SQL

Table des matières

SQL (Structured Query Language) est un langage informatique servant à exploiter des bases de données relationnelles. La partie langage de manipulation des données de SQL permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans les bases de données relationnelles.

1. Introduction

1.1. Base de données

Une base de données informatique est un ensemble de données qui ont été stockées sur un support informatique, et organisées et structurées de manière à pouvoir facilement consulter et modifier leur contenu.

Il ne suffit pas que la base de données existe, il faut aussi pouvoir la gérer, interagir avec cette base. Il faut pouvoir envoyer des message (messages qu'on appellera "requêtes"), afin de pouvoir ajouter des news, modifier des membres, supprimer, et tout simplement afficher des éléments de la base.

Une base de données seule ne suffit donc pas, il est nécessaire d'avoir également :

- un système permettant de gérer cette base
- un langage pour transmettre des instructions à la base de données

Un Système de Gestion de Base de Données (SGBD) est un logiciel (ou un ensemble de logiciels) permettant de manipuler les données d'une base de données. Manipuler, c'est-à-dire sélectionner et afficher des informations tirées de cette base, modifier des données, en ajouter ou en supprimer (ce groupe de quatre opérations étant souvent appelé "CRUD", pour Create, Read, Update, Delete).

La plupart des SGBD sont basés sur un modèle Client - Serveur. C'est-à-dire que la base de données se trouve sur un serveur qui ne sert qu'à ça, et pour interagir avec cette base de données, il faut utiliser un logiciel "client" qui va interroger le serveur et transmettre la réponse que le serveur lui aura donnée.

Par conséquent, lorsque vous installez un SGBD basé sur ce modèle, vous installez en réalité deux choses (au moins) : le serveur, et le client. Chaque requête (insertion/modification/lecture de données) est faite par l'intermédiaire du client. Jamais vous ne discuterez directement avec le serveur (d'ailleurs, il ne comprendrait rien à ce que vous diriez).

Un SGBDR est un SGBD qui implémente la théorie relationnelle. Les données sont contenues dans des relations, qui sont représentées sous forme de tables. Une relation est composée de deux parties, l'en-tête et le corps. L'en-tête est lui-même composé de plusieurs attributs.

Exemple, pour la relation "Client", on crée l'en-tête suivant : Numéro, Prénom, Nom, Email

Quant au corps, il s'agit d'un ensemble de lignes (ou n-uplets) composées d'autant d'éléments qu'il y a d'attributs dans le corps. Voici quatre lignes pour la relation "Client" :

Numéro	**Prénom**	**Nom**	**Email**
1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com

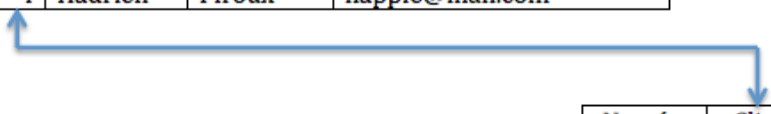
Différentes opérations peuvent alors être appliquées à ces relations, ce qui permet d'en tirer des informations. Parmi les opérations les plus utilisées, on peut citer :

- la sélection (ou restriction) : obtenir des lignes répondant à certains critères
- la projection : obtenir une partie des attributs des lignes

- l'union : obtenir tout ce qui se trouve dans la relation A ou dans la relation B
- l'intersection : obtenir tout ce qui se trouve à la fois dans la relation A et dans la relation B
- la différence : obtenir ce qui se trouve dans la relation A mais pas dans la relation B
- la jointure : obtenir l'ensemble des lignes provenant de la liaison de la relation A et de la relation B à l'aide d'une information commune.

Un petit exemple pour illustrer la jointure : si l'on veut stocker des informations sur les clients d'une société, ainsi que les commandes passées par ces clients, on utilisera deux relations : client et commande, la relation commande étant liée à la relation client par une référence au client ayant passé commande.

Numéro	*Prénom*	*Nom*	*Email*
1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	jacques.nicolas@email.com
4	Hadrien	Piroux	happie@mail.com



Numéro	*Client*	*Produit*	*Quantité*
1	3	Tube de colle	3
2	2	Rame de papier A4	6
3	2	Ciseaux	2

Le client numéro 3, M. Nicolas Jacques, a donc passé une commande de trois tubes de colle, tandis que Mme Marie Malherbe (cliente numéro 2) a passé deux commandes, pour du papier et des ciseaux.

1.2. Le langage SQL

Le SQL (Structured Query Language[1]) est un langage informatique qui permet d'interagir avec des bases de données relationnelles. Il a été créé dans les années 1970 et c'est devenu standard en 1986 (pour la norme ANSI - 1987 en ce qui concerne la norme ISO). Il est encore régulièrement amélioré.

1.3. MySQL

MySQL est un SGBDR, qui utilise le langage SQL. C'est un des SGBDR les plus utilisés car popularité est due en grande partie au fait qu'il s'agit d'un logiciel Open Source.

Le développement de MySQL commence en 1994 par David Axmark et Michael Widenius. En 2008, MySQL AB est rachetée par la société Sun Microsystems, qui est elle-même rachetée par Oracle Corporation en 2010.

MySQL ne suit pas toujours la norme officielle. Certaines syntaxes peuvent donc être propres à MySQL et ne pas fonctionner sous d'autres SGBDR. Par ailleurs, il n'implémente pas certaines fonctionnalités avancées, qui peuvent être utiles pour un projet un tant soit peu ambitieux.

Il existe des dizaines de SGBDR, chacun ayant ses avantages et ses inconvénients. Voici

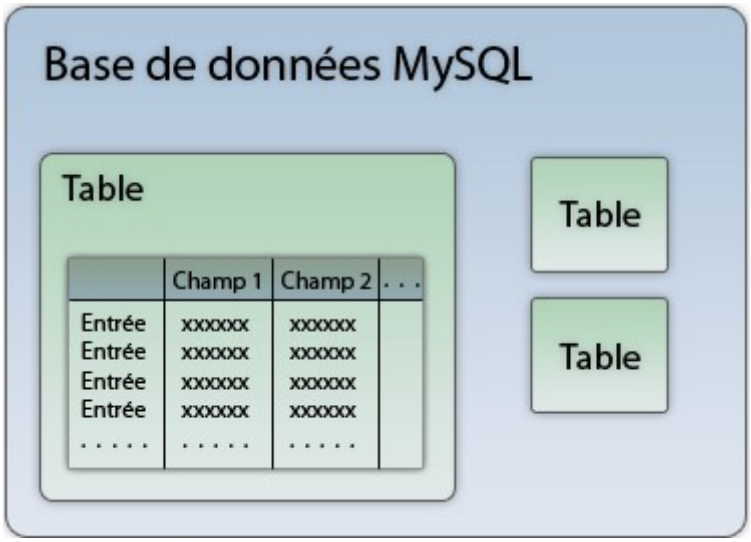
1 langage de requête structurée

succinctement quatre d'entre eux, parmi les plus connus :

- Oracle database, édité par Oracle Corporation.
- PostgreSQL, logiciel Open Source qui a longtemps été disponible uniquement sous Unix.
- MS Access, édité par Microsoft qui ne fonctionne que sous Windows et qui n'est pas adapté pour gérer un grand volume de données.
- SQLite, ce logiciel stocke toutes les données dans de simples fichiers.

