Physique et Ingénierie

Les Bases de Données

Patrick TRAU

UNIVERSITÉ DE STRASBOURG

1) Stockage des données

1.1) Nombres

Vous le savez déjà. Dans les ordinateurs, les nombres sont stockés en binaire. Dans la pratique : on note 1 s'il y a du courant, 0 s'il n'y en a pas. Chaque « chiffre en binaire » (BInary digiT) est appelé un bit. On les regroupe par 8 pour former un octet (byte). Dans un octet, il y a 256 combinaisons de 0 et de 1 possibles. Si cela ne suffit pas, on utilise plusieurs octets pour former un « mot » (word). Rien ne permet de distinguer la fin d'un mot : à la fin d'un octet, on ne sait pas si le mot est fini ou s'il faut continuer avec le suivant. Il faut donc OBLIGATOIREMENT que les différentes tailles de mots soient définies à l'avance par le programmeur.

Pour des entiers, positifs, un octet permet des nombres entre 0 et 255, un mot de 16 bits entre 0 et 65535 (=64k avec 1k=1024), 32 bits entre 0 et 4T... Si l'on veut gérer les nombres négatifs, ici encore, pas moyen de mettre un signe spécial (car on n'a que deux possibilités, courant ou pas), on décide donc de réserver un bit spécifique pour cela (le plus à gauche, pour un mot dans l'octet de poids le plus fort). Attention, les bits suivants ne sont pas les mêmes que ceux de sa valeur absolue (complément à 2, pour ceux qui savent ce que c'est). Pour la virgule, c'est encore plus compliqué, on utilise généralement une méthode dite « virgule flottante », utilisant 4 octets (simple précision) ou 8 (double précision). Je n'en dirai pas plus.

1.2) Texte

Chaque caractère est représenté par une combinaison de 0 et de 1 (et rien ne le distingue d'un nombre). Le code ASCII (American Standard Code for Information Interchange) n'est standard que pour 127 caractères, limités à ceux utilisés par les anglosaxons (pas d'accents). En binaire ils commencent par un 0 suivi de 7 bits. Sur un octet, il reste 128 combinaisons possibles (commençant par un 1 en binaire), utilisées de manière différente suivant les cas (ISO_latin_15 pour l'Europe de l'Ouest comme nous, ISO_Cyrillic, ISO_Arabic, ISO_Greek... et malheureusement Windows-1254 et même MS-DOS). Un caractère « étendu » est une suite de 8 bits commençant par un 1, et sera affiché différemment suivant la configuration de la machine. Nos accents donneront donc des lettres cyrilliques chez un russe. Malheureusement, MS-DOS n'utilisait pas la norme ISO (Mr Gates est au dessus des normes), d'où des caractères qui peuvaient s'afficher bizarrement. Et sous Windows, il en a inventé un nouveau, différent, mais lui aussi différent de l'ISO !

On a donc décidé de faire un code universel. Plutôt que de prévoir obligatoirement plus d'un octet (et donc augmenter énormément la place utilisée), on a décidé de gérer une taille variable : on garde les 127 codes ascii (commençant par un 0). Si le code commence par un 1, c'est qu'on utilisera plus d'un octet (je ne détaille pas plus). Si vous respectez l'ascii, vos textes seront vus de la même matière par tout le monde. Sinon, il faudra qu'ils utilisent la même norme que vous. C'est donc important pour des informations transitant sur divers ordinateurs dans le monde (adresse mail, nom de fichier,...). C'est également important pour ce que vous voulez garder longtemps (photos, vidéos...)

Pour gérer plusieurs caractères, il suffit de les mettre les uns après les autres : le « fichier texte ». Il a fallu créer un code ascii pour certains signes particuliers (dont fin de ligne et fin de fichier). Pour prévoir une mise en page (choix de police, taille...) il y a eu autant de codages que de logiciels, et quand l'un s'est imposé (.doc), Microsoft en invente un autre (docx) ! Il existe de très nombreux moyens de représenter un texte avec sa mise en forme. Certains connaissent Latex, plus près de nous HTML, postscript, pdf...

1.3) Images

Une image peut-être « vectorielle » : elle est composée d'éléments (segment de droite défini par deux points, arc de cercle, rectangle (rempli ou non...), associé pour chacun à quelques attributs (couleur du trait, remplissage, épaisseur...). Mais aucun format n'a émergé , chaque logiciel utilise le sien. On le réserve donc à des logiciels spécifiques, 3D ou CAO. Les formats plus standards gèrent des ensembles de points.

Une image « monochrome » comporte une suite de bits indiquant, pour chaque point de l'image (dans l'ordre du balayage de l'écran cathodique : ligne par ligne en commençant par celle du haut), s'il est allumé ou éteint. Chaque point de l'image (PICTure ELement) est appelé pixel. Evidemment, ce n'est utilisable que pour du texte ou des schémas. Pour une photo (dite à tort noir et blanc), chaque pixel est représenté par un nombre : le « niveau de gris ». Si chaque pixel est sur 8 bits, on a donc 256 niveaux de gris. Si l'on veut plus de niveaux, il faudra aussi plus de place.

Si l'on veut passer en couleur, on peut prévoir une « palette de couleurs » : chaque couleur est représentée par un nombre, chaque pixel est représenté par son numéro de couleur. Le format BMP (bitmap) prévoyait 16 couleurs uniquement ! On est désormais passé à 256 couleurs, dont certaines personnalisables.

Mais on peut aussi représenter chaque pixel par l'intensité des trois couleurs de base : le rouge, le vert et le bleu (RGB Red Green Blue) dont on disposait sur un écran cathodique. On peut soit créer trois images de type « niveau de gris » : c'est le codage par « plans de couleurs », soit mettre les trois composantes pour chaque pixel : c'est le format « true color ». Si vous choisissez 256 niveaux par couleur de base, on prend 3 octets par pixel, ce qui fera TRES gros pour une photo d'un appareil actuel.

