puis, lorsque vous l'arrêtez, tout s'arrête.

Les données sont là, dans l'entreprise ou l'organisation et elles restent dans le temps, des années, des dizaines d'années…

Donc, il va falloir bien gérer cette persistance.

1.2.2 Le volume

Étymologiquement, Au début des bases de données, on gérait des mégaoctets, puis on a géré des gigas, maintenant des téraoctets, voire des exaoctets.

A combien d'octets équivaut un exaoctet ? un exbioctet ?

Un exaoctet vaut 10^18 octets
Un exbioctet vaut 2^60 octets

Vous achetez un disque dur de 320 Go. Vous l'installez sur votre ordinateur. Quelle est la capacité affichée par ce dernier ? Utilisez le tableau des multiples de l'octet ci-dessous.

320 Go vaut a 320*10^9 octets

320 *10^9/2^30 = 298
320 Go vaut a environ 298 Gio

Multiples de l'octet : préfixes décimaux du SI et mésusages

Nom	Symbole	Valeur	Mésusage[a]
kilooctet	ko	10^3	2^{10}
mégaoctet	Mo	10^6	2^{20}
gigaoctet	Go	10^9	2^{30}
téraoctet	To	10^{12}	2^{40}
pétaoctet	Po	10^{15}	2^{50}
exaoctet	Eo	10^{18}	2^{60}
zettaoctet	Zo	10^{21}	2^{70}
yottaoctet	Yo	10^{24}	2^{80}

Multiples de l'octet : préfixes binaires

Nom	Symbole	Valeur
kibioctet	Kio	2^{10}
mébioctet	Mio	2^{20}
gibioctet	Gio	2^{30}
tébioctet	Tio	2^{40}
pébioctet	Pio	2^{50}
exbioctet	Eio	2^{60}
zébioctet	Zio	2^{70}
yobioctet	Yio	2^{80}

1.2.3 Répartition géographique et partage

Une partie de votre organisation est à Toulouse, les données aussi, mais une autre partie est à Milan, à Bengalore, à Québec...et vous partagez les données avec toutes les personnes de l'organisation.

Il va falloir gérer les caractéristiques de toutes ces données, réparties et partagées.

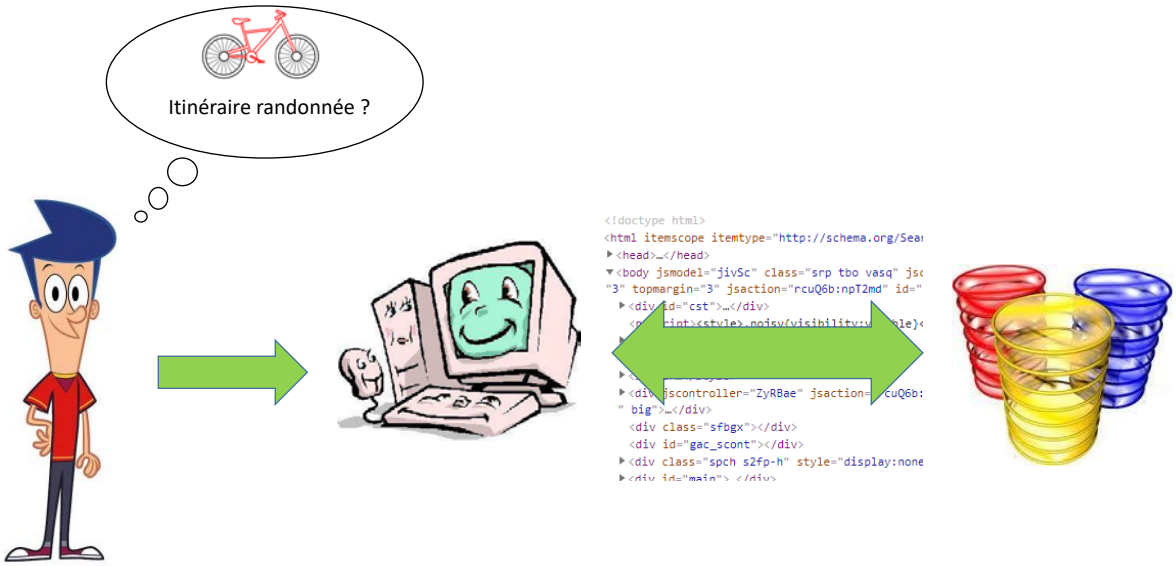
1.2.4 La base de données : un médiateur entre les humains et les données

Imaginons l'histoire de Blake qui souhaite trouver un itinéraire pour faire une randonnée à vélo dans les Cévennes.

Il pourrait écrire son petit programme qui va chercher des données, des octets ici ou là, suit des pointeurs, décode les données, etc…

Mais Blake n'a pas le temps de faire ça. De plus, il se dit que cela est probablement compliqué et qu'il n'a pas les compétences pour écrire des programmes compliqués.

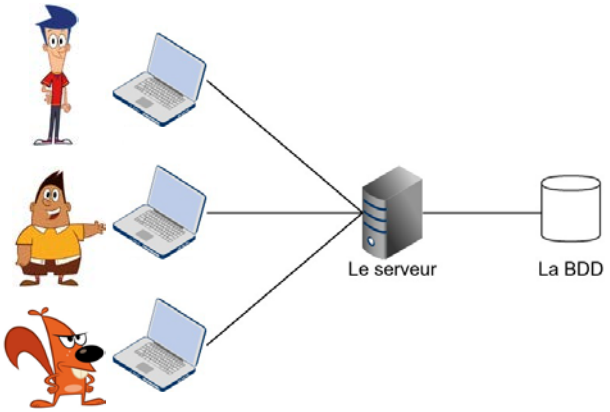
Donc, il veut s'exprimer dans des langages de haut niveau, et le rôle du système va être de traduire, dans ces programmes, qu'il ne veut pas écrire, d'aller chercher les données, là où elles se trouvent, puis de montrer à Blake l'information dont il a envie.



1.2.5 Le modèle d'architecture client-serveur…encore lui !

Blake utilise une application qui repose sur un logiciel. Ce dernier utilise des données.

On distingue deux parties distinctes : le client, sur lequel va tourner l'application, et le serveur, qui va gérer les données, le système de gestion de bases de données.



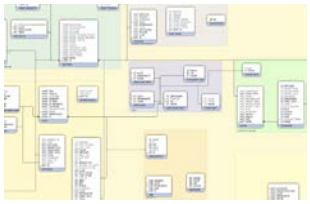
L'application de Blake a besoin de données (les itinéraires de randonnée). Elle envoie une requête au serveur, le serveur va répondre en rendant un certain nombre de résultats.

Les deux logiciels client et serveur coopèrent.

Le premier, c'est l'application, le système très spécifique qui traite les besoins de Blake. L'autre logiciel qui tourne sur le serveur n'est autre que le système de gestion de bases de données, qui concentre les données, de toute l'entreprise. Il est en charge de gérer des requêtes, des transactions, des pannes...et bien d'autres choses encore. Notamment plusieurs clients peuvent accéder à la même base comme Blake, Mitch et Maxus !

2 Le modèle relationnel

2.1 Qu'est-ce que c'est ?



Le **modèle relationnel** est une manière de modéliser les relations existantes entre plusieurs informations, et de les ordonner entre elles. Cette modélisation qui repose sur des principes mathématiques mis en avant par E.F. Codd est souvent retranscrite physiquement (« implémentée ») dans une base de données. (*Wikipédia*).

2.2 Un peu de vocabulaire … sur un exemple

Voici un tableau à 2 dimensions autrement appelé « **table** » dans le jargon des bases de données :

Nom de la relation

Logement

Nom de l'attribut

id	nom	capacité	type	lieu
pi	U Pinzuttu	10	Gîte	Corse
ta	Tabriz	34	Hôtel	Alsace
be	Benbow	45	Auberge	Cévennes
re	Réforme	134	Hôtel	Alpes
sh	Tashilhunpo	28	Monastère	Bretagne

Nuplet (enregistrement)

Attribut

Cette table est issue de la bdd des « Voyageurs ». Son auteur est S. Abiteboul, chercheur à l'INRIA.

Le nom de la « **relation »** va nous permettre d'évoquer les tables (ci-dessus la relation "Logement").
Les colonnes de ces tables sont appelées « **attributs** ».

Les attributs (libellé de colonne) sont nommés : ici "type", "id", "nom", « capacité » et « lieu ». Dans chaque colonne, on trouve la valeur des attributs.
Les lignes de la table sont appelées des « **n-uplets** ». Ici, nous avons un n-uplet qui représente un logement dans les Cévennes. Il y a plusieurs noms pour ces n-uplets, parfois on appellera ça des « **enregistrements** » ou « **record** » ou bien encore « **tuple** » en anglais.

A retenir : D'abord, cette idée selon laquelle les systèmes de gestion de bases de données sont des médiateurs entre les humains, qui veulent des informations et ces informations, qui sont sur des disques. Puis, l'idée que la base de données relationnelle est composée de ces tableaux à deux dimensions qu'on appelle des relations.

2.3 Formalisation de la « relation »

Le point de départ, c'est un **ensemble fini « E » d'attributs**.
Pour chaque attribut "A", on va avoir un "type", donc un ensemble de valeurs que peut prendre cet attribut : les entiers, les chaînes de caractères, ou d'autres types prédéfinis.

Un n-uplet sur "E" est une fonction qui associe à chaque attribut de "E" une valeur du bon type, c'est-à-dire du type de cet attribut.
Dans notre exemple :

be	Benbow	45	Auberge	Cévennes

Nous avons un n-uplet sur ces 5 attributs, à "Id" il donne la valeur "be", à "Nom", "Benbow", à :

```
"capacité" : 45, à "type" : "Auberge", à "Lieu" : "Cévennes"
```

Donc, une relation sur "E" est **un ensemble fini de n-uplets** sur un **ensemble fini « E » d'attributs**.