1.4. Organisation d'une base de données

Les données sont représentées sous forme de tables. Une base va donc contenir plusieurs tables qui définissent un certain nombre de champs, qui sont les caractéristiques de l'objet représenté par la table (les attributs de l'en-tête dans la théorie relationnelle). Chaque donnée introduite le sera sous forme d'entrées dans une table, définissant la valeur de chaque champ pour cette donnée.



Exemple : un champ "Nom", un champ "Prénom", un champ "Email" et un champ "Numéro" permettent d'identifier les clients individuellement.

Numéro	**Prénom**	**Nom**	**Email**
1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com

2. Les types de données

Un mauvais type de données peut entraîner :

- un gaspillage de mémoire.
- des problèmes de performance.
- un comportement contraire à celui attendu.
- l'impossibilité d'utiliser des fonctionnalités propres à un type de données.

2.1. Types numériques

2.1.1. Nombres entiers

Les types de données qui acceptent des nombres entiers comme valeur sont désignés par le mot-clé INT, et ses déclinaisons :

Type	Nombre d'octets	Minimum	Maximum
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUMINT	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	-9223372036854775808	9223372036854775807

L'attribut UNSIGNED permet de travailler avec des valeurs non signées : 0-255 pour TINYINT, 65535 pour SMALLINT, etc...

2.1.2. Nombres décimaux

Cinq mots-clés permettent de stocker des nombres décimaux dans un champ : DECIMAL, NUMERIC, FLOAT, REAL et DOUBLE.

NUMERIC et DECIMAL sont équivalents et acceptent deux paramètres : la précision et l'échelle.

- La précision définit le nombre de chiffres significatifs stockés (les 0 à gauche ne comptent pas).
- L'échelle définit le nombre de chiffres après la virgule.

Dans un champ DECIMAL(5,3), on peut donc stocker des nombres de 5 chiffres significatifs maximum, dont 3 chiffres sont après la virgule. Par exemple : 12.354, -54.258, 89.2 ou -56.

FLOAT et REAL utilisent quatre octets pour stocker les valeurs du champ.

DOUBLE utilise 8 octets de stockage.

Remarque : MySQL utilise 8 octets pour REAL et DOUBLE. Utilisez DOUBLE pour éviter les surprises en cas de changement de SGBDR.

Les nombres stockés en tant que NUMERIC ou DECIMAL sont stockés sous forme de chaînes de caractères. Par conséquent, c'est la valeur exacte qui est stockée. Par contre, les types FLOAT, DOUBLE et REAL sont stockés sous forme de nombres, et c'est une valeur approchée qui est stockée. Cela peut poser problème pour des comparaisons.

2.2. Types alphanumériques

CHAR et VARCHAR sont utilisés pour stocker un texte de moins de 256 octets. Un CHAR(x) stockera toujours x octets, tandis qu'un VARCHAR(x) stockera jusqu'à x octets (entre 0 et x) avec la taille du texte stocké.

Pour des textes supérieur à 255 caractères utilisez le type TEXT, ou un de ses dérivés.

Type	Longueur maximale	Mémoire occupée

TINYTEXT	2^8 octets	Longueur de la chaîne + 1 octet
TEXT	2^{16} octets	Longueur de la chaîne + 2 octets
MEDIUMTEXT	2^{24} octets	Longueur de la chaîne + 3 octets
LONGTEXT	2^{32} octets	Longueur de la chaîne + 4 octets

2.3. Types énumérés

Un champ de type ENUM est un champ pour laquelle on définit un certain nombre de valeurs autorisées, de type "chaîne de caractère". Par exemple, si l'on définit un champ espece (pour une espèce animale) de la manière suivante :

especе ENUM('chat', 'chien', 'tortue')

Le champ espece pourra alors contenir les chaînes : "chat", "chien" ou "tortue", mais pas les chaînes "lapin" ou "cheval".

Pour remplir un champ de type ENUM, deux possibilités s'offrent à vous :

- soit remplir directement avec la valeur choisie ("chat", "chien" ou "tortue" dans notre exemple).
- soit utiliser l'index de la valeur, c'est-à-dire le nombre associé par MySQL à la valeur. Ce nombre est compris entre 1 et le nombre de valeurs définies. L'index est attribué selon l'ordre dans lequel les valeurs ont été données lors de la création du champ (une chaîne vide correspond à l'index 0).

Un ENUM peut avoir maximum 65535 valeurs possibles

Un champ de type SET est un champ qui permet de stocker entre 0 et x valeur(s), x étant le nombre de valeurs autorisées.

Donc, si l'on définit un champ de type SET de la manière suivante :

especе SET('chat', 'chien', 'tortue')

Le champ espece pourra alors contenir les chaînes : 'chat', 'chat,tortue', 'chat,chien,tortue', 'chien,tortue', ...

Les valeurs autorisées d'un champ SET ne peuvent pas contenir de virgule elles-mêmes ni stocker la même valeur plusieurs fois.

Les champs SET utilisent également un système d'index binaire. La présence/absence des valeurs autorisées va être enregistrée sous forme de bits, mis à 1 si la valeur correspondante est présente, à 0 si la valeur correspondante est absente.

Dans l'exemple, il faut donc trois bits pour savoir quelles valeurs sont stockées dans le champ :

- 000 signifie qu'aucune valeur n'est présente.
- 001 signifie que 'chat' est présent.
- 100 signifie que 'tortue' est présent.
- 110 signifie que 'chien' et 'tortue' sont présents.
- ...

Un champ de type SET peut avoir au plus 64 valeurs définies

Remarque : SET et ENUM sont des types propres à MySQL car MySQL n'implémentent pas les contraintes d'assertions utilisées dans la plupart des SGBD.

2.4. Types temporels

Les cinq types temporels sont DATE, DATETIME, TIME, TIMESTAMP et YEAR.

DATE, TIME et DATETIME

Pour entrer une DATE, il faut donner d'abord l'année (deux ou quatre chiffres), ensuite le mois (deux chiffres) et pour finir, le jour (deux chiffres), sous forme de nombre ou de chaîne de caractères. S'il s'agit d'une chaîne de caractères, n'importe quelle ponctuation peut être utilisée pour délimiter les parties (ou aucune). Exemples :

- 'AAAA-MM-JJ' (c'est sous ce format-ci qu'une DATE est stockée)
- 'AAMMJJ'
- 'AAAA/MM/JJ'
- AAAAMMJJ (nombre)
- AAMMJJ (nombre)

MySQL supporte des DATE allant de '1001-01-01' à '9999-12-31'

DATETIME permet de stocker une heure, en plus d'une date. Pour la date, année-mois-jour, et pour l'heure, il faut donner d'abord l'heure, ensuite les minutes, puis les secondes. Exemples :

- 'AAAA-MM-JJ HH:MM:SS' (c'est sous ce format-ci qu'un DATETIME est stocké)
- 'AA*MM*JJ HH+MM+SS'
- AAAAMMJJHHMMSS (nombre)

MySQL supporte des DATETIME allant de '1001-01-01 00:00:00' à '9999-12-31 23:59:59'

Le type TIME permet non seulement de stocker une heure précise, mais aussi un intervalle de temps. Il faut d'abord donner l'heure, puis les minutes, puis les secondes, chaque partie pouvant être séparée des autres par le caractère :. Exemples :

- 'HH:MM:SS'
- 'HHH:MM:SS'
- 'MM:SS'
- 'HHMMSS'
- HHMMSS (nombre)

MySQL supporte des TIME allant de '-838:59:59' à '838:59:59'

YEAR est un type intéressant car il ne prend qu'un seul octet en mémoire : on ne peut y stocker que des années entre 1901 et 2155. On peut entrer une donnée de type YEAR sous forme de chaîne de caractères ou d'entiers, avec 2 ou 4 chiffres.