Les images prenant beaucoup de place, on choisit de les comprimer. Par exemple, à partir d' un BMP, on peut obtenir un format GIF. Au lieu de noter la suite des numéros de couleurs, on note la couleur et le nombre d'octets suivants qui sont de la même couleur. Donc si on a des grandes plages de couleur uniforme, on prend peu de place (d'autant qu'on est limité en nombre de couleurs possibles). C'est donc le format idéal pour un schéma. Et, comme pour un ZIP, le résultat est exactement le dessin initial.

Pour la photo, il n'y a pas deux points avec exactement la même couleur (dégradés, ombres...). On utilise un format « true color ». Mais quand on stocke un pixel, on vérifie si les points alentour sont presque identiques, et dans ce cas on ne sotcke que la couleur médiane. C'est le format JPEG : on définit un taux d'approximation maximale de la couleur. Plus on est précis, plus on prend de place. Mais attention, une fois sauvegardé avec une précision donnée, il n'y a plus moyen de retrouver les nuances initiales.

Un film est une suite d'images jpeg (mpeg). On peut encore comprimer plus : si un pixel ne change pas dans une suite d'images, inutile de le stocker à chaque fois. C'est le principe du Divx, on comprime bien s'il y a peu de mouvement. Si l'on a beaucoup de mouvement (et qu'on a quand même beaucoup compressé), le mouvement deviendra saccadé.

1.4) Le son

On stocke une suite de fréquences, avec une fréquence d'échantillonnage fixe : le format WAV. On peut ensuite comprimer, en général avec pertes (MP3 par ex)

2) Organisation des données

2.1) 1D

La première méthode est de stocker les données séquentiellement (l'une après l'autre), c'est un « fichier ». Historiquement, on utilisait pour cela des bandes magnétiques, c'était obligatoirement séquentiel (dimension 1). Mais il y a deux possibilités : les enregistrements peuvent être de longueur variable, il faut donc un séparateur (par exemple « entrée »). L'inconvénient est que pour trouver la 42è ligne, je dois chercher les 41 séparateurs précédents, en parcourant le fichier depuis le début. L'autre possibilité est

d'avoir des enregistrements de taille fixe prédéfinie (complété par des espaces par exemple). Dans ce cas je sais trouver directement le 42è (à 41*taille du début) : on les appelle les fichiers à accès direct. C'est plus rapide mais on risque de perdre beaucoup d'espace.

2.2) 2D

La seconde dimension est obtenue en utilisant un tableau : lignes, colonnes. Chaque ligne (on dit aussi « enregistrement ») comporte les données d'une « entité ». Chaque colonne (on dit aussi « champ ») contient une propriété commune des entités.

Les lignes doivent toutes être distinctes : pas deux lignes identiques (si l'on voulait en entrer plusieurs identiques, juste pour les dénombrer, c'est qu'on a oublié le champ « nombre »). Si ce n'est pas le cas, on rajoute par exemple une colonne pour numéroter les entités. La colonne (ou l'ensemble de colonnes, comme « nom » + « prénom ») qui définissent cette unicité est appelée « clé ». Pour chaque entité, la valeur de la clé est appelée son « identifiant ».

Il ne faut pas enregistrer une donnée que l'on peut calculer à partir des autres données stockées : une modification d'une partie des données donnerait une table incohérente ! Il faut alors disposer d'un outil de gestion des données (le tableur) qui sache faire automatiquement (et rapidement) ces calculs : les « formules » (voire même des graphiques, et dans l'avenir peut-être de la synthèse vocale, du film 3D...)

2.3) 3D

Le passage à la troisième dimension se fait en utilisant plusieurs tables. Chaque table gère des entités de type différent, et comporte donc des champs différents (sinon, on se serait limité à une seule table). Mais dans une table, on peut faire « référence » à une entité d'une autre table. On dira que les deux tables sont « liées par une relation ». Pour référencer une entité, on utilisera bien évidement son identifiant. La colonne de ces références est appelée une « clé étrangère ». Dans la table « référée », s'il peut y avoir plusieurs clés, on en choisira une qui sera celle qu'on utilisera comme référence, on l'appelle la « clé primaire ».

L'outil qui permet de gérer de multiples tables est appelé « système de gestion de bases de données » : SGBD. Il saura bien sûr au minimum faire ce que fait un tableur. S'il gère les relations entre tables (clés primaires, clés étrangères), c'est un SGBDR (SGBD relationnel). Un SGBD doit donc gérer les tables multiples et en vérifier l'intégrité (unicité des identifiants, existence d'une clé primaire pour chaque clé étrangère...). Il permet d'ajouter, supprimer, modifier les données. Il permet d'extraire des données, suite à une demande appelée « requête ». De plus, s'il est multi-utilisateur (ce qui est généralement le cas), il gère les identités et mots de passe, et permet de définir qui peut regarder les données, qui peut les modifier, qui peut modifier la structure de la base (ajouter des champs, des tables...). Un SGBDR doit aussi garantir l'intégrité des relations : si l'on supprime une ligne d'une table, que faire des autres tables référant cette ligne ?

Il existe deux grands types de SGBD :

a) Le SGBD local : il tourne sur l'ordinateur qui contient les données. A un instant donné, seul un utilisateur unique a accès à la base. Chez Microsoft, c'est ACCESS. Je préfère bien sûr « libre office base » ou « OoBase ». Ces logiciels peuvent être utilisés en réseau : les données sont sur un serveur de fichiers, le logiciel installé (et payé) sur chaque poste utilisateur. Si sur un poste l'on fait une requête, le logiciel va devoir lire toutes les données, qui devront donc lui être transmises via le réseau. Si l'on refait une requête un peu plus tard, il faut à nouveau transférer toutes les données (au cas où un autre poste aurait fait une modification). Certains artifices ont été mis en œuvre dans ces logiciels pour limiter ces problèmes, mais ce n'est efficace que pour un système avec peu de postes, et surtout peu de modifications.

b) L'architecture client-serveur (SGBD distribué) : un serveur contient le logiciel et accède aux données (en général sur son propre disque dur). Le logiciel serveur SGBD attend qu'on lui transmette une requête, il la traite, et retourne la réponse à l'envoyeur (et uniquement la réponse). Pour cela, il faut, pour chaque utilisateur, un logiciel « client » : il permet de préparer la requête (avec une interface avec le clavier, et

désormais la souris voire le micro), la transmettre, puis, à la réception de la réponse, la mettre en forme (texte voire graphiques). Aujourd'hui, les clients sont le plus souvent des navigateurs internet. A côté du serveur SGBD, un serveur web sert d'interface. Parmi les SGBD les plus connus : Oracle, MySQL (que j'utilise), MS-SQL (même Microsoft a compris qu'Access n'était pas une solution suffisante).