Il faut retenir que tous les n-uplets ont la même structure (on ne mélange pas des choux et des carottes !).

2.4 Le schéma

Le **schéma** d'une base de données est un ensemble fini de relations. Il définit la **structure** des données de la base.

Reprenons notre base de données exemple, celle des « Voyageurs ». Cette base est constituée de quatre relations nommées "Logement", "Activité", "Client", "Séjour" :

- Logement(id, nom, capacité, type, lieu)
- Activité(idLogement, codeActivité, description)
- Client(id, nom, prénom, ville, pays)
- Séjour(id, idClient, idLogement, début, fin)

Le type de chacune de ces relations nous donne la structure de cette relation. Pour notre base des Voyageurs, nous avons :

- type(Logement) = { id, nom, capacité, type, lieu }
- type(Client) = { id, nom, prénom, ville, pays }

Une **instance** de la base de données associe à chaque nom "N" de relation, une relation sur "type(N)". Elle définit la **valeur des données**. Par exemple, "Voyageurs(Logement)" sera une relation sur "type(Logement)" ou encore un ensemble fini de n-uplets sur type(Logement).

"Voyageurs(Activité)" sera une relation sur :

```
type(Activité) = {idLogement, codeActivité, description}
```

Par conséquent, une Base de données est dite relationnelle si :

- ses relations sont des ensembles finis de n-uplets. Par exemple, un tableau d'entiers à deux dimensions n'est pas une relation au sens des bases de données relationnelles, parce qu'il n'est pas fini.
- ses n-uplets possèdent la même structure (le type de la relation)
- les entrées de ses tables doivent être atomiques. C'est-à-dire qu'une entrée ne peut pas être une autre relation ou un ensemble.

2.5 La clé primaire

La **clé primaire** d'une relation est un ou plusieurs attributs qui identifient de manière **unique** un n-uplet de la relation.

Logement

id	nom	capacité	type	lieu
pi	U Pinzuttu	10	Gîte	Corse
ta	Tabriz	34	Hôtel	Alsace
be	Benbow	45	Auberge	Cévennes
re	Réforme	134	Hôtel	Alpes
sh	Tashilhunpo	28	Monastère	Bretagne

Clé primaire

Clé du nuplet

Nous avons vu que le nom de l'attribut définissait la colonne. D'une façon analogue, la clé du n-uplet définit complètement le n-uplet.
Par conséquent, 2 n-uplets distincts d'une relation ne peuvent pas avoir la même clé.
Nous verrons plus loin qu'on définit également d'autres clés qui ne sont pas « primaires ».

Dans notre base de données des Voyageurs, les clés primaires (en gras et italique) sont :

- Logement (***id***, nom, capacité, type, lieu)
- Activité (***idLogement, codeActivité***, description)
- Client (***id***, nom, prénom, ville, pays)
- Séjour (***id***, idClient, idLogement, début, fin)

2.6 A Faire Vous-même I

Voici un tableau « T » à deux dimensions :

764768	44	184	CTRE	COM DE LA BOULETTERIE	CENTRE COM DE LA BOULETTERIE	BOULETTE
764769	44	184	CITE	D HERBINS	CITE D HERBINS	HERBINS
764770	44	184	LOT	ILE DU PE	LOTISSEMENT ILE DU PE	PE
764771	44	184	CTRE	REPUBLIQUE	CENTRE REPUBLIQUE	REPUBLIQ
764772	44	184	CITE	DE KERBRUN	CITE DE KERBRUN	KERBRUN
764773	44	184	CTRE	CIAL DE KERLEDE	CENTRE CIAL DE KERLEDE	KERLEDE
764774	44	184	CTRE	CIAL DE TREBALE	CENTRE CIAL DE TREBALE	TREBALE
764775	44	184	PKG	DU BOIS SAVARY	PARKING DU BOIS SAVARY	SAVARY
764776	44	184	PKG	COEUR DE VILLE	PARKING COEUR DE VILLE	VILLE
764777	44	184	PKG	DU COMMANDANT GATE	PARKING DU COMMANDANT GATE	GATE

a. Peut-on caractériser les attributs de ce tableau ? Si oui, essayer de leur donner un nom (libellé).

id departement bureau_distributeur type nom nom_entier rue

b. L'ensemble d'attributs « E » est-il fini ? Si, oui, combien y a-t-il d'attributs ?

Il y a 7 attributs

c. Pouvez-vous donner un type à chaque attribut ?

```
id,departement,bureau_distributeur,type,nom,nom_entier,rue
entier,ent,ent,chaine_de_caractere,ch_de_cara,ch_de_cara,ch_de_cara
```

d. Formaliser l'ensemble E d'attributs

```
type(E) = {id,departement,bureau_distributeur,type,nom,nom_entier,rue}
```

e. Le tableau « T » est-il composé d'un nombre fini de n-uplets ? Quel est ce nombre ?

```
10 n-uplets
```

f. « T » peut-il être qualifié de « relation » ? Justifiez votre réponse

```
Oui car T est un ensemble de n-uplets sur l'ennsemble E fini d'attributs
```

g. Donnez un nom à cette relation. Quel est son type ? Répondez à cette question en complétant l'expression type(<nom_de_relation>) = {….}.

```
type(Rues_StNazaire) = {id,departement,bureau_distributeur,type,nom,
                        nom_entier,rue}
```

h. Tous les n-uplets de « Rues_StNazaire » sont-ils unique ? Selon vous, quelle est la clé primaire de la table « Rues_StNazaire » ?

```
Oui car l'id est la cle primaire de la table
```

3 La conception des bases de données relationnelles

Nous n'allons pas nous lancer dans l'apprentissage des méthodes de conception. Néanmoins il est intéressant d'aborder :

- les grands principes
- les contraintes

inhérents à la conception d'une base de donnée relationnelle.

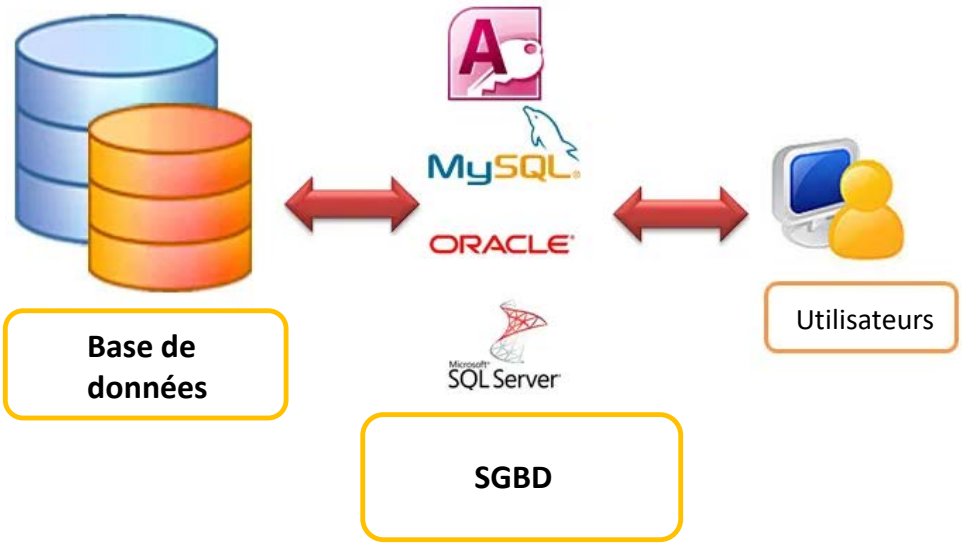
3.1 Les trois grands principes

Le principe d'Universalité

Les SGBD (Systèmes de Gestion de Base de Données) sont conçus pour :

- Toutes les données d'une entreprise pour
- Tous les utilisateurs et pour
- Toutes les applications

Des fonctionnalités étendues sont accessibles afin de garantir le respect de ce principe : gestion de la concurrence, des pannes, de la distribution…



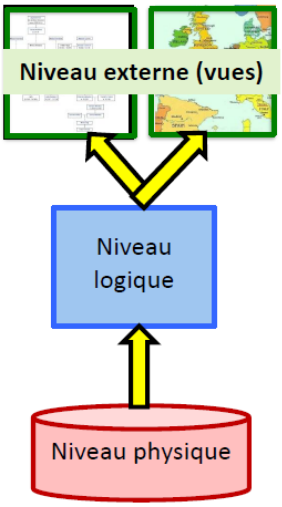
Le principe d'Abstraction

Le respect de ce principe permet de :

- Voir les données de façon abstraite, par exemple des tableaux à deux dimensions (en ignorant les détails de leur organisation réelle sur disque)
- Les manipuler avec des langages de haut niveau (et non en écrivant du code informatique compliqué)

Le SQL (Strutured Query Language) est le langage standard universel.

Le principe d'Indépendance



L'indépendance consiste d'abord à regarder les données à partir de plusieurs niveaux :

- Le premier est le niveau physique : la façon dont les données sont organisées sur disque
- Le deuxième niveau est le niveau logique : c'est une vue abstraite par le biais de laquelle on voit des tableaux à deux dimensions (modèle relationnel)
- Le troisième niveau, le niveau externe : des vues adaptées à chaque utilisateur ou à chaque type d'utilisateur. Ces vues sont particulières et adaptées aux besoins de chacun.

3.2 Les contraintes

Lorsqu'on conçoit une base de données, on réfléchit à sa structure. Celle-ci doit :

- Répondre aux besoins des demandeurs,
- Satisfaire un certain nombre de contraintes et d'exigences

Notamment, ces contraintes doivent être prises en compte par le schéma relationnel de la base de données.