Par définition, le TIMESTAMP d'une date est le nombre de secondes écoulées depuis le 1er janvier 1970, 0h0min0s (TUC) et la date en question. Le type TIMESTAMP de SQL est cependant un peu particulier car il ne sert pas à stocker un nombre de secondes, mais bien une date sous format numérique AAAAMMJJHHMMSS (c'est-à-dire l'équivalent, au format numérique, du

DATETIME).

Il n'est donc pas possible de stocker un "vrai" timestamp dans un champ de type TIMESTAMP. C'est évidemment contre-intuitif, et source d'erreur.

Malgré cela, le TIMESTAMP SQL a les même limites qu'un vrai timestamp : il n'acceptera que des dates entre le 1e janvier 1970 à 00h00min00s et le 19 janvier 2038 à 3h14min7s.

La date par défaut

Lorsque MySQL rencontre une date/heure incorrecte, ou qui n'est pas dans l'intervalle de validité du champ, la valeur par défaut est stockée à la place. Il s'agit de la valeur "zéro" du type

Type	**Date par défaut ("zéro")**
DATE	'0000-00-00'
DATETIME	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000
TIMESTAMP	00000000000000

3. Création d'une base de données

Ne jamais utiliser d'espaces ou d'accents dans les noms de bases, tables ou champs ni de mots-clés SQL. Une convention largement répandue veut que les commandes et mots-clés SQL soient écrits complètement en majuscules, les noms de bases, tables et champs sont écrits en minuscules.

Lees commandes SQL à utiliser pour interagir avec les bases de données ont des options facultatives, dans ces cas-là on utilisera des crochets [ ] pour indiquer ce qui est facultatif.

3.1. Création d'une base de données

La commande SQL pour créer une base de données est la suivante :

CREATE DATABASE nom_base;

Cependant, il faut également définir l'encodage utilisé (l'UTF-8 dans notre cas). Voici donc la commande complète à taper pour créer votre base :

CREATE DATABASE nom_base CHARACTER SET 'utf8';

3.2. Suppression d'une base de données

Soyez très prudents, car vous effacez tous les fichiers créés par MySQL qui servent à stocker les informations de votre base.

DROP DATABASE nom_base;

Si vous essayez cette commande alors que la base de données nom_base n'existe pas, MySQL vous affichera une erreur :

```
mysql> DROP DATABASE elevage;
ERROR 1008 (HY000) : Can't drop database 'nom_base'; database doesn't exist
mysql>
```

Pour éviter ce message d'erreur, vous pouvez utiliser l'option IF EXISTS, de la manière suivante :

```
DROP DATABASE IF EXISTS nom_base;
```

Pour afficher les warnings de MySQL, il faut utiliser la commande

```
SHOW WARNINGS;
```

Cette commande affiche un tableau :

Level	**Code**	**Message**
Note	1008	Can't drop database 'elevage'; database doesn't exist

3.3. Utilisation d'une base de données

Une fois créé la base de données, il faut la sélectionner pour pouvoir agir sur cette base :

```
USE nom_base;
```

À partir de maintenant, toutes les actions effectuées le seront sur la base de données nom_base.

Notez que vous pouvez spécifier la base de données sur laquelle vous allez travailler lors de la connexion à MySQL :

```
mysql -u nom_utilisateur -p  nom_base
```

4. Création de tables

Pour commencer, il faudra définir de quels champs (et leur type) la table sera composée. Cette étape est la plus importante. Dans les exemples ci-dessous, nous allons créer une base de données pour gérer un élevage d'animaux d'un laboratoire de biologie.

4.1. Définition des champs

Avant de choisir le type des champs, il faut définir les champs :

- Espèce : on peut caractériser l'espèce par un ou plusieurs mots. Ce sera donc un champ de type alphanumérique.
- Les noms d'espèces sont relativement courts, mais n'ont pas tous la même longueur. On choisira donc un VARCHAR jusqu'à 40 caractères.
- Sexe : deux choix possibles (mâle ou femelle). Il serait possible d'utiliser un ENUM. Cependant, ENUM reste un type non standard. Pour cette raison, nous utiliserons plutôt un champ CHAR(1), contenant soit 'M' (mâle), soit 'F' (femelle).
- Date de naissance : l'heure de la naissance étant importante pour les soins lors des premiers jours, on choisira un DATETIME.
- Commentaires : type alphanumérique de tyepe TEXT.
- Nom : on prendra un VARCHAR(30).

NULL or NOT NULL ?

Il faut maintenant déterminer si l'on autorise les champs à ne pas stocker de valeur (ce qui est donc représenté par NULL).

- Espèce : un éleveur connaît l'espèce des animaux qu'il élève. On n'autorisera donc pas la champ espece à être NULL.
- Sexe : le sexe de certains animaux est très difficile à déterminer à la naissance. Il n'est donc pas impossible qu'on doive attendre plusieurs semaines. Par conséquent, la champ sexe peut contenir NULL.
- Date de naissance : pour garantir la pureté des races, on ne travaille qu'avec des individus dont on connaît la provenance. Ce champ ne peut donc pas être NULL.
- Commentaires : ce champ peut très bien ne rien.
- Nom : il est parfois difficile d'inventer des noms pour des espèces ayant une nomreuse portée. Il vaut mieux autoriser ce champ à être NULL.

4.2. Introduction aux clés primaires

Comment différencier deux animaux de la même espèce, du même sexe, nés le même jour ? Ils auront donc exactement les mêmes caractéristiques, pourtant, ce ne sont pas les mêmes individus. Il faut donc les différencier. Pour cela, on va ajouter un champ à notre table et donner à chaque animal un numéro d'identité

Le champ qu'on ajoutera s'appellera donc id, et il s'agira d'un INT, toujours positif donc UNSIGNED. Ce champ ne pourra bien sûr pas être NULL, sinon il perdrait toute son utilité.

4.2.1. Clé primaire

La clé primaire d'une table est une contrainte d'unicité, composée d'une ou plusieurs champs. La clé primaire d'une entrée permet d'identifier de manière unique cette entrée dans la table. Lorsqu'une table possède une clé primaire, celle-ci doit être définie (elle correspond au numéro d'identité dont nous venons de parler). Nous définirons donc id comme la clé primaire de la table, en utilisant les mots-clés PRIMARY KEY(id).

Lorsque vous insérerez une nouvelle entrée dans la table, MySQL vérifiera que vous insérez bien un id, et que cet id n'existe pas encore dans la table. Si vous ne respectez pas ces deux contraintes, MySQL n'insérera pas l'entrée et vous renverra l'erreur suivante :

ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'

4.2.2. Auto-incrémentation

Il faut donc, pour chaque animal, décider d'une valeur pour id. En utilisant l'auto-incrémentation des champs grâce au mot-clé AUTO_INCREMENT, MySQL mettra automatiquement à jour cet id en l'incrémentant en prenant la dernière valeur insérée.