Autrefois, la forme de la requête (et de la réponse) était spécifique à la marque du SGBD. Désormais, tous utilisent une langage standardisé : SQL (structured query language), bien que chaque SGBD ait gardé quelques spécificités.

c) j'avais dit qu'il existe deux solutions. Une troisième pointe désormais son nez : l'organisation en « nuage » (cloud). Les données sont éclatées en de multiples endroits, les requêtes arrivant sur un poste sont traitées, en répondant aux parties dont on la réponse, et en lançant d'autres requêtes pour les autres. Tout se diffuse donc dans le réseau, reste à savoir comment on peut contrôler son expansion exponentielle. N'ayant pas (encore) la réponse, nous exclurons cette méthode dans la suite de ce document.

3) Structure de la base

Supposons que l'on veuille gérer les notes de nos étudiants. Bien sûr, cette base de données existe, et est beaucoup plus compliquée. Nous utilisons 3 tables : les étudiants (identifiés par leur numéro d'étudiant), les matières (identifiées par un code sur 6 caractères), et enfin les notes (un étudiant, une matière => une note). J'ai crée les tables (sous Libre Office OoBase) et y ai mis quelques données :

	Nom de champ	Type de champ	Description
🔑	numetu	Texte (fixe) [CHAR]	12 caractères, clé primaire
	prenom	Texte [VARCHAR]	50 char max
	nom	Texte [VARCHAR]	50 char max
	naissance	Date [DATE]	
	mail	Texte [VARCHAR]	50 char max

numetu	prenom	nom	naissance	mail
123456789	jean	dupont	01/01/80	jdupont@unistra.fr
123456790	jules	martin		jmartin@gmail.com
123456791	marc	decafe	01/01/80	marco@hotmail.fr

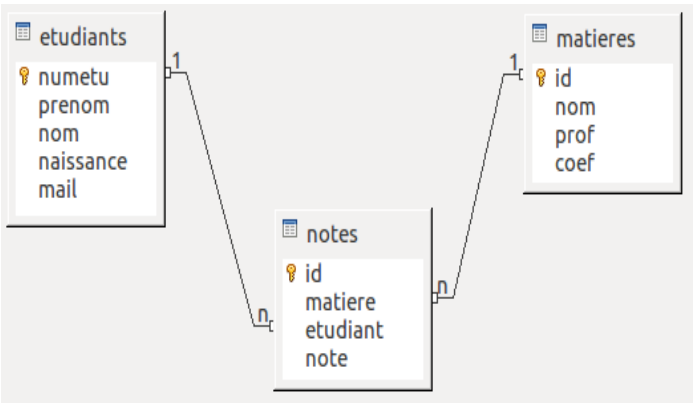
	Nom de cha...	Type de champ	Description
🔑	id	Texte (fixe) [CHAR]	6 caractères, clé primaire
	nom	Texte [VARCHAR]	nom de la matière, 30 char max
	prof	Texte [VARCHAR]	nom du prof, 30 char max
	coef	Float [FLOAT]	

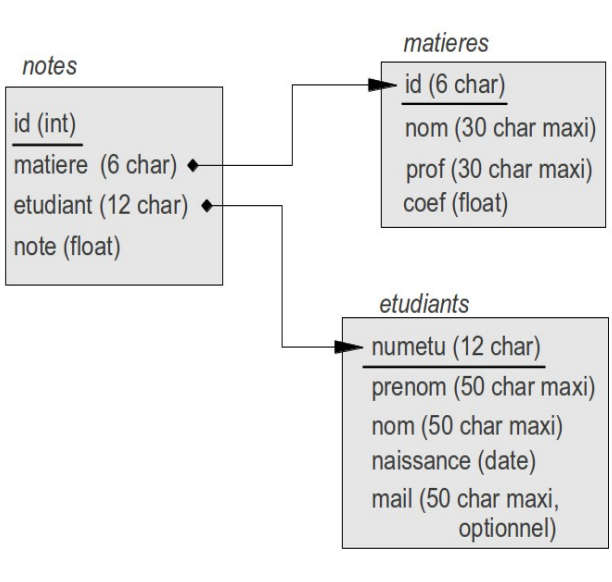
id	nom	prof	coef
math	maths	Einstein	1,5
pbd	prog é ba	P. Trau	1
sci	sciences	G Trouve	2

	Nom de cha...	Type de champ	Description
🔑	id	Integer [INTEGER]	clé primaire (pas nécessaire pour l'instant)
	matiere	Texte (fixe) [CHAR]	clé étrangère (code matière sur 6 char)
	etudiant	Texte (fixe) [CHAR]	clé étrangère (num étu sur 12 char)
	note	Float [FLOAT]	

id	matiere	etudiant	note
1	math	123456789	10
2	pbd	123456789	12
3	math	123456790	8
4	pbd	123456790	12
5	math	123456791	15
6	pbd	123456791	18
7	sci	123456789	11
8	sci	123456791	8

Pour représenter la structure, on réalise un « schéma structurel » : Chaque table est représentée dans un rectangle, le titre est le nom de la table. Puis on indique le nom de chaque champ (l'un en dessous de l'autre). Normalement, on y rajoute (entre parenthèses) les divers attributs du champ : type (integer, float, char, varchar, date...), taille, clé primaire (ou bien on la souligne), obligatoire ou optionnelle (un champ obligatoire signifie qu'on doit lui donner une valeur, les optionnels peuvent ne pas être renseignés). Puis on note les relations : une flèche **part** d'une référence (clé étrangère) et **rejoint** la clé primaire correspondante. J'ai représenté ici le schéma donné par OoBase et le mien :