Nous allons passer en revue les contraintes majeures qui pèsent sur le schéma d'une base de données relationnelle. Pratiquement, nous répondrons à deux questions :

3.2.1 Qu'est-ce qu'un mauvais schéma relationnel ?

Reprenons notre de la base de données des Voyageurs. Nous avons vu que son schéma était composé de quatre relations :

- Logement (***id***, nom, capacité, type, lieu)
- Activité (***idLogement, codeActivité***, description)
- Client (***id***, nom, prénom, ville, pays)
- Séjour (***id***, idClient, idLogement, début, fin)

Pourquoi quatre relations ? C'est vrai que cela a l'air un peu compliqué. Tout ça pour savoir quel(s) client(s) ont effectué un séjour dans un logement et quelle(s) activité(s) le(s) client(s) peu(ven)t-il(s) pratiquer dans ce(s) logement(s) ?

Ne peut-on pas faire des regroupements pour simplifier un peu ?

A faire vous-même II

Pour simplifier notre analyse, imaginons que notre base de données des Voyageurs ne contient plus que les deux relations. On l'appelle « Voyageurs2 » :

- Logement (***id***, nom, capacité, type, lieu)
- Activité (***idLogement, codeActivité***, description)

Logement

code	nom	capacité	type	lieu
ca	Causses	45	Auberge	Cévennes
ge	Génépi	134	Hôtel	Alpes
pi	U Pinzutu	10	Gîte	Corse
ta	Tabriz	34	Hôtel	Bretagne

Activité

codeLogement	codeActivité	description
ca	Randonnée	Sorties d'une journée en groupe
ge	Piscine	Nage loisir non encadrée
ge	Ski	Sur piste uniquement
pi	Plongée	Baptèmes et préparation des brevets
pi	Voile	Pratique du dériveur et du catamaran

a. Regrouper les deux tables « Logement » et « Activité ». Dans un objectif de simplification, on ne gardera que cinq attributs. On appelle cette table « Logement_Activité ». Donner ci-dessous le contenu de l'unique table obtenue:

Logement_Activité

Code	Nom	Capacité	Type	Lieu	Code Activité
ca	Causses	45	auberge	Cévennes	Randonnée
ge	Génépi	134	Hotel	Alpes	Piscine
ge	Génépi	134	Hotel	Alpes	Ski
pi	U Pinzutu	10	Gite	Corse	Plongée
pi	U Pinzutu	10	Gite	Corse	Voile

Cela a l'air plus simple non ? Nous avons tout sous la main, il y a un seul espace de recherche. Regardons cela d'un peu plus près !

b. Tentez d'insérer dans la table « Logement » de « Voyageurs2 » le n-uplet (el, Elbazet, 50, Auberge, Occitanie) :

Code	Nom	Capacité	Type	Lieu	Code Activité
ca	Causses	45	Auberge	Cévennes	Randonnée
ge	Génépi	134	Hotel	Alpes	Piscine
ge	Génépi	134	Hotel	Alpes	Ski
pi	U Pinzutu	10	Gite	Corse	Plongée
pi	U Pinzutu	10	Gite	Corse	Voile
el	elbazet	50	Auberge	Occitanie	

Code Activité n'a pas de valeur

c. Quelles sont les deux options possibles ? Qu'est-ce que cela implique ?

Code	Nom	Capacité	Type	Lieu	Code Activité
ca	Causses	45	Auberge	Cévennes	Randonnée
ge	Génépi	134	Hotel	Alpes	Piscine
ge	Génépi	134	Hotel	Alpes	Ski
pi	U Pinzutu	10	Gite	Corse	Plongée
pi	U Pinzutu	10	Gite	Corse	Voile
el	elbazet	50	Auberge	Occitanie	NULL

d. Admettons qu'on puisse pratiquer le VTT, le tennis et la pêche dans le logement « Elbazet ». Ecrivez ci-dessous ce que cela implique en termes de n-uplets. Que remarquez-vous ?

Cela va faire 3 n-uplets avec seulement l'attribut Code_Activité qui va changer

e. Vous venez de recevoir une information selon laquelle le propriétaire de l'auberge a mis son logement aux normes de l'hôtellerie. Par conséquent, le logement prend désormais le type « Hôtel ». Vous faites faire cette modification par un tiers qui commet une erreur. Dans un n-uplet, le type saisi est « hotel » et non « Hôtel ». Ecrivez ci-dessous ce que cela implique en termes de n-uplets. Selon vous l'erreur peut-elle avoir des conséquences ? Lesquelles ?

L'erreur peut avoir comme consequence que on ne peux plus retrouver tous les Hôtel d'un seul coup vu que 2 syntaxe differentes signifie "Hotel"

A retenir :
A moins que cela fasse l'objet d'un calcul (une vue), mettre toute l'information dans une seule table n'est pas une bonne idée. C'est un mauvais schéma relationnel
Nous avons mis en évidence plusieurs anomalies :

- accepter qu'un attribut puisse prendre la valeur « NULL »
- générer de la redondance d'attributs
- introduire des incohérences dans une base de données.

Tôt ou tard, ces anomalies dévaloriseront sérieusement la base de données, ou pire la rendront inutile !

3.2.2 Qu'est-ce qu'un bon schéma relationnel ?

Un bon schéma est un schéma qui ne comprend pas d'anomalie !

La conception d'une base de données doit aboutir à la réalisation d'un schéma normalisé sans perte d'information. Bien que nous ne voyions pas les concepts de la normalisation des schémas des tables d'une base de données, nous allons partir de l'étape précédente (§ 3.2.1) pour en énoncer les grands fondements.
La normalisation consiste à décomposer en plusieurs tables. Nous n'aurons pratiquement jamais un schéma de base de données avec une seule table. Au contraire, à l'instar de la base des « Voyageurs », nous aurons toujours plusieurs tables où l'information est décomposée.

Au moment de la décomposition, il faut veiller à ne pas perdre d'information. Par exemple, il ne faut pas perdre l'information selon laquelle une activité a lieu dans un certain logement. Si on fait la décomposition brutale en mettant « logements » d'un côté et « activités » de l'autre, nous aurons peut-être gagné l'absence d'anomalies mais nous aurons perdu un lien d'information essentiel. Un peu plus loin dans cette activité, nous apprendrons le concept de « jointure » en SQL.
Les jointures en SQL permettent de reconstituer l'information qu'on aurait eue en ayant tout dans une seule table. Elles garantissent que le schéma normalisé soit sans perte d'information.

Dans « Voyageurs », si nous faisons autant de jointures que nécessaire, nous recréerons la situation où tout était dans une seule table, mais cette fois par calcul. Le fait de le faire par calcul évite les anomalies. Si nous voulons en disposer de manière un peu permanente, nous le faisons sous forme de vue.

3.3 Comprendre la normalisation relationnelle

Pour nos logements, prenons comme exemple le schéma (1) suivant : (code, nom, adresse, activité, description)

code	nom	adresse	activité	description
pi	U Pinzutu	12 quai Napoléon	Plongée	Dans le golfe
ta	Tabriz	Avenue du grand large	Plongée	Dans la rade
pi	U Spuntinu	12 avenue Paoli	Voile	En club associatif
ma	Macchia	12 quai Napoléon	Gastronomie	Spécialités locales

Table « Logement » tirée du cours de Philippe Rigaux, Professeur des Universités au CNAM

3.3.1 La notion essentielle de Dépendance Fonctionnelle (DF)

Soient les attributs A et B. Il y a « Dépendance Fonctionnelle » (DF) de A vers B quand la connaissance de la valeur de A implique la connaissance de la valeur de B. Elles permettent de raisonner sur le contenu d'une base et d'analyser sa qualité.

Soient les DF suivantes :

- code → nom, adresse
- adresse → code, nom
- code, activité → description

A faire vous-même III

a. Tenter de justifier ces DF dans la table « Logements ». Pour chaque DF, dire si elle est toujours respectée par les n-uplets de la table « Logements » ci-dessus ? Si non, dire pourquoi et ce que cela signifie.

```
- code -> nom,adresse : n'est pas respecté car le code pi n'a pas
toujours le meme nom et adresse

- adresse -> code,nom : n'est pas respecté car l'adresse 12 quai
Napoléon ne donne pas le meme code ni le meme nom

- code,activvité -> description : toujours respecté dans la table
```

b. Etes-vous d'accord avec la phrase suivante : « si on a deux fois le même code logement et le même code activité, les deux n-uplets doivent être strictement identiques si on veut respecter les dépendances fonctionnelles ». Expliquez votre opinion et le cas échéant, le raisonnement sous-jacent :

```
Le code donnera le nom et l'adresse.
L'activité et le code donnerons la description
Tous les arguments serons alors connu grace au code et a l'activité si
on veux respecter les dependences fonctionnelles

Le raisonement est vrai
```

Un bon schéma relationnel respecte les DF.

3.3.2 La notion essentielle de Clé

La clé d'une relation est un ensemble minimal d'attributs dont dépendent fonctionnellement tous les autres attributs.
Tous les autres attributs doivent dépendre fonctionnellement de la clé.

A faire vous-même IV

a. La paire d'attributs (code, activité) est une clé de la relation (code, nom, adresse, activité, description). Expliquer pourquoi :

Grace aux 2 attributs (code,activité) et les dependences fonctionneles, on obtient les attributs adresse, nom, descrition

b. « code » est-il une clé ? Pourquoi ?

code n'est pas une clé car la description ni l'activité ne peuvent etre connu

c. (code, activité, adresse) est-il une clé ? Pourquoi ?

Ce n'est pas une clé car ce n'est pas l'ensemble minimal dont dépendent fonctionnellement tous les autres attributs

Si j'ai 2 n-uplets avec la même valeur de clé, comme elle détermine tous les autres attributs, les 2 n-uplets doivent être strictement identiques.
Par conséquent, une manière simple de respecter la dépendance fonctionnelle est d'imposer qu'une valeur de clé soit unique dans une relation.
La connaissance d'une clé détermine une contrainte d'unicité dans le but de respecter les dépendances fonctionnelles.