4.3. Les moteurs de tables

Les moteurs de tables sont une spécificité de MySQL. Ce sont des moteurs de stockage qui gérent différemment les tables. Les deux moteurs les plus connus sont MyISAM et InnoDB.

- MyISAM est le moteur par défaut. Les commandes d'insertion et sélection de données sont particulièrement rapides sur les tables utilisant ce moteur. Cependant, il ne gère pas certaines fonctionnalités importantes comme les clés étrangères, qui permettent de vérifier l'intégrité d'une référence d'une table à une autre table ou les transactions, qui permettent de réaliser

des séries de modifications "en bloc" ou au contraire d'annuler ces modifications.

- InnoDB est plus lent et plus gourmand en ressources que MyISAM. Ce moteur gère les clés étrangères et les transactions. De plus, en cas de crash du serveur, il possède un système de récupération automatique des données.

Pour qu'une table utilise le moteur de notre choix, il suffit d'ajouter ceci à la fin de la commande de création :

ENGINE = moteur;

En remplaçant bien sûr "moteur" par le nom du moteur à utiliser.

4.4. Syntaxe de CREATE TABLE

Nous devons créer une table Animal avec six champs telles que décrites dans le tableau suivant :

Caractéristique	**Nom du champ**	**Type**	**NULL?**	**Divers**
uméro d'identité	id	SMALLINT	Non	Clé primaire + auto-incrément + UNSIGNED
Espèce	espece	VARCHAR(40)	Non	
Sexe	sexe	CHAR(1)	Oui	
Date de naissance	date_naissance	DATETIME	Non	
Commentaires	commentaires	TEXT	Oui	
Nom	nom	VARCHAR(30)	Oui	

La syntaxe globale de la commande est :

```
CREATE TABLE [IF NOT EXISTS] Nom_table (
        champ1 description_champ1,
        [champ2 description_champ2,
        champ3 description_champ3,
        ...,]
        [PRIMARY KEY (champ_clé_primaire)]
)
[ENGINE=moteur];
```

IF NOT EXISTS est facultatif (d'où l'utilisation de crochets [ ]) : si une table de ce nom existe déjà dans la base de données, la requête renverra un warning plutôt qu'une erreur.

Pour définir un champ, il faut donner son nom en premier, puis sa description. La description est constituée au minimum du type du champ. C'est aussi dans la description que l'on précise si la champ peut contenir NULL ou pas.

L'auto-incrémentation se définit également à cet endroit. Notez qu'il est également possible de définir un champ comme étant la clé primaire dans sa description.

Enfin, on peut donner une valeur par défaut au champ. Si lorsque l'on insère une ligne, aucune valeur n'est précisée pour le champ, c'est la valeur par défaut qui sera utilisée. Notez que si un champ est autorisée à contenir NULL et qu'on ne précise pas de valeur par défaut, alors NULL est implicitement considéré comme valeur par défaut.

Exemple : espece VARCHAR(40) NOT NULL DEFAULT 'chien' ;

Si l'on met tout cela ensemble pour créer la table Animal avec le moteur InnoDB, on a donc :

```
CREATE TABLE Animal (
        id                  SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
        espece              VARCHAR(40) NOT NULL,
        sexe                CHAR(1),
        date_naissance      DATETIME NOT NULL,
        nom                 VARCHAR(30),
        commentaires        TEXT,
        PRIMARY KEY (id)
)
ENGINE=InnoDB;
```

Voici deux commandes qui permettent de vérifier que la table Animal a été correctment créée :

- SHOW TABLES; -- liste les tables de la base de données
- DESCRIBE Nom_table; -- liste les champs de la table avec leurs caractéristiques

4.5. Suppression d'une table

La commande pour supprimer une table est la même que celle pour supprimer une base de données. Elle est irréversible.

DROP TABLE Nom_table ;

5. Modification d'une table

5.1. Syntaxe de la requête

Lorsque l'on modifie une table, on peut vouloir lui ajouter, retirer ou modifier quelque chose. Dans les trois cas, c'est la commande ALTER TABLE qui sera utilisée, une variante existant pour chacune des opérations :

- ALTER TABLE nom_table ADD … -- ajouter quelque chose
- ALTER TABLE nom_table DROP … -- retirer quelque chose
- ALTER TABLE nom_table CHANGE ...
- ALTER TABLE nom_table MODIFY … -- modifier un champ

5.2. Ajout d'un champ

On utilise la syntaxe suivante :

```
ALTER TABLE nom_table
ADD [COLUMN] nom_champ description_champ [DEFAULT valeur]
[AFTER nom_champ];
```

- COLUMN est facultatif, donc si à la suite de ADD vous ne précisez pas ce que vous voulez ajouter, MySQL considérera qu'il s'agit d'un champ.

- description_champ contient le type de donnée et éventuellement NULL ou NOT NULL, etc.
- DEFAULT est facultatif
- AFTER est facultatif et indique à quel endroit dans la table il faut insérer la nouvelle champ.

Exemple : ajoutons un champ salle_soins, qui prendra comme valeur par défaut 1, à la table Animal après la champ nom.

```
ALTER TABLE Animal
ADD salle_soins SMALLINT UNSIGNED NOT NULL DEFAULT '1'
AFTER nom ;
```

5.3. Suppression d'un champ

On utilise la syntaxe suivante :

```
ALTER TABLE nom_table
DROP [COLUMN] nom_champ;
```

- COLUMN est facultatif, MySQL considérera qu'il s'agit d'un champ.

Exemple : nous allons supprimer la champ salle_soins.

```
ALTER TABLE Animal
DROP salle_soins;              -- Suppression du champ date_insertion
```

5.4. Modification d'un champ

Utiliser la commande suivante pour changer le nom d'un champ :

```
ALTER TABLE nom_table
CHANGE ancien_nom nouveau_nom description_champ;
```

Exemple : renommer la champ sexe par genre.

```
ALTER TABLE  Animal
CHANGE sexe genre VARCHAR(10) NOT NULL;
```

NB : la description du champ doit être complète, sinon elle sera également modifiée. Si vous ne précisez pas NOT NULL dans la commande précédente, genre pourra contenir NULL, alors que du temps où elle s'appelait sexe, cela lui était interdit.

Les mots-clés CHANGE et MODIFY peuvent être utilisés pour changer le type de donnée du champ, mais aussi changer la valeur par défaut ou ajouter/supprimer une propriété AUTO_INCREMENT. Si vous utilisez CHANGE, vous pouvez renommer la champ en même temps. Si vous ne désirez pas la renommer, il suffit d'indiquer deux fois le même nom.

Exemples de syntaxes possibles :

```
ALTER TABLE nom_table
CHANGE ancien_nom nouveau_nom nouvelle_description;
```

```
ALTER TABLE nom_table
MODIFY nom_champ nouvelle_description;
```

Pour les autres possibilités et combinaisons de la commande ALTER TABLE reportez-vous à la documentation officielle.

6. Insertion de données

6.1. Syntaxe de INSERT

Utiliser la commande suivante pour changer le nom d'un champ :

INSERT INTO table [(nom_champ_1, nom_champ_2, ...]
VALUES ('valeur 1', 'valeur 2', ...)