Dans ma représentation, la flèche part d'une clé étrangère, et pointe vers une clé primaire. Par exemple, de la table « notes » vers la table « matieres » : dans la table notes (côté du point), on peut retrouver plusieurs fois la même matière (il y a en général plus d'un étudiant en cours). Mais dans la table « matieres » (côté de la flèche), on ne peut trouver qu'un fois un id donné. On appelle ce type de relation « 1,N », et c'est ainsi que l'a représenté OoBase. On peut également (mais plus rarement) utiliser une relation « 1,1 » (que je représente avec une flèche à chaque extrémité) : on dit alors que la clé étrangère est « unique ». Exemple d'utilisation : si pour les étudiants étrangers, il fallait un certain nombre d'informations supplémentaires, qu'il est inutile de prévoir pour les autres, on pourrait les mettre dans une autre table, avec une relation 1,1 (la référence serait alors optionnelle)

Attention, un certain nombre de mes étudiants ont du mal à ne pas spécifier des entités dans les schémas de structure (dans notre exemple, certains y incluraient les mots math, pbd, sci...). Dans un schéma structurel, on ne parle pas des données contenues dans les tables, mais uniquement des noms des champs !

4) Le langage SQL

Le SQL (Structured Query Language) est un langage normalisé (depuis les années 80-90). Les éditeurs des gros SGBD ont pris du temps à accepter ce langage, leur propre langage (propriétaire) forçait leurs clients à se lier à leur produit. Mais désormais seuls ceux qui l'ont accepté ont survécu.

SQL permet de faire une requête (query) à un SGBD. Il y a trois manières de l'utiliser :

- directement dans le logiciel (par exemple dans OoBase)
- pour un SGBD distribué, utilisation d'un client qui permet de taper directement du SQL au clavier, et l'envoie au serveur (j'utilise généralement mysql-client dans mes démonstrations)
- utiliser un client évolué, qui a une interface graphique, et qui forme une requête SQL en fonction de ce que vous cliquez, l'envoie au serveur, puis, quand il reçoit la réponse, la traite pour vous afficher un résultat graphique. Exemples : phpMyAdmin (que j'utilise souvent en TP) qui est prévu pour les spécialistes des bases de données ; la page web de votre livreur de pizzas, qui vous calcule automatiquement le prix en fonction des ingrédients que vous choisissez, destinée au non spécialiste (bien que les informaticiens aiment aussi commander des pizzas, surtout quand un programme leur résiste tard dans la soirée).

SQL est indépendant de la casse, mais je mettrai les mots clef en majuscules pour les faire ressortir. Les espaces sont des séparateurs, donc interdits dans les noms de tables ou de colonnes (mais autorisé dans les données, à condition d'entourer le texte par des guillemets (simple ou double quote). Une commande est envoyée à MySQL par son « ; » final (vous pouvez donc aller à la ligne dans une commande, ainsi qu'indenter).

4.1) Accès au SGBD en SQL

Dans le cas d'un SGBD local comme OoBase : cliquez sur « requêtes » puis « créer une requête SQL ». Il n'y a plus qu'à taper vos requêtes dans la fenêtre.

Dans le cas d'un SGBD distribué, si vous disposez du client « mysql-client » (sous linux) : "mysql" ou, pour travailler sous un nom d'utilisateur (avec mot de passe) : "mysql -u nomuser -p" ou, pour travaille sur un serveur distant : "mysql -h hôte -u nomuser -p".

On peut utiliser une interface Web, par exemple phpMyAdmin. Cela oblige à mettre en place un serveur Web sur le serveur SGBD. Mais c'est d'un usage très intuitif, et ce sera peut-être de toute façon nécessaire pour l'usage par les utilisateurs finaux. Là aussi, il suffit de trouver l'icône « SQL » (en vert, dans le menu principal à gauche), et de taper les requêtes dans la fenêtre.

Il y a dans SQL toute une gestion des utilisateurs, avec définition de droits divers pour chaque table, (grâce à la commande GRANT ACCESS), mais nous n'en parlerons pas ici, car c'est notre administrateur réseau qui le gère (et il n'a plus besoin de lire ce document). En tout cas, il doit vous avoir attribué une base de données, un nom d'utilisateur (quelquefois le même que la base) et un mot de passe.

4.2) Connexion à une base

- Pour se connecter à une base : `CONNECT base;`
- `SHOW DATABASES;` pour voir lesquelles existent (du moins celles auxquelles on a accès)
- `CREATE DATABASE base;` pour la créer
- `DROP DATABASE base;` pour l'effacer

4.3) Créer une table

```
CREATE TABLE voitures (
      Numero VARCHAR(12),
      Marque VARCHAR(15), Modele VARCHAR(15),
      Serie VARCHAR(20), Km INTEGER, Prix FLOAT,
      MiseEnCirculation DATE,
      PRIMARY KEY(Numero)
   );
```

Les types possibles sont `CHAR(n)` (ou `CHARACTER`, chaîne de caractères de longueur n fixe), `VARCHAR(n)` (ou `VARYING CHARACTER`) si la longueur est variable (<n), `INT` (ou `INTEGER`, entier 32 bits, pouvant valoir + ou - 4G), `SMALLINT` (entier 16 bits, + ou - 32K), `FLOAT` (nombre à virgule), `NUMERIC(n)` ou `NUMERIC(n,d)` (entiers ou flottants dont un veut définir exactement le nombre n de chiffres dont d après la virgule), `DATE` (2006-02-28), `TIME` (11:59:59.99) `TIMESTAMP` (date+time), et des tas d'autres qui dépendent du SGBD. En cas d'ambiguïté, un champ (ou une donnée) peut être encadré de ' (simples quotes) ou " (doubles quotes). C'est obligatoire pour les textes, les noms avec espaces, caractères accentués ou spéciaux.