3.3.3 La normalisation

Un schéma de relation R est en troisième forme normale quand tout attribut non-clé dépend directement d'une clé.

Nous en retiendrons l'approximation suivante : dans toute dépendance fonctionnelle de K → A sur les attributs de R, K est une clé. Cela signifie que toutes les dépendances fonctionnelles ont pour origine la clé d'une relation.

A faire vous-même V

Rappelons le schéma (1) précédent : (code, nom, adresse, activité, description) avec les DF suivantes :

- code → nom, adresse
- adresse → code, nom
- code, activité → description

En vous aidant des définitions précédentes, dire si le schéma précédent est normalisé. Expliquer pourquoi.

Le shéma precedent n'est pas normalisé car toutes les dependences fonctioneles ne sont pas respecter pour tous les atributs

Notre intuition nous souffle que nous ne devions pas mélanger des informations de natures différentes dans une même relation, dans un même schéma.

Le schéma normalisé est donc :

- Logement (***code***, nom, ***adresse***)
- Activité (***code, activité,*** description)

Logement possède deux clés, code et adresse. Activité possède une clé, la paire (code, activité).
Ces deux relations sont en troisième Forme Normale.

Au cours de notre prochaine séance sur SQL, nous verrons que nous n'avons perdu aucune information car nous reconstituerons la relation initiale par jointure.

La normalisation relationnelle décompose l'ensemble des attributs en relations saines
- Toute relation a une clé,
- Tous les attributs dépendent directement de la clé (forme normale)
- Une valeur de clé doit apparaître une seule fois
- La décomposition en plusieurs relations est compensée par la possibilité d'effectuer des jointures

3.4 Le langage SQL

Afin de mettre en pratique les concepts découverts dans les séances précédentes, nous allons utiliser des outils spécifiques. Ces outils nous permettront de découvrir et d'apprendre les bases du langage SQL.

3.4.1 Outils de mise en pratique

Notre activité repose sur quatre logiciels libres installés sur notre nano ordinateur préféré : notre Raspberry Pi ! Maintenant, faisons les présentations :

RASPBIAN

- Le Système d'exploitation Raspbian (ditribution Linux pour Raspberry) assure l'attribution des ressources aux autres composants (Cf. Activité « Architectures matérielles et OS » réalisée en classe de Première)

- Apache est le serveur web « frontal » : il est « devant » les autres et répond directement aux requêtes du client web (navigateur)

Apache

MariaDB

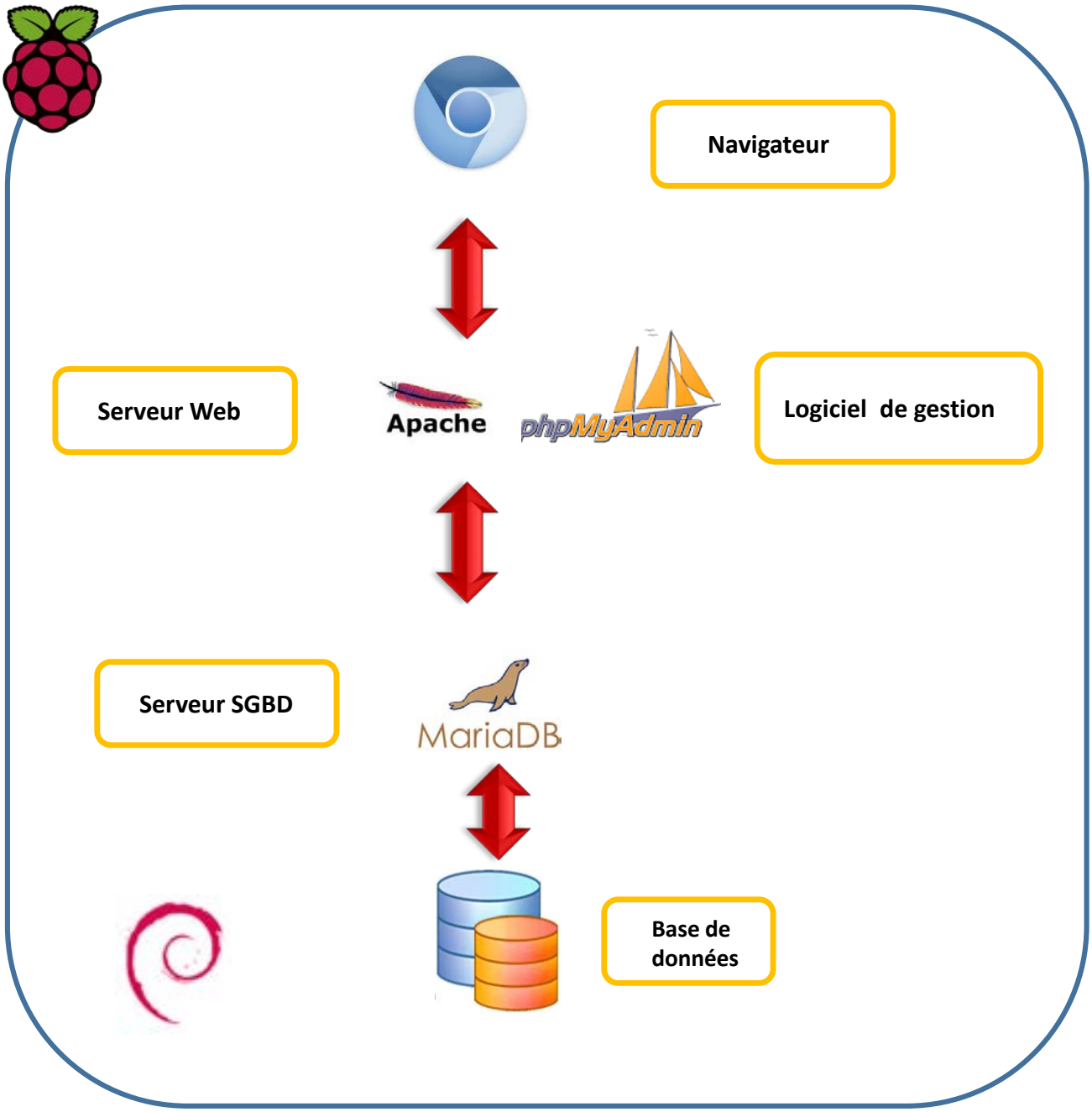
- MariaDB : le système de gestion de bases de données (SGBD). Comme nous l'avons déjà vu, il permet de stocker et d'organiser des données

- phpMyAdmin (PMA) : c'est une application Web de gestion pour les systèmes de gestion de base de données tels MariaDB. PMA est réalisée principalement en PHP et distribuée sous licence GNU GPL. Le langage de script PHP permet la génération de pages web dynamiques et la communication avec le serveur MariaDB. Nous pouvons également utiliser Perl ou Python.

phpMyAdmin

MariaDB propose nativement une interface non graphique. Cette interface utilise la « ligne de commande ». Nous verrons qu'il tout à fait possible d'utiliser et de gérer une base de données uniquement avec cette interface.

Clients et serveurs étant hébergés par le système, nous avons le schéma de fonctionnement suivant :

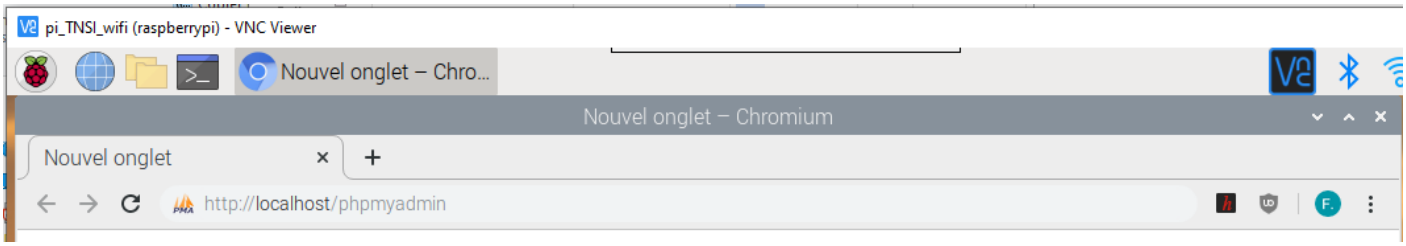


3.4.2 Préambule

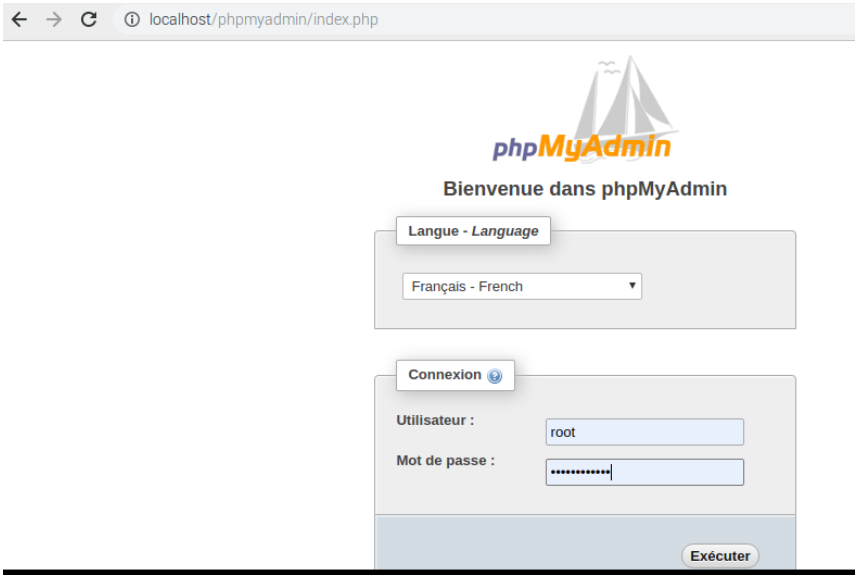
Dans un premier temps, familiarisons-nous avec l'application phpMyAdmin.