Il n'est pas obligatoire de donner une valeur pour chaque champ de la ligne.

Exemple :

INSERT INTO Animal
VALUES (NULL, 'chien', 'M', '2010-04-05 13:43:00', 'Rox', 'Mordille beaucoup');

INSERT INTO Animal
VALUES (2, 'chat', NULL, '2010-03-24 02:23:00', 'Roucky', NULL);

La table Animal contient deux champs :

Id	Espèce	Sexe	Date de naissance	Nom	Commentaires
1	chien	M	2010-04-05 13:43:00	Rox	Mordille beaucoup
2	chat	NULL	2010-03-24 02:23:00	Roucky	NULL

Deux choses importantes à retenir ici.

- id est un nombre, on ne met pas de guillemets autour. Par contre, l'espèce, le nom, la date de naissance et le sexe sont donnés sous forme de chaînes de caractères. Les guillemets sont indispensables. Quant à NULL, il s'agit d'un marqueur SQL qui signifie "pas de valeur". Pas de guillemets donc.
- Les valeurs des champs sont données dans le bon ordre (dans l'ordre donné lors de la création de la table).

Pour faire une insertion en précisant les champs, il faut écrire explicitement à quelle(s) champ(s) sont affectées les valeurs. Dans ce cas :

- donner les valeurs dans l'ordre précisé par la requête.
- inutile de donner une valeur à chaque champ.
- il est possible d'insérer plusieurs lignes en une seule requête.

Exemple :

INSERT INTO Animal (espece, sexe, date_naissance, nom)
VALUES ('chien', 'F', '2008-12-06 05:18:00', 'Caroline'),
('chat', 'M', '2008-09-11 15:38:00', 'Bagherra'),
('tortue', NULL, '2010-08-23 05:18:00', NULL);

6.2. Syntaxe alternative de MySQL

MySQL propose une syntaxe alternative à INSERT INTO ... VALUES ... pour insérer des données dans une table.

INSERT INTO table
SET nom_champ_1='valeur 1', nom_champ_2='valeur 2', … ;

NB : cette syntaxe est propre à MySQL.

6.3. Utilisation de fichiers externes

Pour éviter d'écrire toutes les requêtes d'insertion, une solution consiste à écrire les requêtes dans un fichier texte, puis de dire à MySQL d'exécuter les requêtes contenues dans ce fichier.

SOURCE monFichier.sql;

7. Sélection des données

7.1. Syntaxe de SELECT

SELECT permet de sélectionner et afficher des données. Sa syntaxe est la suivante :

SELECT champ1, champ2, ...
FROM nom_table;

Par exemple, si l'on veut sélectionner l'espèce, le nom et le sexe des animaux présents dans la table Animal, on écrira :

SELECT espece, nom, sexe
FROM Animal;

Pour sélectionner toutes les champs, utiliser le caractère * dans la requête :

SELECT *
FROM Animal;

Il est cependant déconseillé d'utiliser SELECT * pour :

- être certain de ce qui est récupéré.
- économiser de ressources.

7.2. La clause WHERE

La clause WHERE ("où" en anglais) permet de restreindre les résultats selon des critères de recherche. On peut par exemple vouloir ne sélectionner que les chiens :

SELECT champ1, champ2, ...
FROM nom_table
WHERE criteres ;

Les critères de recherche peuvent inclure des opérateurs de comparaison ou logique :

Opérateur	**Signification**
=	égal
<	inférieur
<=	inférieur ou égal

Opérateur	**Signification**
AND	ET logique
OR	OU logique
XOR	OU exclusif logique

>	supérieur
>=	supérieur ou égal
<> ou !=	différent
<=>	égal (valable pour NULL aussi)

NOT	NON logique

Exemple :

```
SELECT nom
FROM Animal
WHERE date_naissance < '2008-01-01'     -- Animaux nés avant 2008
AND espece <> 'chat';                   -- Tous les animaux sauf les chats
```

Le cas de NULL

Remarque : il n'est pas possible de tester directement le marqueur NULL. Il faut utiliser l'opérateur de comparaison <=> ou les mots-clés IS NULL.

7.3. Tri des données

Pour trier les données, il suffit d'ajouter ORDER BY.

```
SELECT champ1, champ2, ...
FROM nom_table
WHERE criteres
ORDER BY champ1, champ2, ... [DESC | ASC]
```

Pour déterminer le sens du tri effectué, SQL possède deux mots-clés :

- ASC pour ascendant (par défaut : du plus petit nombre au plus grand, de la date la plus ancienne à la plus récente, et pour les chaînes de caractères et les textes par ordre alphabétique)
- DESC pour descendant. Par défaut, si vous ne précisez rien, c'est un tri ascendant qui est effectué : qui est utilisé.

Exemple :

```
SELECT nom
FROM Animal
WHERE espece='chien'
ORDER BY date_naissance DESC nom ASC;
```

Les plus vieux chiens sont récupérés en premier et par ordre alphabétique.

Cas particulier : les ENUM sont triés selon l'ordre dans lequel les possibilités ont été définies.

7.4. Élimination des doublons

Il peut arriver que MySQL donne plusieurs fois le même résultat car certaines informations sont présentes plusieurs fois dans la table.

Le mot-clé DISTINCT se place juste après SELECT et permet d'éliminer les doublons.

Exemple :

```
SELECT DISTINCT espece
FROM Animal;
```

7.5. Restreindre les résultats

En plus de restreindre une recherche en lui donnant des critères grâce à la clause WHERE, il est possible de restreindre le nombre de lignes récupérées grâce à la clause LIMIT.

LIMIT nombre_de_lignes [OFFSET decalage];

- nombre_de_lignes : le nombre de lignes que l'on veut récupérer.
-
- Decalage : indique à partir de quelle entrée on récupère les résultats (0 par défaut).

Exemple :

```
SELECT *
FROM Animal
ORDER BY id
LIMIT 6 OFFSET 3;
```

Récupére six lignes à partit du quatrième plus petit id.

7.6. Opérateur LIKE

L'opérateur LIKE permet de faire des recherches en utilisant des "jokers", c'est-à-dire des caractères qui représentent n'importe quel caractère.

Deux jokers existent pour LIKE :

- '%' : qui représente n'importe quelle chaîne de caractères, quelle que soit sa longueur (y compris une chaîne de longueur 0) ;
- '_' : qui représente un seul caractère (ou aucun).

Exemples :

- 'b%' cherchera toutes les chaînes de caractères commençant par 'b'
- 'B_' cherchera toutes les chaînes de caractères contenant une ou deux lettres dont la première est 'b'
- '%ch%ne' cherchera toutes les chaînes de caractères contenant 'ch' et finissant par 'ne'
- '_ch_ne' cherchera toutes les chaînes de caractères commençant par 'ch', éventuellement précédées d'une seule lettre, suivies de zéro ou d'un caractère au choix et enfin se terminant par 'ne'

Remarque : pour cherchez une chaîne de caractères contenant '%' ou '_' , il suffit de mettre le caractère d'échappement \ devant le '%' ou le '_'.