Chaque champ peut avoir des attributs : `UNIQUE` (pas deux identiques dans toute la colonne, c'est ainsi que l'on définit une clé **non** primaire), `NOT NULL` (obligatoire) ou `NULL` (champ facultatif, par défaut), `DEFAULT(valeur)` (pour donner une valeur par défaut, par ex 0), `AUTO_INCREMENT` (donne automatiquement une nouvelle valeur si vous n'en donnez pas, utile pour les clés), `PRIMARY KEY` (clé primaire, automatiquement NOT NULL et UNIQUE), `REFERENCES nomtable( champ_référencé)` permet de définir une clé étrangère, en spécifiant la table et la colonne cible (la clé primaire si on ne spécifie rien).

autre exemple, avec attributs :

```
CREATE TABLE clients(
      Ref       INTEGER PRIMARY KEY DEFAULT 1 AUTO_INCREMENT,  --la clé primaire
      Nom       VARCHAR(30) NOT NULL DEFAULT "---",
      SaVoiture VARCHAR(12) REFERENCES voitures(Numero)  --une clé étrangère
      ...etc…       );
```

En général il faut une clé primaire dans chaque table. On peut donner l'attribut PRIMARY KEY à un champ, ou utiliser la forme d'une « contrainte » : voir dans l'exemple "voitures" . Cette forme permet de spécifier plusieurs champs (par exemple `PRIMARY KEY(Nom,Prenom)` accepte des frères (même nom) ou des prénoms identiques, mais refuse un deuxième couple nom,prénom identique). De même, on peut définir une clé étrangère par la contrainte `FOREIGN KEY (nomchamp) REFERENCES nomtable(clés) ;` (si on ne précise pas de clé, ce sera la clé primaire).

4.4) Autres instructions de gestion des tables

- `SHOW TABLES ;` liste toutes les tables de la base
- `RENAME TABLE vieux_nom TO nouveau_nom ;`
- `DROP TABLE nom_table ;` suppression d'une table (attention pas de CTRL Z)
- `DESCRIBE table ;` retrouver la définition des champs d'une table
- `ALTER TABLE nom_table ADD COLUMN nom type ;` ajouter un champ
- `ALTER TABLE nom_table DROP COLUMN nom ;` supprimer une colonne
- `ALTER TABLE nom_table ALTER COLUMN nom_champ SET nom_attribut ;` ajouter un attribut de champ : default, primary key, unique....
- `ALTER TABLE nom_table ALTER COLUMN nom_champ DROP nom_attribut ;` supprimer un attribut
- `ALTER TABLE nomtable MODIFY COLUMN nomchamp <nouveau_type> ;`

4.5) Alimenter la table

```
INSERT INTO voitures
VALUES      ('AA 111 BB','Renault','Twingo','1.2L',49700,4700,'2001-08-01'),
      ('4444 ZA 67','Renault','Twingo','expression',102000,3200,'2008-5-15'),
      ('BB 231 ZA','Renault','Espace','DCI',85000,15900,'2007-01-18'),
      ('BZ 211 DF','Renault','Espace','privilege',17000,23900,'2010-05-12'),
      ('1252 XC 67','Renault','Espace','TD',257000,2900,'1999-04-22'),
      ('AA 121 GC','Ford','Fiesta','1.4 CLX',67000,3000,'2001-08-27' ),
      ('BC 345 DF','Volkswagen','Golf','TDI 100CV',45000,11000,'2009-9-15'),
      ('5555 BCD 57','Volkswagen','Coccinelle',NULL,45000,NULL,'1972-9-15')
;
```

Ici, on donne tous les champs, dans l'ordre de leur définition. Mais si on veut les donner dans un autre ordre, ou si l'on ne se rappelle plus de l'ordre, ou qu'on veuille laisser vide (NULL) certains champs : `INSERT INTO nomtable(col1,col2,col3) VALUES(val1,val2,val3);`

4.6) Sélection dans une table

SELECT permet d'extraire des données dans la base. Le principe est de définir ce qu'on cherche en ne sélectionnant que la partie des données qui nous intéresse. La syntaxe de base est : `SELECT choix des champs FROM choix des tables WHERE limitation des lignes ;` : on dit quelles tables, quelles colonnes, quelles lignes (pour les deux premiers on donne leur nom, pour le dernier on donne des conditions que doivent respecter les lignes qui nous intéressent). Le format plus général est le suivant :

```
SELECT [DISTINCT] <liste des noms de colonnes> | * | <calcul>
      FROM <Liste des tables>
      [WHERE <condition logique>]
      [GROUP BY...]
      [HAVING...]
      [ORDER BY <nom_ou_numéro_de_colonne> [ASC|DESC] ]
```

Les conditions logiques peuvent utiliser entre autres : les opérateurs logiques AND, OR, NOT ; les comparateurs arithmétiques (=, != ou <>, >, <, >=, <=), les comparateurs de chaîne (IN, BETWEEN, LIKE), des opérateurs arithmétiques (+, -, *, /, %, &, -, ^, ~), des opérateurs pour chaînes de caractères (CONCAT(liste args), SUBSTRING, LOWER, UPPER...). Nous allons détailler cela petit à petit, à l'aide d'exemples :

```
SELECT * FROM voitures; /* tous les champs, aucune restriction sur les lignes*/
```

Numero	Marque	Modele	Serie	Km	Prix	MiseEnCirculation
1252 XC 67	Renault	Espace	TD	257000	2900	1999-04-22
4444 ZA 67	Renault	Twingo	expression	102000	3200	2008-05-15
5555 BCD 57	Volkswagen	Coccinelle	NULL	45000	NULL	1972-09-15
AA 111 BB	Renault	Twingo	1.2L	49700	4700	2001-08-01
AA 121 GC	Ford	Fiesta	1.4 CLX	67000	3000	2001-08-27
BB 231 ZA	Renault	Espace	DCI	85000	15900	2007-01-18
BC 345 DF	Volkswagen	Golf	TDI 100CV	45000	11000	2009-09-15
BZ 211 DF	Renault	Espace	privilege	17000	23900	2010-05-12

```
SELECT Marque,Km,CONCAT(Prix,' €') AS Euros FROM voitures; /* j'ai changé un titre
qui aurait été peu compréhensible */
```