A faire vous-même VI

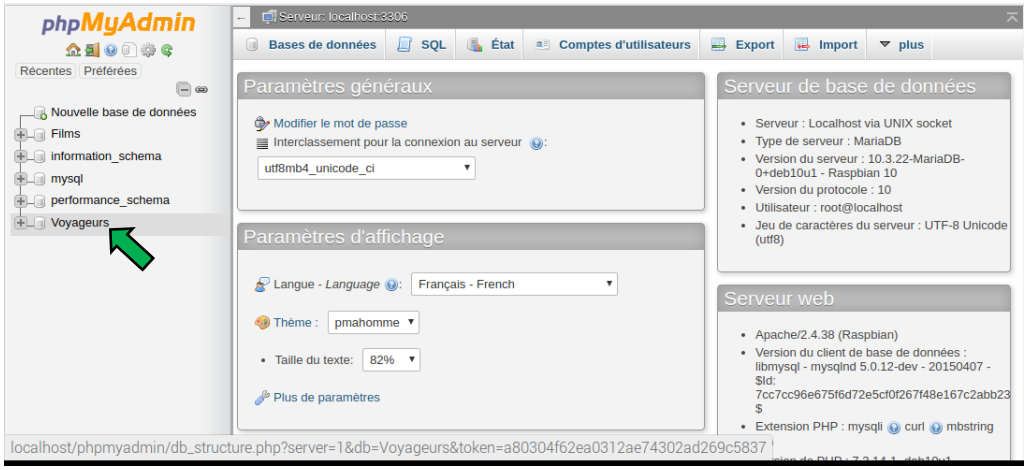
a. Connexion à la base de données MariaDB via phpMyAdmin. Dans la barre d'Url du navigateur Chromium, saisir l'URL suivante :



b. Saisir le mot de passe « raspberrynsi » puis cliquer sur le bouton « Exécuter » :



c. Cliquer sur le bandeau « Voyageurs » afin d'afficher le contenu de la base de données :



d. Etablir le schéma de la base de données.

Structure de la base : quelles sont les tables qui composent la base de données ?

```
La base est composé des tables : Activité, Logement, Séjour, Voyageur
```

En sélectionnant une table puis son onglet « Structure », nous pouvons visualiser sa structure.

En déduire le schéma relationnel de la base de données « Voyageurs » :

```
Activité(codeLogement, codeActivité, description)
Logement(code, nom, capacité, type, lieu)
Séjour(idSéjour, idVoyageur, codeLogement, début, fin)
Voyageur(idVoyageur, nom, prénom, ville, région)
```

e. Identifier les clés primaires de chaque table. Quel est leur rôle ?

```
Clé primaire Activité : codeLogement, codeActivité
Clé primaire Logement : code
Clé primaire Séjour : idSéjour
Clé primaire Voyageur : idVoyageur
Leur role est de connaitre les autres atributs grace aux DP
```

Remarque importante : la table « Séjour » possède un attribut « IdVoyageur ». Cet attribut est la clé primaire de la table Voyageur. Par conséquent, les deux tables « Séjour » et « Voyageur » partageant des valeurs par le biais de leur attribut « idVoyageur », on peut donc associer des n-uplets de ces deux tables dans la mesure où ils partagent une valeur de l'attribut en commun.

L'opération qui permet d'associer les n-uplets dans la mesure où ils partagent une valeur d'un attribut en commun s'appelle la jointure.

Ici, dans « Voyageur », l'attribut « idVoyageur » est une clé primaire. Dans « Séjour », cet attribut porte le nom de « **clé étrangère** ».

De manière générale, les bases sont conçues de manière à ce que les n-uplets qui sont liés partagent des valeurs communes, souvent basées sur les clés.

Une clé étrangère peut être une partie d'une clé primaire.

f. Existe-t-il d'autres clés étrangères dans « Séjour » ? dans d'autres tables de la base ? Lesquelles ?

```
Dans Séjour il y a aussi codeLogement.
Il n'y en as pas dans les autres tables.
```

Afin de bien comprendre un schéma relationnel, il faut :

- Identifier la clé primaire de chaque relation (il doit toujours y en avoir une)
- Identifier les clés étrangères et comprendre quelle clé primaire elles référencent.
- Comprendre comment les n-uplets sont liés les uns aux autres.

3.4.3 Parlons SQL

A faire vous-même VII

a. Visualiser le contenu de chaque table de « Voyageurs » avec l'ordre « select ». Cliquez sur une table de « Voyageurs » puis cliquez sur l'onglet « SQL ». Quelle requête apparait en ligne 1 ?

```
SELECT * FROM `Activité` WHERE 1
```

b. Cliquez sur le bouton « Exécuter » en bas et à droite de la fenêtre « SQL ». Caractériser l'affichage. Refaire les opérations a et b pour chaque table de la base.

```
Ca affiche toute la table Activité

SELECT * FROM `Logement` WHERE 1
Ca affiche toute la table Logement

SELECT * FROM `Séjour` WHERE 1
Ca affiche toute la table Séjour

SELECT * FROM `Voyageur` WHERE 1
Ca affiche toute la table Voyageur
```

La forme de base d'une requête « SQL » est celle d'un "bloc" constitué de trois clauses.

select expressions
from une_table
[**where** conditions]

L'interprétation est toujours la suivante (dans l'ordre) :

- from : l'espace de recherche. C'est toujours une table
- where : conditions imposées aux nuplets du « from »
- select : construit un nuplet-résultat à partir de chaque nuplet du « from » qui satisfait le « where »

Dans un premier temps : on suppose que l'espace de recherche est une des tables de la base

Exemple : dans la table « Logement », la requête dont le résultat est « Tabriz » est :

```
SELECT nom FROM `Logement` WHERE code = 'ta'
```

c. Dans la table « Voyageur », écrire la requête dont le résultat est « Alexandra David-Néel »

```
SELECT prénom,nom FROM `Voyageur` WHERE idVoyageur = 30
```

La clause « select » nous permet de sélectionner un ou plusieurs attributs A_1, … , A_n
Les conditions de l'opérateur « **where »** :
Comparaison (=, <, >, ! =) entre un attribut et une constante
Comparaison entre un attribut et un autre attribut
Le « where » est une formule propositionnelle (avec and, or, not et le parenthésage) combinant ces conditions.

d. Quel est l'espace de recherche de la requête suivante ? :

```
SELECT * FROM `Logement` WHERE lieu = 'Corse' OR (capacité > 30 AND type = 'Hôtel')
```

Pouvez-vous prévoir son résultat ? Vérifiez-le en tapant la requête dans un onglet SQL.

```
L'espace de recherche est la table Logement

On pouvais prevoir le resultat car la requete demande les logements de
Corse et tout les hotel avec un capacité superieur a 30
```

e. Ecrire la requête dont le résultat est : les Auberges d'Occitanie et des Cévennes. Testez cette requête dans un onglet SQL.

```
SELECT * FROM `Logement` WHERE type = 'Auberge' AND (lieu = 'Cévennes' OR 'Occitanie')
```

f. Dans une requête, Il peut être aisé voire nécessaire de renommer des colonnes ou des tables d'une base de données avec un alias. Cette bonne pratique facilite la lecture des requêtes. Dans l'onglet SQL de la table « Voyageur », saisir la requête suivante :

select prénom as P, nom as N from Voyageur as V where V.idVoyageur > 20

Le résultat est-il différent de celui obtenu au c. ? Pourquoi ?

```
Le resultat est different car la condition est differente
```

g. Ré-écrire la requête d. :
- en renommant la table « Logement » L
- en affichant le résultat sous la forme suivante :

co	n	ca
ge	Génépi	134
pi	U Pinzutu	10
ta	Tabriz	34

```
SELECT code as co, nom as n, capacité as ca FROM `Logement` as L where L.lieu = 'Corse' OR (L.capacité > 30 AND L.type = 'Hôtel')
```

h. SQL ne fait pas de tri afin de pouvoir éliminer les doublons plus facilement. Pour ce faire, on utilise la clause DISTINCT. Par exemple, avant d'organiser un séjour, admettons que l'on souhaite connaitre les différents types de logement proposés. Dans l'onglet SQL de la table correspondante, saisir la requête suivante :

« SELECT type FROM Logement »

Quelle est la caractéristique du résultat obtenu ? Eliminez cette caractéristique en saisissant la requête :

« SELECT DISTINCT type FROM Logement »