LIKE n'est pas sensible à la casse car l'interclassement par défaut du jeu de caractère UTF-8 n'est pas sensible à la casse. Pour faire une recherche sensible à la casse, il fat définir la chaîne de recherche comme une chaîne de type binaire :

```
SELECT *
FROM Animal
```

```
WHERE nom LIKE BINARY '%Lu%';       -- sensible à la casse
```

7.7. Recherche dans un intervalle

Pour faire une recherche sur un intervalle à l'aide uniquement des opérateurs de comparaison, il faut utiliser des opérateurs logiques.

Exemple :

```
SELECT *
FROM Animal
WHERE date_naissance <= '2009-03-23'
AND date_naissance >= '2008-01-05';
```

Cependant, SQL dispose de l'opérateur spécifique BETWEEN pour les intervalles. La requête précédente peut donc s'écrire :

```
SELECT *
FROM Animal
WHERE date_naissance BETWEEN '2008-01-05' AND '2009-03-23';
```

BETWEEN peut s'utiliser aussi avec des nombres (BETWEEN 0 AND 100) ou avec des chaînes de caractères (BETWEEN 'a' AND 'd') auquel cas c'est l'ordre alphabétique qui sera utilisé (toujours insensible à la casse sauf si l'on utilise des chaînes binaires : BETWEEN BINARY 'a' AND BINARY 'd').

Bien évidemment, on peut aussi exclure un intervalle avec NOT BETWEEN.

7.8. Set de critères

L'opérateur IN permet de faire des recherches parmi une liste de valeurs.

Exemple :

```
SELECT *
FROM Animal
WHERE nom IN ('Moka', 'Bilba', 'Tortilla', 'Balou', 'Dana', 'Redbul', 'Gingko');
```

8. Suppression et modification de données

8.1. Sauvegarde d'une base de données

Pour sauvegarder une base de données MySQL dispose donc d'un outil en mode console spécialement dédié : mysqldump.

La syntaxe de la commande est la suivante :

```
mysqldump -u nom_utilisateur -p --opt nom_de_la_base > sauvegarde.sql
```

- mysqldump : client permettant de sauvegarder les bases.
- --opt : option de mysqldump qui lance la commande avec une série de paramètres qui font que la commande s'effectue très rapidement.
- nom_de_la_base : le nom de la base qu'on veut sauvegarder.

- > sauvegarde.sql : le signe > indique que l'on va donner la destination de ce qui va être généré par la commande : sauvegarde.sql. Il s'agit du nom du fichier texte qui contiendra la sauvegarde de la base.

Si la base a été efacée, il vous faut d'abord la recréer (CREATE DATABASE nom_base), puis exécuter la commande suivante en mode console :

mysql nom_base < sauvegarde.sql

Ou, directement à partir de MySQL :

USE nom_base;
SOURCE fichier_de_sauvegarde.sql;

8.2. Suppression

La commande utilisée pour supprimer des données est DELETE. Cette opération est irréversible.

DELETE FROM nom_table
[WHERE criteres];

Utiliser la clause WHERE pour préciser quelles lignes doivent être supprimées.

8.3. Modification

La modification des données se fait grâce à la commande UPDATE :

UPDATE nom_table
SET nom_champ_1='valeur 1', nom_champ_2='valeur 2, ...
[WHERE criteres];

Exemple :

UPDATE Animal
SET sexe='F', nom='Pataude'
WHERE id=21;

NB : si la clause WHERE est omise, la modification se fera sur toutes les lignes de la table.

9. Index

Un index est une structure de données qui reprend la liste ordonnée des valeurs auxquelles il se rapporte.

Lorsque vous créez un index sur une table, MySQL stocke cet index sous forme d'une structure particulière, contenant les valeurs des champs impliquées dans l'index. Cette structure stocke les valeurs triées et permet d'accéder à chacune de manière efficace et rapide afin d'accélérer les requêtes qui utilisent des champs indexées comme critères de recherche.

Id	Id	Espèce	Sexe	Date de naissance	Nom	Commentaires	Date de naissance
1	2	chat	NULL	2010-03-24 02:23:00	Roucky	NULL	2008-09-11 15:38:00
2	1	chien	male	2010-04-05 13:43:00	Rox	Mordille beaucoup	2008-12-06 05:18:00
3	3	chat	femelle	2010-09-13 15:02:00	Schtroumpfette	NULL	2009-06-13 08:17:00
4	6	tortue	femelle	2009-06-13 08:17:00	Bobosse	Carapace bizarre	2009-08-03 05:12:00
5	9	tortue	NULL	2010-08-23 05:18:00	NULL	NULL	2010-03-24 02:23:00
6	4	tortue	femelle	2009-08-03 05:12:00	NULL	NULL	2010-04-05 13:43:00
7	7	chien	femelle	2008-12-06 05:18:00	Caroline	NULL	2010-08-23 05:18:00
8	8	chat	male	2008-09-11 15:38:00	Bagherra	NULL	2010-09-13 15:02:00
9	5	chat	NULL	2010-10-03 16:44:00	Choupi	Né sans oreille gauche	2010-10-03 16:44:00

Cependant, les index ont deux inconvénients :

- Ils prennent de la place en mémoire
- Ils ralentissent les requêtes d'insertion, modification et suppression.
- Ils ajoutent une contrainte.

Par conséquent, il faut donc ajouter un index uniquement sur un champ sur laquelle il y a beaucoup de recherches.

9.1. Index unique

Les index permettent aussi d'assurer l'intégrité des données de la base. Ainsi, avoir un index UNIQUE sur un champ (ou plusieurs) permet de s'assurer que jamais vous n'insérerez deux fois la même valeur (ou combinaison de valeurs) dans la table.

Exemple : dans la table Animal, il est décidé de ne pas nommer de la même manière deux animaux de la même espèce.

9.2. Création des index

Les index sont représentés par le mot-clé INDEX ou KEY et peuvent être créés de deux manières :

- soit directement lors de la création de la table ;
- soit en les ajoutant par la suite.

```
CREATE TABLE nom_table (
        champ1 INT KEY,                    -- Crée un index simple sur champ1
        champ2 VARCHAR(40) UNIQUE,  -- Crée un index unique sur champ2
);
```

- seul le mot KEY peut être utilisé pour définir un index simple. Ailleurs, vous pourrez utiliser KEY ou INDEX.
- pour définir un index UNIQUE de cette manière, on n'utilise que le mot-clé UNIQUE, sans le faire précéder de INDEX ou de KEY.