Marque	Km	Euros
Renault	257000	2900 €
Renault	102000	3200 €
Volkswagen	45000	NULL
Renault	49700	4700 €
Ford	67000	3000 €
Renault	85000	15900 €
Volkswagen	45000	11000 €
Renault	17000	23900 €

```
SELECT * FROM voitures WHERE (Km<100000) AND (MiseEnCirculation>'2008-01-01');
```

Numero	Marque	Modele	Serie	Km	Prix	MiseEnCirculation
BC 345 DF	Volkswagen	Golf	TDI 100CV	45000	11000	2009-09-15
BZ 211 DF	Renault	Espace	privilege	17000	23900	2010-05-12

```
SELECT * FROM voitures WHERE Km BETWEEN 50000 AND 80000;
```

Numero	Marque	Modele	Serie	Km	Prix	MiseEnCirculation
AA 121 GC	Ford	Fiesta	1.4 CLX	67000	3000	2001-08-27

```
SELECT * FROM voitures WHERE Marque IN ('Renault','Ford');
```

Numero	Marque	Modele	Serie	Km	Prix	MiseEnCirculation
1252 XC 67	Renault	Espace	TD	257000	2900	1999-04-22
4444 ZA 67	Renault	Twingo	expression	102000	3200	2008-05-15
AA 111 BB	Renault	Twingo	1.2L	49700	4700	2001-08-01
AA 121 GC	Ford	Fiesta	1.4 CLX	67000	3000	2001-08-27
BB 231 ZA	Renault	Espace	DCI	85000	15900	2007-01-18
BZ 211 DF	Renault	Espace	privilege	17000	23900	2010-05-12

```
SELECT * FROM voitures WHERE Marque NOT LIKE 'R_n%'; /* _ = 1CHAR, % 0 OU PLUS*/
```

Numero	Marque	Modele	Serie	Km	Prix	MiseEnCirculation
5555 BCD 57	Volkswagen	Coccinelle	NULL	45000	NULL	1972-09-15
AA 121 GC	Ford	Fiesta	1.4 CLX	67000	3000	2001-08-27
BC 345 DF	Volkswagen	Golf	TDI 100CV	45000	11000	2009-09-15

```
SELECT * FROM voitures WHERE Numero LIKE '%67';
```

Numero	Marque	Modele	Serie	Km	Prix	MiseEnCirculation
1252 XC 67	Renault	Espace	TD	257000	2900	1999-04-22
4444 ZA 67	Renault	Twingo	expression	102000	3200	2008-05-15

```
SELECT * FROM voitures WHERE Prix IS NULL;
```

Numero	Marque	Modele	Serie	Km	Prix	MiseEnCirculation
5555 BCD 57	Volkwagen	Coccinelle	NULL	45000	NULL	1972-09-15

```
SELECT * FROM voitures ORDER BY prix;
```

```
+-------------+------------+------------+------------+--------+-------+-------------------+
| Numero      | Marque     | Modele     | Serie      | Km     | Prix  | MiseEnCirculation |
+-------------+------------+------------+------------+--------+-------+-------------------+
| 5555 BCD 57 | Volkswagen | Coccinelle | NULL       |  45000 |  NULL | 1972-09-15        |
| 1252 XC 67  | Renault    | Espace     | TD         | 257000 |  2900 | 1999-04-22        |
| AA 121 GC   | Ford       | Fiesta     | 1.4 CLX    |  67000 |  3000 | 2001-08-27        |
| 4444 ZA 67  | Renault    | Twingo     | expression | 102000 |  3200 | 2008-05-15        |
| AA 111 BB   | Renault    | Twingo     | 1.2L       |  49700 |  4700 | 2001-08-01        |
| BC 345 DF   | Volkswagen | Golf       | TDI 100CV  |  45000 | 11000 | 2009-09-15        |
| BB 231 ZA   | Renault    | Espace     | DCI        |  85000 | 15900 | 2007-01-18        |
| BZ 211 DF   | Renault    | Espace     | privilege  |  17000 | 23900 | 2010-05-12        |
+-------------+------------+------------+------------+--------+-------+-------------------+
SELECT * FROM voitures ORDER BY Marque ASC, Km DESC;
+-------------+------------+------------+------------+--------+-------+-------------------+
| Numero      | Marque     | Modele     | Serie      | Km     | Prix  | MiseEnCirculation |
+-------------+------------+------------+------------+--------+-------+-------------------+
| 5555 BCD 57 | Volkswagen | Coccinelle | NULL       |  45000 |  NULL | 1972-09-15        |
| 1252 XC 67  | Renault    | Espace     | TD         | 257000 |  2900 | 1999-04-22        |
| AA 121 GC   | Ford       | Fiesta     | 1.4 CLX    |  67000 |  3000 | 2001-08-27        |
| 4444 ZA 67  | Renault    | Twingo     | expression | 102000 |  3200 | 2008-05-15        |
| AA 111 BB   | Renault    | Twingo     | 1.2L       |  49700 |  4700 | 2001-08-01        |
| BC 345 DF   | Volkswagen | Golf       | TDI 100CV  |  45000 | 11000 | 2009-09-15        |
| BB 231 ZA   | Renault    | Espace     | DCI        |  85000 | 15900 | 2007-01-18        |
| BZ 211 DF   | Renault    | Espace     | privilege  |  17000 | 23900 | 2010-05-12        |
+-------------+------------+------------+------------+--------+-------+-------------------+
```