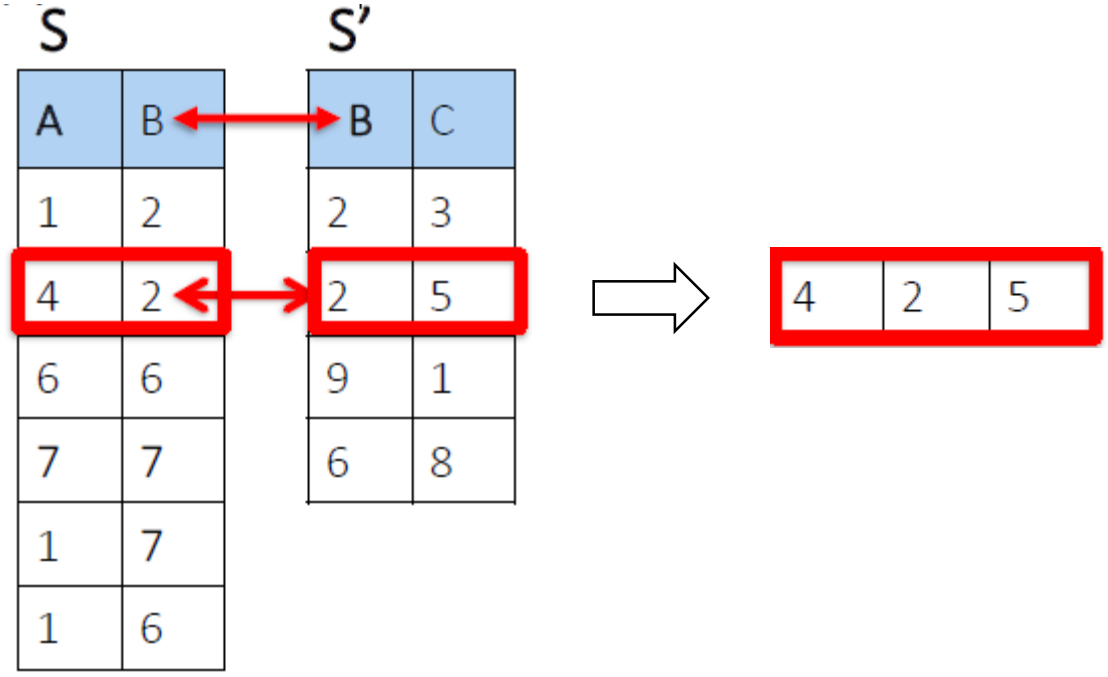
3.4.4 Les jointures

Il existe plusieurs façons d'appréhender le concept de jointure. Dans cette activité, nous l'aborderons sous deux angles :

- la jointure dite « naturelle »
- la jointure issue d'un produit cartésien

A. La jointure dite « naturelle »

Commençons par illustrer ce concept avec un exemple simple.
Soient deux tables S et S'



L'auteur de l'exemple est S. Abiteboul, chercheur à l'INRIA.

Dans l'exemple ci-dessus, la jointure considère les deux tables S et S', en extrait deux n-uplets qui ont la même valeur pour la partie commune des attributs (ici B), et les combine. Effectivement, nous avons un attribut en commun dans les deux types :
type(S) = (A,B) et type(S') = (B,C). Donc type(S) ∩ type(S') = B.

Ici, nous avons deux attributs qui sont d'accord sur la partie en commun.

Cela paraît naturel de construire un nouvel n-uplet (4,2,5) avec les deux n-uplets (4,2) et (2,5).
En faisant de même pour tous les n-uplets des deux tables, on s'aperçoit qu'on peut construire la table « résultat » R.

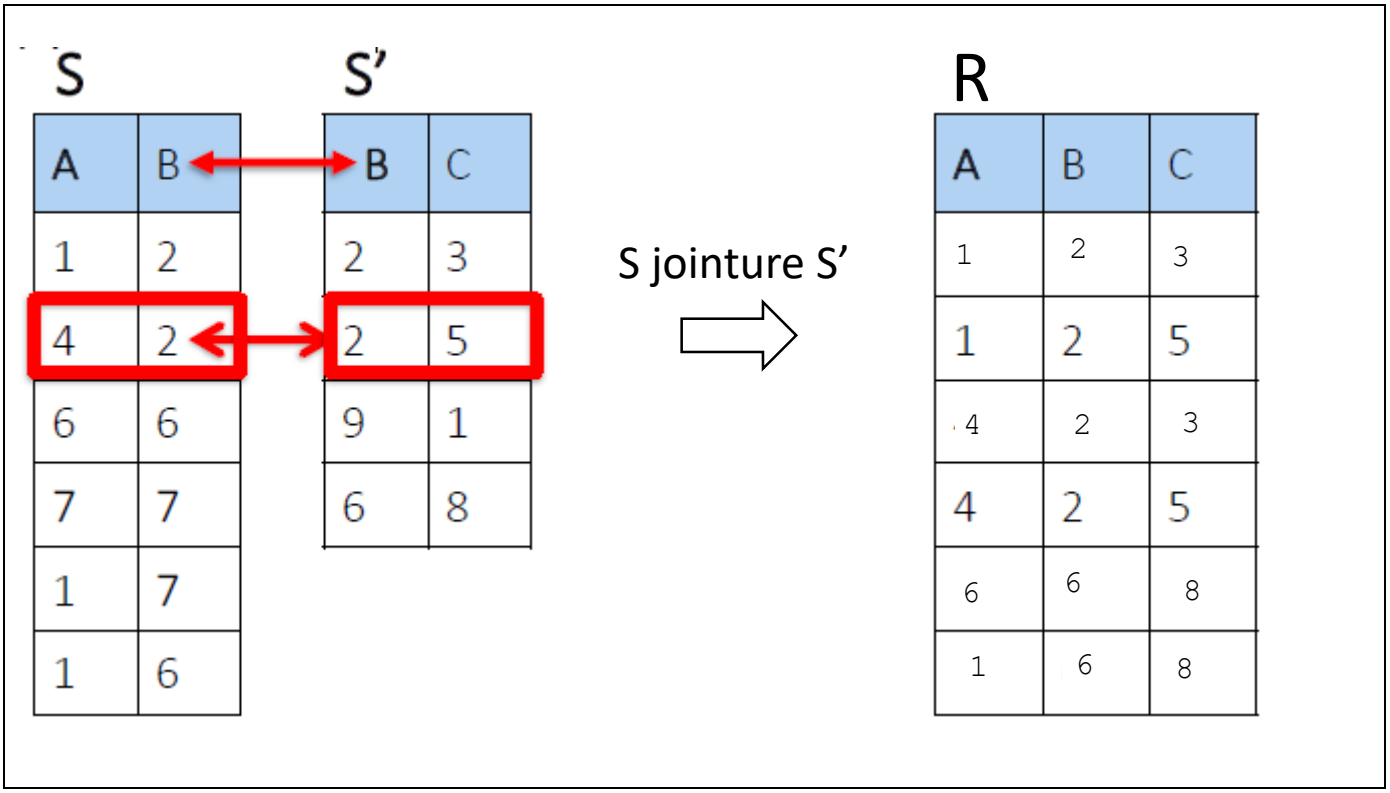
A faire vous-même VIII

Compléter l'algorithme simplifié de l'opération de jointure :

```
Tant que la fin de S n'est pas atteinte
        Lire n-uplet de S
        Tant que la fin de S' n'est pas atteinte
            Lire n-uplet de S'
            Si test d'égalité des valeurs des attributs en commun est VRAI
                    n-uplet = ABC
                    Stocker le n-uplet résultat dans la table « Résultat »
            FinSi
            n-uplet de S' = n-uplet de S' + 1
      n-uplet de S = n-uplet de S +1
```

A faire vous-même IX

A l'aide de l'algorithme simplifié ci-dessus, complétez la table « résultat » R ci-dessous :



A	B
1	2
4	2
6	6
7	7
1	7
1	6

B	C
2	3
2	5
9	1
6	8

A	B	C
1	2	3
1	2	5
4	2	3
4	2	5
6	6	8
1	6	8

Analyse du contenu de « R » : Retrouve-t-on dans « R » tous les n-uplets contenus dans S et S' ?

Non car des n-uplets n'avais pas de valeurs en commun entre S et S'

Une jointure est donc une manière de combiner les n-uplets de plusieurs tables distinctes.

Dans notre exemple, c'est l'ensemble des n-uplets sur l'union des deux ensembles d'attributs, tel qu'on puisse trouver un n-uplet dans S et un n-uplet dans S' qui soient d'accord sur l'intersection.

A faire vous-même X

Illustrons ceci au sein de notre base des « Voyageurs »

a. Dans l'onglet SQL de la table « Séjour », saisir la requête suivante :

```
select * from Séjour natural join Voyageur
```

.....

idVoyageur	idSéjour	codeLogement	début	fin	nom	prénom	ville	région
10	1	pi	20	20	Fogg	Phileas	Ajaccio	Corse
20	2	ta	21	22	Bouvier	Nicolas	Aurillac	Auvergne
30	3	ge	2	3	David-Néel	Alexandra	Lhassa	Tibet
20	4	pi	19	23	Bouvier	Nicolas	Aurillac	Auvergne
20	5	ge	22	24	Bouvier	Nicolas	Aurillac	Auvergne
10	6	pi	10	12	Fogg	Phileas	Ajaccio	Corse
30	7	ca	13	18	David-Néel	Alexandra	Lhassa	Tibet
20	8	ca	21	22	Bouvier	Nicolas	Aurillac	Auvergne

.....

Analyse du résultat de la requête

b. De quel type de jointure s'agit-il ? Quel mot clé nous l'indique ?

C'est une jointure Naturelle, c'est indiqué grace au mot "NATURAL"

c. Comment pouviez-vous prévoir le résultat ?

On pouvais prevoir le resultat grace a la cle étrangere idVoyageur present dans les 2 tables Séjour et Voyageur

d. Sur quel(s) attribut(s) la jointure s'est-elle exécutée ? Pourquoi ?

La jointure c'est faite sur l'atribut idVoyageur

e. Cette requête reprend-elle la forme de base présentée au §3.4.3 ? Est-ce un problème ? Pourquoi selon vous ?

Cette requete ne contient pas de condition WHERE
Pour moi cela ne presente pas de probleme car on peux utiliser cette condition where par la suite dans la requete SQL

La jointure naturelle n'est pas la plus utilisée, mais la syntaxe est très économique quand le schéma relationnel s'y prête.

B. L'approche « produit cartésien »

Qu'est-ce que le produit cartésien de deux tables ?