L'autre possibilité est d'ajouter les index à la suite des champs, en séparant chaque élément par une virgule :

```
CREATE TABLE nom_table (
        champ1 description_champ1,
        [champ2 description_champ2,
        champ3 description_champ3,
        ...,]
        [PRIMARY KEY (champ_clé_primaire)],
        [INDEX [nom_index] (champ1_index [, champ2_index, ...]]
)
[ENGINE=moteur];
```

Exemple : on veur créer la table Animal avec un index sur la date de naissance, et un autre sur les 10 premières lettres du nom et en ajoutant un index UNIQUE sur (nom, espece).

```
CREATE TABLE Animal (
        id                  SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
        espece              VARCHAR(40) NOT NULL,
        sexe                CHAR(1),
        date_naissance      DATETIME NOT NULL,
        nom                 VARCHAR(30),
        commentaires        TEXT,
        PRIMARY KEY (id),
        INDEX idx_date_naissance (date_naissance),      -- index sur la date de naissance
        INDEX idx_nom (nom(10))                         -- index sur le nom
        UNIQUE INDEX udx_nom_espece (nom, espece) -- Index sur le nom et l'espece
)
ENGINE=InnoDB;
```

9.3. Ajout des index après création de la table

Il existe deux commandes permettant de créer des index sur une table existante : ALTER TABLE, que vous connaissez déjà un peu, et CREATE INDEX.

```
ALTER TABLE nom_table
ADD INDEX [nom_index] (champ_index [, champ2_index ...]); --Ajout d'un index simple
```

```
ALTER TABLE nom_table
ADD UNIQUE [nom_index] (champ_index [, champ2_index ...]); --Ajout d'un index UNIQUE
```

La syntaxe de CREATE INDEX est très simple :

```
CREATE INDEX nom_index
ON nom_table (champ_index [, champ2_index ...]);  -- Crée un index simple
```

```
CREATE UNIQUE INDEX nom_index
ON nom_table (champ_index [, champ2_index ...]);  -- Crée un index UNIQUE
```

9.4. Suppression d'un index

La syntaxe est la suivante :

```
ALTER TABLE nom_table
DROP INDEX nom_index;
```

NB : il n'existe pas de commande permettant de modifier un index. Le cas échéant, il faudra supprimer, puis recréer l'index avec vos modifications.

10. Clés primaires et étrangères

10.1. Les clés primaires

La clé primaire d'une table est une contrainte d'unicité, composée d'une ou plusieurs champs, et qui permet d'identifier de manière unique chaque entrée de la table. Ce qui implique :

- contrainte d'unicité : index UNIQUE.
- les clés peuvent être composites.
- une clé primaire ne peut pas être NULL.

Le choix d'une clé primaire est une étape importante dans la conception d'une table.On est bien souvent obligé d'ajouter un champ auto-incrémentée pour jouer le rôle de la clé primaire car :

- les recherches sont beaucoup plus rapides sur des nombres que sur des textes.
- l'auto-incrémentation donne un identifiant unique.

10.2. Création d'une clé primaire

La création des clés primaires est semblable à la création d'index simples. La clé primaire peut être créée en même temps que la table, ou par la suite.

```
CREATE TABLE [IF NOT EXISTS] Nom_table (
        champ1 description_champ1 AUTO_INCREMENT PRIMARY KEY [,
        champ2 description_champ2,
        champ3 description_champ3,
        ...,]
)
[ENGINE=moteur];
```

On peut toujours utiliser ALTER TABLE. Par contre, CREATE INDEX n'est pas utilisable pour les clés primaires.

```
ALTER TABLE nom_table
ADD PRIMARY KEY (champ_pk1 [, champ_pk2, ...]);
```

10.3. Suppression de la clé primaire

La syntaxe est la suivante :

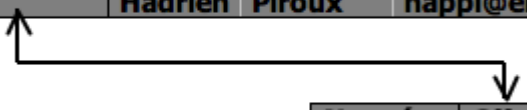
```
ALTER TABLE nom_table
DROP PRIMARY KEY ;
```

10.4. Clés étrangères

Les clés étrangères ont pour fonction principale la vérification de l'intégrité de votre base et de s'assurer que vous n'insérez pas de bêtises…

Dans l'exemple ci-dessous, la table Commande a un champ qui contient une référence au client. À l'insertion d'une entrée dans la table Commande,la clé étrangère va vérifier que le numéro de client indiqué correspond bien à quelque chose dans la table Client.

Numéro	Nom	Prénom	Email
1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com
4	Hadrien	Piroux	happi@email.com



Numéro	Client	Produit	Quantité
1	3	Tube de colle	3
2	2	Rame de papier A4	6
3	2	Ciseaux	2

- Comme pour les index et les clés primaires, il est possible de créer des clés étrangères composites.
- Lorsque vous créez une clé étrangère sur un champ, un index est automatiquement ajouté sur celle-ci.
- La champ qui sert de référence doit déjà posséder un index (où être clé primaire).
- La champ sur laquelle la clé est créée doit être exactement du même type que la champ qu'elle référence. Cela implique qu'en cas de clé composite, il faut le même nombre de champs dans la clé et la référence.
- Tous les moteurs de table ne permettent pas l'utilisation des clés étrangères. Par exemple, MyISAM ne le permet pas, contrairement à InnoDB.

Pour créer une clé étrangère il faut :

- utiliser FOREIGN KEY sur la champs sur laquelle on crée la clé.
- utiliser REFERENCES sur la ou les champs qui vaservir de référence.

```
CREATE TABLE [IF NOT EXISTS] Nom_table (
        champ1 description_champ1,
        [champ2 description_champ2,
        champ3 description_champ3,
        ...,]
        FOREIGN KEY (champ(s)_clé_étrangère)
        REFERENCES table_référence (champ(s)_référence)]
)
[ENGINE=moteur];
```

Exemple :

```
CREATE TABLE Commande (
        numero      INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
        client      INT UNSIGNED NOT NULL,
        produit     VARCHAR(40),
        quantite    SMALLINT DEFAULT 1,
```

```
        FOREIGN KEY (client)            -- champ sur laquelle on crée la clé
        REFERENCES Client(numero)       -- champ de référence
)
ENGINE=InnoDB;              -- MyISAM interdit
```

11. Jointures

Le principe des jointures est de joindre plusieurs tables. Pour ce faire, on utilise les informations communes à ces tables.

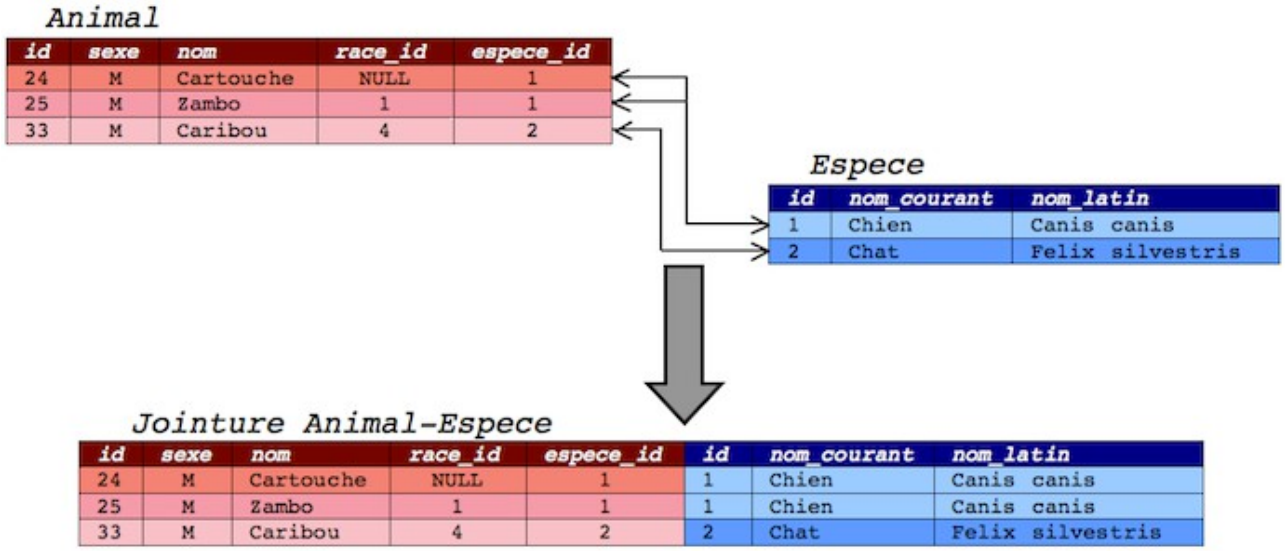
Dans l'exemple de la table Animal, c'est une très mauvaise idée d'y avoir stockées les informations sur les espèces. En effet, il suffit de saisir le nom de l'espèce avec une faute d'orthographe pour introduire une nouvelle espèce et avoir un résultat erroné sur les requêtes de recherche. De même, si on veut introduire des informations complémentaires sur les espèces, il faudra alors renseigner toutes les champs !