On peut faire des calculs sur une colonne : moyenne AVG, nb de lignes COUNT, MAX, MIN, total SUM. Seules les valeurs existantes (non NULL) sont prises en compte (et bien sûr seules celles qui seront affichées d'après le where). GROUP BY permet de regrouper des lignes d'après un critère, c'est utile avec les calculs. Et on peut rajouter une clause HAVING qui peut restreindre les lignes prises en compte dans le groupage (mais qui seront néanmoins prises en compte dans les calculs) :

```
SELECT COUNT(*) AS nb, AVG(Prix) AS Prix_moy, SUM(Prix) AS Immobilisation FROM
voitures;
+----+------------------+----------------+
| nb | Prix_moy         | Immobilisation |
+----+------------------+----------------+
|  8 | 9228.57142857143 |          64600 |
+----+------------------+----------------+
SELECT Marque, AVG(Km) AS Km_Moy, count(*) AS nb FROM voitures GROUP BY Marque;
+------------+-------------+----+
| Marque     | Km_Moy      | nb |
+------------+-------------+----+
| Ford       |  67000.0000 |  1 |
| Renault    | 102140.0000 |  5 |
| Volkswagen |  45000.0000 |  2 |
+------------+-------------+----+
SELECT Marque,Modele,COUNT(Numero) FROM voitures GROUP BY Marque,Modele;
+------------+------------+---------------+
| Marque     | Modele     | COUNT(Numero) |
+------------+------------+---------------+
| Ford       | Fiesta     |             1 |
| Renault    | Espace     |             3 |
| Renault    | Twingo     |             2 |
| Volkswagen | Coccinelle |             1 |
| Volkswagen | Golf       |             1 |
+------------+------------+---------------+
SELECT Marque, AVG(Km) AS Km_Moy FROM voitures GROUP BY Marque ORDER BY Km_Moy;
+------------+-------------+
| Marque     | Km_Moy      |
+------------+-------------+
| Volkswagen |  45000.0000 |
| Ford       |  67000.0000 |
| Renault    | 102140.0000 |
+------------+-------------+
SELECT Marque, SUM(Prix) as Total, AVG(Km) AS Km_Moy FROM voitures
       GROUP BY Marque HAVING Total>15000;
+---------+-------+-------------+
| Marque  | Total | Km_Moy      |
+---------+-------+-------------+
| Renault | 50600 | 102140.0000 |
+---------+-------+-------------+
```

DISTINCT n'écrit qu'une fois les lignes exactement identiques. Comparez (on a dans la table 2 twingo et 3 espaces) :

```
SELECT Marque,Modele FROM voitures
ORDER BY Marque,Modele;
```

```
+------------+------------+
| Marque     | Modele     |
+------------+------------+
| Ford       | Fiesta     |
| Renault    | Espace     |
| Renault    | Espace     |
| Renault    | Espace     |
| Renault    | Twingo     |
| Renault    | Twingo     |
| Volkswagen | Coccinelle |
| Volkswagen | Golf       |
+------------+------------+
```

```
SELECT DISTINCT Marque,Modele FROM
voitures ORDER BY Marque,Modele;
```

```
+------------+------------+
| Marque     | Modele     |
+------------+------------+
| Ford       | Fiesta     |
| Renault    | Espace     |
| Renault    | Twingo     |
| Volkswagen | Coccinelle |
| Volkswagen | Golf       |
+------------+------------+
```

On peut même faire des sous-requêtes (du moins, dans presque tous les SGBD). Attention, on ne peut utiliser, à un endroit où la syntaxe de SQL n'accepte qu'un valeur, qu'une sous-requête n'envoyant qu'un résultat. Sil y a plusieurs résultats, on peut utiliser l'opérateur IN (ou EXISTS, ALL, ANY). Les sous-requêtes sont surtout utiles lorsque l'on à plusieurs tables (voir plus loin).

```
SELECT * FROM voitures WHERE Prix < (SELECT AVG(Prix) FROM voitures);
+------------+---------+--------+------------+--------+------+-------------------+
| Numero     | Marque  | Modele | Serie      | Km     | Prix | MiseEnCirculation |
+------------+---------+--------+------------+--------+------+-------------------+
| 1252 XC 67 | Renault | Espace | TD         | 257000 | 2900 | 1999-04-22        |
| 4444 ZA 67 | Renault | Twingo | expression | 102000 | 3200 | 2008-05-15        |
| AA 111 BB  | Renault | Twingo | 1.2L       |  49700 | 4700 | 2001-08-01        |
| AA 121 GC  | Ford    | Fiesta | 1.4 CLX    |  67000 | 3000 | 2001-08-27        |
+------------+---------+--------+------------+--------+------+-------------------+
SELECT * FROM voitures WHERE Prix = (SELECT MAX(Prix) FROM voitures);
+-----------+---------+--------+-----------+-------+-------+-------------------+
| Numero    | Marque  | Modele | Serie     | Km    | Prix  | MiseEnCirculation |
+-----------+---------+--------+-----------+-------+-------+-------------------+
| BZ 211 DF | Renault | Espace | privilege | 17000 | 23900 | 2010-05-12        |
+-----------+---------+--------+-----------+-------+-------+-------------------+
```

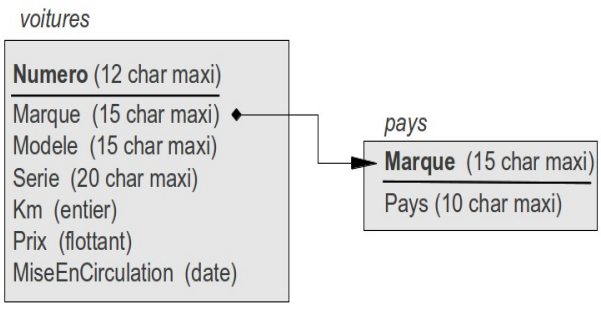
4.7) Jointure (plusieurs tables) et relations.

Au paragraphe précédent, nous n'avons utilisé la sélection que sur des données toutes dans une même table. Nous allons désormais utiliser plusieurs tables. Il n'y a aucune difficulté dans le principe : il suffit de mettre plusieurs noms de tables dans le FROM. Quand les champs ont des noms différents dans les deux tables, il n'y a rien à changer. Par contre, si l'on a le même nom de colonne dans deux tables, on précise « nom_table.nom_champ ». Mais c'est quand on utilise des relations que c'est (un peu) plus compliqué.