Soient deux tables T et T' :

T

A	B
6	6
7	7

T'

B'	C
9	1
6	8

Le produit cartésien (noté ⋈ en algèbre relationnel) des deux tables T et T'est la table :

T ⋈ T'

A	B	B'	C
6	6	9	1
6	6	6	8
7	7	9	1
7	7	6	8

Dans ce cas, puisqu'il n'y a aucun attribut en commun, nous ne pouvons joindre aucun n-uplet de la première table avec aucun n-uplet de la deuxième table. Par conséquent, n'importe quelle combinaison va donner un résultat.
Ici, nous avons :

- (A , B) ∩ (B', C) = ∅
- Produit cartésien T × T' = T ⋈ T'
- taille(T ⋈ T') = taille(T) × taille(T') ➔ 16 = 4 x 4

Dans une jointure de type « produit cartésien » , le nombre de n-uplets résultats, c'est le nombre de n-uplets de la première table T multiplié par celui de la deuxième T'. Nous avons :

- type(T) ∩ type(T') = ∅
- Produit cartésien T × T' = T ⋈ T'
- taille(T ⋈ T') = taille(T) × taille(T')

Dans un produit cartésien, chaque n-uplet de T est associé à tous les n-uplets de T'.

A faire vous-même XI

Illustrons ceci au sein de notre base des « Voyageurs » et plus précisément avec les tables « Logement » et « Activité »

a. En vous référant aux notions vues précédemment, pouvez-vous joindre « naturellement » les tables « Logement » et « Activité » ? Expliquez pourquoi.

```
Non car ils n'ont pas d'attribut en commun
```

b. Dans l'onglet SQL de la table « Logement », saisir la requête suivante :

```
select * from Logement cross join Activité
```

Le mot clé « cross » nous indique qu'il s'agit d'une jointure de type « produit cartésien »

code	nom	capacité	type	lieu	codeLogement	codeActivité	description
ca	Causses	45	Auberge	Cévennes	ca	Randonnée	Sorties d'une journée en groupe
el	Elbazet	50	Auberge	Occitanie	ca	Randonnée	Sorties d'une journée en groupe
ge	Génépi	134	Hôtel	Alpes	ca	Randonnée	Sorties d'une journée en groupe
pi	U Pinzutu	10	Gîte	Corse	ca	Randonnée	Sorties d'une journée en groupe
ta	Tabriz	34	Hôtel	Bretagne	ca	Randonnée	Sorties d'une journée en groupe
ca	Causses	45	Auberge	Cévennes	ge	Piscine	Nage loisir non encadrée
el	Elbazet	50	Auberge	Occitanie	ge	Piscine	Nage loisir non encadrée
ge	Génépi	134	Hôtel	Alpes	ge	Piscine	Nage loisir non encadrée
pi	U Pinzutu	10	Gîte	Corse	ge	Piscine	Nage loisir non encadrée
ta	Tabriz	34	Hôtel	Bretagne	ge	Piscine	Nage loisir non encadrée
ca	Causses	45	Auberge	Cévennes	ge	Ski	Sur piste uniquement
el	Elbazet	50	Auberge	Occitanie	ge	Ski	Sur piste uniquement
ge	Génépi	134	Hôtel	Alpes	ge	Ski	Sur piste uniquement
pi	U Pinzutu	10	Gîte	Corse	ge	Ski	Sur piste uniquement
ta	Tabriz	34	Hôtel	Bretagne	ge	Ski	Sur piste uniquement
ca	Causses	45	Auberge	Cévennes	pi	Plongée	Baptèmes et préparation des brevets
el	Elbazet	50	Auberge	Occitanie	pi	Plongée	Baptèmes et préparation des brevets
ge	Génépi	134	Hôtel	Alpes	pi	Plongée	Baptèmes et préparation des brevets
pi	U Pinzutu	10	Gîte	Corse	pi	Plongée	Baptèmes et préparation des brevets
ta	Tabriz	34	Hôtel	Bretagne	pi	Plongée	Baptèmes et préparation des brevets
ca	Causses	45	Auberge	Cévennes	pi	Voile	Pratique du dériveur et du catamaran
el	Elbazet	50	Auberge	Occitanie	pi	Voile	Pratique du dériveur et du catamaran
ge	Génépi	134	Hôtel	Alpes	pi	Voile	Pratique du dériveur et du catamaran
pi	U Pinzutu	10	Gîte	Corse	pi	Voile	Pratique du dériveur et du catamaran
ta	Tabriz	34	Hôtel	Bretagne	pi	Voile	Pratique du dériveur et du catamaran

Analyse du résultat de la requête

c. Quelle est la taille (en nombre de n-uplets) de la table « résultat » ? Justifier cette taille.

```
Taille = Taille(Activité) * Taille(Logement) = 5*4 = 20 n-uplets
```

d. En vous appuyant sur la définition de la forme de base d'une requête SQL (§3.4.3 b.), déduire de la réponse précédente (c.) la nature de l'espace de recherche. Interpréter cette nature.

```
La nature de l'espace de recherche est Logement
```

e. En choisissant un exemple dans « Logement », montrer que chaque n-uplet de "Logement" est associé à tous les n-uplets de « Activité ». Construire et tester la requête qui illustre la vérification.

code	nom	capacité	type	lieu	codeLogement	codeActivité	description
ca	Causses	45	Auberge	Cévennes	ca	Randonnée	Sorties d'une journée en groupe
el	Elbazet	50	Auberge	Occitanie	ca	Randonnée	Sorties d'une journée en groupe
ge	Génépi	134	Hôtel	Alpes	ca	Randonnée	Sorties d'une journée en groupe
pi	U Pinzutu	10	Gîte	Corse	ca	Randonnée	Sorties d'une journée en groupe
ta	Tabriz	34	Hôtel	Bretagne	ca	Randonnée	Sorties d'une journée en groupe

```
SELECT * FROM `Logement` cross join Activité as A where A.codeLogement
= "ca"
```

f. Considérons les informations issues de la jointure « produit cartésien ». Est-ce que tous les n-uplets nous intéressent ? Par exemple, est-ce intéressant de rapprocher un logement qui est une auberge, "Elbazet", avec une randonnée qui a lieu dans un autre hôtel ? Est-ce intéressant de rapprocher un logement qui est un hôtel, "Génépi", avec l'activité « Ski » ?

Tout ces n-uplets ne sont pas utiles, il nous faut seulemnt ceux dont l'activité est au meme endroit que le logement cest a dire ceux dont l'atribut code dans la tables Logement est egal au codeLogement de la table Activité

g. Ce qui nous intéresse, a priori, c'est d'obtenir les logements et les activités qui ont lieu dans ces logements. En considérant cette affirmation et en vous appuyant sur la définition de la forme de base d'une requête SQL (§3.4.3 b.), saisir la requête SQL qui nous permette d'obtenir ce qui nous intéresse. Ecrire le résultat de cette requête.

code	nom	capacité	type	lieu	codeLogement	codeActivité	description
ca	Causses	45	Auberge	Cévennes	ca	Randonnée	Sorties d'une journée en groupe
ge	Génépi	134	Hôtel	Alpes	ge	Piscine	Nage loisir non encadrée
ge	Génépi	134	Hôtel	Alpes	ge	Ski	Sur piste uniquement
pi	U Pinzutu	10	Gîte	Corse	pi	Plongée	Baptèmes et préparation des brevets
pi	U Pinzutu	10	Gîte	Corse	pi	Voile	Pratique du dériveur et du catamaran

```
SELECT * FROM `Logement` as L cross join Activité as A where L.code = A.codeLogement
```

h. Ecrire la requête qui améliore la lisibilité du résultat en ne sélectionnant que les attributs nom, lieu et description.

```
SELECT L.nom,L.lieu,A.description FROM `Logement` as L cross join Activité as A where L.code = A.codeLogement
```

nom	lieu	description
Causses	Cévennes	Sorties d'une journée en groupe
Génépi	Alpes	Nage loisir non encadrée
Génépi	Alpes	Sur piste uniquement
U Pinzutu	Corse	Baptèmes et préparation des brevets
U Pinzutu	Corse	Pratique du dériveur et du catamaran

A retenir :
Afin de réaliser une jointure, nous complétons un produit cartésien en exprimant le critère de rapprochement et d'égalité entre deux attributs qui viennent de deux tables différentes.

En général, ce ne sont pas n'importe quels attributs. Ces attributs figurent dans le schéma de la base en tant que :

- clé primaire d'une part (dans notre exemple, l'attribut « code » dans « Logement »)
- clé étrangère d'autre part (dans notre exemple, l'attribut « codeLogement » dans « Activité »). Cette clé étrangère, c'est la valeur, l'identifiant d'un logement, qui a été placé dans les activités.

Si besoin, n'hésitez pas à reprendre le § 3.4.2 afvm VI e. afin de vous approprier le concept de clé étrangère.

Afin de bien appréhender l'opération essentielle de jointure dans des bases relationnelles, il faut considérer toutes les combinaisons possibles (conceptuellement), et en sélectionner certaines afin d'obtenir le résultat recherché.