De façon générale, il ne faut jamais dupliquer l'information dans une base de données.

Nous devons alors créer une nouvelle table Espece, dans laquelle se trouvera le nom de l'espèce (chien, chat, …) et, par exemple, une information complémentaire comme son nom en latin. À la place du nom de l'espèce dans la table Animal, on renseignera un champ espece_id qui pointera sur l'entrée de la table Espece où sont enregistrés tous les renseignements ad hoc.

De ce fait, pour afficher l'espèce de l'animal, il faudra deux requêtes.

1. Étape 1 : on trouve l'id de l'espèce de l'animal grâce à la table Animal.
2. Étape 2 : on trouve le nom de l'espèce grâce à son id dans la table Espece.



Pour obtenir cette information avec une seule requête, il va falloir utiliser une jointure entre les deux tables : en l'occurence, l'information commune sera l'id de l'espèce comme indiqué sur la schéma ci-dessus.

La syntaxe de la clause SELECT deviendra alors :

```
SELECT table1.champ_table1 , table2.champ_table2, ...
FROM table1, table2, ...
WHERE table1.champ = table2.champ;
```

```
[ORDER BY …]                        -- clauses habituelles
[LIMIT …]
```

Exemple :

```
SELECT Espece.nom_courant, Animal.nom
FROM Espece,  Animal
WHERE Espece.id = Animal.espece_id
ORDER BY Animal.nom;
```

Une autre solution consiste à utiliser la clause JOIN :

```
SELECT *                            -- sélection des champs
FROM table1
[INNER] JOIN table2                 -- INNER explicite le fait qu'il s'agit d'une jointure interne
ON champ_table1 = champ_table2      -- sur quelles champs se fait la jointure
[WHERE ...]
[ORDER BY …]                        -- clauses habituelles
[LIMIT …]
```

Exemple :

```
SELECT      Espece.id,              -- ici, pas le choix, il faut préciser
            Espece.nom_courant ,    -- ici, on pourrait mettre juste nom_courant
            Animal.nom              -- idem, la précision n'est pas obligatoire
FROM Espece
INNER JOIN Animal
ON Espece.id = Animal.espece_id
WHERE Animal.id = '4';
```

- SELECT Espece.id : sélectionne la champ id de la table Espece.
- FROM Espece : on travaille sur la table Espece.
- INNER JOIN Animal : jointure interne à la table Animal.
- ON Espece.id = Animal.espece_id : la jointure se fait sur les champs id de la table Espece et espece_id de la table Animal, qui doivent correspondre.
- WHERE Animal.id = '4' : dans la table résultant de la jointure, sélectionne de l'entrée qui a la valeur "4" dans la champ id venant de la table Animal.

La clause ON sert à préciser la condition de la jointure. C'est-à-dire sur quel(s) critère(s) les deux tables doivent être jointes. Dans la plupart des cas, il s'agira d'une condition d'égalité simple mais il est cependant tout à fait possible d'avoir plusieurs conditions à remplir pour lier les deux tables. On utilise alors les opérateurs logiques habituels

```
SELECT *
FROM table1
INNER JOIN table2
ON champ1_table1 = champ1_table2
AND champ2_table1 = champ2_table2
[AND ...];
```

12. Les fonctions SQL

Les fonctions SQL peuvent être classées en deux catégories :

- les fonctions scalaires : elles agissent sur chaque entrée. Par exemple, vous pouvez transformer en majuscules la valeur de chacune des entrées d'un champ ;
- les fonctions d'agrégat : lorsque vous utilisez ce type de fonctions, des calculs sont faits sur l'ensemble de la table pour retourner une valeur.

12.1. Les fonctions scalaires

Il existe beaucoup de fonctions SQL de ce type à découvrir dans la documentation de MySQL.

Voici quelques fonctions scalaires utiles :

Fonction	**Description**
UPPER	Cette fonction convertit le texte d'un champ en majuscules.
LOWER	Cette fonction a l'effet inverse : le contenu sera entièrement écrit en minuscules.
LENGTH	Cette fonction compte le nombre de caractères.
ROUND	Cette fonction arrondit un nombre décimal. La fonction ROUND() s'utilise sur des champs comportant des valeurs décimales. Celle-ci prend deux paramètres : le nom du champ à arrondir et le nombre de chiffres après la virgule que l'on souhaite obtenir.

Exemple d'utilisation d'une fonction scalaire SQL : supposons que nous souhaitions obtenir les noms de tous les anmaux en majuscules.

SELECT UPPER(nom) FROM Animal

NB : le contenu de la table reste inchangé. La fonction UPPER modifie seulement la valeur envoyée à PHP.

Cela crée en fait un « champ virtuel », appelé alias, qui n'existe que le temps de la requête. Il est conseillé de donner un nom à cet alias qui représente les noms en majuscules en utilisant pour cela le mot-clé AS :

SELECT UPPER(nom) AS nom_maj FROM Animal

On récupère ainsi les noms des animaux en majuscules via un l'alias nom_maj.

```
<?php
$reponse = $bdd->query("SELECT UPPER(nom) AS nom_maj FROM Animal");
while ( $donnees = $reponse->fetch() )
{
        echo $donnees['nom_maj'] ."<br/>";
}
$reponse->closeCursor();
?>
```

12.2. Les fonctions d'agrégat

Il existe également fr nombreuses fonctions d'agrégat à découvrir dans la documentation.

Ces fonctions diffèrent assez des précédentes car plutôt que de modifier des valeurs une à une, elles

font des opérations sur plusieurs entrées pour retourner une seule valeur.

Voici quelques fonctions d'agrégat représentatives :

Fonction	**Description**
AVG	calcule la moyenne d'un champ contenant des nombres.
SUM	permet d'additionner toutes les valeurs d'un champ.
MAX	Cette fonction analyse un champ et retourne la valeur maximale trouvée.
MIN	retourne la valeur minimale.
COUNT	La fonction COUNT permet de compter le nombre d'entrées. NB : utiliser le joker * pour toutes les entrées.
GROUP BY	grouper des données. Il faut utiliser GROUP BY en même temps qu'une fonction d'agrégat, sinon il ne sert à rien. Ex : SELECT COUNT(sexe) AS nb_sexe, espece_id FROM Animal WHERE sexe = 'M' GROUP BY espece_id
HAVING	filtrer les données regroupées. HAVING ne doit s'utiliser que sur le résultat d'une fonction d'agrégat. Ex : SELECT COUNT(sexe) AS nb_sexe, espece_id FROM Animal WHERE sexe = 'M' GROUP BY espece_id HAVING nb_sexe < 10

.