Deux remarques :

- SQL nous permet de réaliser des jointures sur plusieurs tables. Il suffit pour cela de les énumérer dans le « from » Dans ce cas, il est fréquent d'employer le renommage « as » afin de gérer les éventuelles ambiguïtés soulevées par les noms d'attributs.
- SQL nous permet de réaliser des « CROSS JOIN » de « CROSS JOIN ». On a l'habitude d'alléger l'écriture de telles requêtes en remplaçant l'expression « CROSS JOIN » par une virgule « , ».

i. Cas où la jointure concerne plusieurs tables. Dans l'onglet SQL de la table « Logement », saisir la requête suivante :

```
select V.nom as nomVoyageur, L.nom as nomLogement
from Logement as L, Séjour as S, Voyageur as V
where code = S.codeLogement
and S.idVoyageur = V.idVoyageur
```

Ecrire le résultat de cette requête
Quelles sont les tables qu'on associe dans cette jointure ?
Quels sont les critères de rapprochement des n-uplets ?
Finalement à quelle question répond-on en soumettant cette requête ?

```
Les tables sont Logement,Séjour,Voyageur

Les criteres de rapprochement sont :
- le code de Logement est egal au
codeLogement de la table Séjour
- ET que idVoyageur de la table Séjour est
egal a idVoyageur de la table Voyageur

Grace a cette requette nous pouvons voir ou
les voyageurs sont partis en vacances
```

nomVoyageur	nomLogement
Fogg	U Pinzutu
Bouvier	Tabriz
David-Néel	Génépi
Bouvier	U Pinzutu
Bouvier	Génépi
Fogg	U Pinzutu
David-Néel	Causses
Bouvier	Causses

j. On souhaite maintenant ordonner l'affichage du résultat de la requête en classant les voyageurs par ordre alphabétique. Afin d'atteindre un tel objectif, SQL propose l'utilisation de la clause « ORDER BY » suivie d'un nom de colonne. Modifiez la requête précédente afin d'obtenir le résultat suivant :

nomVoyageur	nomLogement
Bouvier	U Pinzutu
Bouvier	Causses
Bouvier	Génépi
Bouvier	Tabriz
David-Néel	Génépi
David-Néel	Causses
Fogg	U Pinzutu
Fogg	U Pinzutu

```
select V.nom as nomVoyageur, L.nom as nomLogement from Logement as L,
Séjour as S, Voyageur as V where code = S.codeLogement and S.idVoyageur
= V.idVoyageur ORDER BY nomVoyageur
```

k. Le résultat comporte un doublon. Expliquez cet affichage à partir de l'analyse des tables incluses dans l'espace de recherche de la requête.

```
Le doublon est du au fait que Fogg soit parti 2 fois a U Pinzutu
```

l. Pour éviter des redondances dans les résultats, il faut simplement ajouter DISTINCT après le mot SELECT. Ecrire et tester la requête qui permet d'obtenir le résultat suivant :

nomVoyageur	nomLogement
Bouvier	Tabriz
Bouvier	Causses
Bouvier	U Pinzutu
Bouvier	Génépi
David-Néel	Causses
David-Néel	Génépi
Fogg	U Pinzutu

```
select DISTINCT V.nom as nomVoyageur, L.nom as nomLogement from
Logement as L, Séjour as S, Voyageur as V where code = S.codeLogement
and S.idVoyageur = V.idVoyageur ORDER by nomVoyageur
```

A retenir
La jointure est obtenue en effectuant un produit cartésien dans le « from » :
• Tout se passe comme si on interrogeait une seule table, celle définie par table1 X table2
• Le critère de jointure est exprimé dans le « where », avec les critères (éventuels) de sélection
• Les synonymes (attributs ou tables) entrainent des ambigüités, et nécessitent des renommages

3.4.5 Mise à jour d'une base de données

Au cours des paragraphes précédents, nous avons appris à interroger une base de données avec le langage SQL. Nous allons maintenant apprendre à mettre à jour une base de données en utilisant ce même langage SQL.

Les opérations de « Mise à jour » d'une table utilisent principalement trois opérations :

- la mise à jour (Commande UPDATE),
- l'insertion (Commande INSERT INTO),
- la suppression (Commande DELETE),

d'un n-uplet.

Afin de réaliser ces opérations, nous allons utiliser la ligne de commande du SGBD. Certes, l'interface est plus austère que celle de PMA mais elle est idéale afin de bien comprendre le fonctionnement du SGBD ainsi que le sens des commandes utilisées.

A faire vous-même XII

Nous souhaitons mettre à jour la base de données des « Voyageurs » ainsi :

- Créer le voyageur « Enzo Ciriesse » de Metz dans le Grand Est.
- Mettre à jour la capacité du logement « Causses » à 50 personnes.

a. Connexion à la base de données.

Ouvrir une fenêtre « Terminal X » en cliquant sur l'icone « LXTerminal ». Puis accédez au SGDB en tant qu'utilisateur « root » :

```
pi@raspberry:~ $ mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 16
Server version: 10.3.22-MariaDB-0+deb10u1 Debian 10

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

b. Sélectionner la base de données des « Voyageurs » en utilisant la commande « USE » :

« USE Voyageurs ; »

Que remarquez-vous au niveau du prompt ?

```
Le prompt est passé de MariaDB[(none)] a MariaDB[Voyageurs]
Ce qui indique que nous sommes dans la base de donnée Voyageurs
```

c. Avant d'insérer un enregistrement dans la table « Voyageur », nous devons :
   - Vérifier que cette table fait bien partie de la BDD,
   - Afficher le contenu de la table (liste des n-uplets)

Afin de lister le contenu de la BDD, utiliser la commande :

« SHOW tables ; »

Ensuite, lister les n-uplets de la table avec la commande de base : « select * from <nom_de_table> ;». Combien de n-uplets obtenez-vous ? Est-ce correct ? Dans quelle ville habite Mr Stevenson ?

```
On obtient 4 n-uplets
Mr Stevenson habite a Vannes en Bretagne
```

d. L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO. Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup. Sa syntaxe est la suivante :

« INSERT INTO table VALUES ('valeur 1', 'valeur 2', ...) ; »

Créer le voyageur « Enzo Ciriesse » de Metz dans le Grand Est avec l'attribut « idVoyageur » renseigné avec la valeur « 50 ». Vérifiez l'insertion à l'aide de la commande « select … »

```
INSERT INTO Voyageur VALUES(50,"Ciriesse","Enzo","Metz","Grand Est");
```

```
MariaDB [Voyageurs]> select * from Voyageur;
+------------+-------------+-------------+----------+-----------+
| idVoyageur | nom         | prénom      | ville    | région    |
+------------+-------------+-------------+----------+-----------+
|         10 | Fogg        | Phileas     | Ajaccio  | Corse     |
|         20 | Bouvier     | Nicolas     | Aurillac | Auvergne  |
|         30 | David-Néel  | Alexandra   | Lhassa   | Tibet     |
|         40 | Stevenson   | Robert Louis| Vannes   | Bretagne  |
|         50 | Ciriesse    | Enzo        | Metz     | Grand Est |
+------------+-------------+-------------+----------+-----------+
```

e. La commande « UPDATE » permet d'effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec « WHERE » pour spécifier sur quelles lignes doivent porter la ou les modifications. Sa syntaxe est la suivante :

```
UPDATE table
SET nom_colonne_1 = 'nouvelle valeur'
WHERE condition
```

Après avoir lister le contenu de la table « Logement », mettre à jour la capacité du logement « Causses » à 50 personnes. Vérifiez alors l'effet de la commande à l'aide de la commande « select … »

```
UPDATE Logement SET capacité = 50 WHERE  nom = "Causses";
```

```
+------+-----------+-----------+---------+-----------+
| code | nom       | capacité  | type    | lieu      |
+------+-----------+-----------+---------+-----------+
| ca   | Causses   |        50 | Auberge | Cévennes  |
| ge   | Génépi    |       134 | Hôtel   | Alpes     |
| pi   | U Pinzutu |        10 | Gîte    | Corse     |
| ta   | Tabriz    |        34 | Hôtel   | Bretagne  |
+------+-----------+-----------+---------+-----------+
```

A faire vous-même XIII

a. Mettre à jour la Base des Voyageurs selon les critères suivants :
   - Créer le gîte « Quercy » (code « qu ») d'une capacité de 30 places en Occitanie
   - Créer le séjour « 9 » de Mr Ciriesse dans le gîte « Quercy » du 30 Avril au 6 mai 2019
   - Mettre à jour la date (le 21) de fin de séjour de Nicolas BOUVIER dans le logement Tabriz

```
INSERT INTO Logement VALUES("qu","Quercy",30,"Gîte","Occitanie");

INSERT INTO Séjour VALUES(9,50,"qu",30,6);

UPDATE Séjour SET fin = 21 WHERE idVoyageur = 20 AND codeLogement = "ta";
```

b. Ecrire et tester les requêtes qui vous permettront de répondre aux questions suivantes :

Quels sont les logements fréquentés par Nicolas BOUVIER ?

Aide :

- Avant d'écrire la requête, affichez le contenu des tables qui définissent l'espace de recherche
- Utilisez les jointures entre tables afin de les lier entre elles
- Le résultat est le suivant :

nomVoyageur	nomLogement
Bouvier	Causses
Bouvier	Génépi
Bouvier	U Pinzutu
Bouvier	Tabriz

```
SELECT DISTINCT V.nom as nomVoyageur, L.nom as nomLogement from
Voyageur as V CROSS JOIN Logement as L,Séjour as S where S.idVoyageur
= V.idVoyageur AND L.code = S.codeLogement and V.idVoyageur = 20
```

A quelle(s) date(s) ont eu lieu le(s) voyage(s) d'Alexandra David-Néel ?

Aide :

- Inspirez-vous de la requête précédente
- Le résultat est le suivant :

nomVoyageur	nomLogement	JourDébut	JourFin
David-Néel	Génépi	2	3
David-Néel	Causses	13	18

```
select DISTINCT V.nom as nomVoyageur, L.nom as nomLogement,S.début as
JourDébut,S.fin as JourFin from Voyageur as V cross join Logement
as L, Séjour as S where S.idVoyageur = V.idVoyageur and L.code =
S.codeLogement and V.idVoyageur = 30;
```

Quel(s) voyageurs a(ont) effectué des séjours sans nuitée et dans quel(s) logement(s) ?
Aide :

- le résultat est le suivant :

nomVoyageur	nomLogement	JourDébut	JourFin
Fogg	U Pinzutu	20	20
Bouvier	Tabriz	21	21

```
select DISTINCT V.nom as nomVoyageur, L.nom as nomLogement,S.début as JourDébut,S.fin as JourFin from Voyageur as V cross join Logement as L, Séjour as S where S.idVoyageur = V.idVoyageur and L.code = S.codeLogement and S.début = S.fin;
```

***** Fin du document *****