4.7.1) Créons une seconde table

Revenons à notre exemple. Pour beaucoup d'acheteurs de véhicules, le pays d'origine est un critère de choix important (l'objectif de ce cours n'est pas de disserter sur la mondialisation, et bien que ce soit un problème très important, je n'en parlerai pas ici). Il est bien sûr hors de question de devoir saisir le pays d'origine pour toutes les voitures que nous gérerons. Nous allons créer une nouvelle table, la table Pays, qui pour chaque marque nous donnera son pays :

```
CREATE TABLE pays (Marque VARCHAR(15) PRIMARY KEY, Pays VARCHAR(10));
insert into pays VALUES('Volkswagen','Allemagne');
insert into pays VALUES('Renault','France');
insert into pays VALUES('Peugeot','France');
insert into pays VALUES('Fiat','Italie');
insert into pays VALUES('Ford','USA');
```



Vérifions le contenu de la table :

```
SELECT * FROM pays;
+------------+-----------+
| Marque     | Pays      |
+------------+-----------+
| Fiat       | Italie    |
| Ford       | USA       |
| Peugeot    | France    |
```

```
| Renault    | France    |
| Volkswagen | Allemagne |
+------------+-----------+
```

4.7.2) Réalisation de la jointure dans un select

Si maintenant on demande de sélectionner des données des deux tables, il essaye TOUTES les combinaisons possibles C'est le « cross join », ou « produit cartésien ». Comme j'ai 8 voitures et 5 pays, ça me fait 40 lignes :

```
SELECT Modele, Pays FROM voitures, pays;
```

```
+------------+-----------+
| Modele     | Pays      |
+------------+-----------+
| Espace     | Italie    |
| Espace     | USA       |
| Espace     | France    |
| Espace     | France    |
| Espace     | Allemagne |
| Twingo     | Italie    |
| Twingo     | USA       |
| Twingo     | France    |
etc... (40 lignes)
```

On peut aussi utiliser la syntaxe :

```
SELECT Modele, Pays FROM voitures
                    CROSS JOIN pays;
```

évidement, ici on voulait restreindre ce nombre de lignes, en ne sélectionnant que celles qui respectent la relation entre les deux tables (c'est l'« inner join ») :

```
SELECT DISTINCT Modele, Pays FROM voitures,pays WHERE voitures.Marque = pays.Marque;
```

```
+------------+-----------+
| Modele     | Pays      |
+------------+-----------+
| Fiesta     | USA       |
| Espace     | France    |
| Twingo     | France    |
| Coccinelle | Allemagne |
| Golf       | Allemagne |
+------------+-----------+
```

On peut aussi utiliser la syntaxe (résultat identique) :

```
SELECT DISTINCT Modele, Pays FROM voitures
INNER JOIN pays ON voitures.Marque = pays.Marque;
```

et même, puisqu'ici on a donné le même nom de champ à la clé primaire et étrangère (et pas aux autres champs) :

```
SELECT DISTINCT Modele, Pays FROM voitures
NATURAL JOIN pays ;
```

On n'est obligé de mettre le nom de la table (devant un champ, séparé par un .) que si le nom est identique dans 2 tables. Mais on peut aussi donner un raccourci au nom des tables

```
SELECT v.Marque, v.Modele, p.Pays, COUNT(Numero) AS Nb FROM voitures v,pays p WHERE
v.Marque = p.Marque GROUP BY v.Marque, v.Modele;
+------------+------------+-----------+----+
| Marque     | Modele     | Pays      | Nb |
+------------+------------+-----------+----+
| Ford       | Fiesta     | USA       |  1 |
| Renault    | Espace     | France    |  3 |
| Renault    | Twingo     | France    |  2 |
| Volkswagen | Coccinelle | Allemagne |  1 |
| Volkswagen | Golf       | Allemagne |  1 |
+------------+------------+-----------+----+
```

4.7.3 Jointure par cascade de select

Si je ne veux afficher que les voitures françaises, je peux faire la jointure comme précisé précédemment :

```
SELECT v.* FROM voitures v,pays p WHERE v.Marque = p.Marque AND p.Pays='France';
```

mais on peut faire un select par table : Dans « pays » je cherche les marques françaises, puis dans « voitures » je cherche ces marques :

```
SELECT * FROM voitures WHERE Marque IN (SELECT Marque FROM pays WHERE Pays='France');
```

Dans les deux cas, on obtient exactement la même chose :

```
+------------+---------+--------+------------+--------+-------+-------------------+
| Numero     | Marque  | Modele | Serie      | Km     | Prix  | MiseEnCirculation |
+------------+---------+--------+------------+--------+-------+-------------------+
| 1252 XC 67 | Renault | Espace | TD         | 257000 |  2900 | 1999-04-22        |
| 4444 ZA 67 | Renault | Twingo | expression | 102000 |  3200 | 2008-05-15        |
| AA 111 BB  | Renault | Twingo | 1.2L       |  49700 |  4700 | 2001-08-01        |
| BB 231 ZA  | Renault | Espace | DCI        |  85000 | 15900 | 2007-01-18        |
```

```
| BZ 211 DF  | Renault | Espace | privilege  |  17000 | 23900 | 2010-05-12        |
+------------+---------+--------+------------+--------+-------+-------------------+
```

Il est parfois plus simple de faire une cascade de requêtes (limitées à une table à la fois) plutôt qu'une seule requête sur l'ensemble des tables (surtout quand on a beaucoup de tables).

4.7.4) SGBD Relationnel

Jusque là, j'ai utilisé le principe de jointure, mais je n'ai pas utilisé la partie relationnelle de mon SGBDR. Si je saisis une nouvelle voiture, d'une marque non encore définie dans la table pays, cela ne le gêne pas :

```
INSERT INTO voitures VALUES ('BB 111 GH','Honda','Civic','1.8 VTEC',32700,12700,'2010-08-01');
```

je l'enlève pour l'instant :

```
DELETE FROM voitures WHERE Marque='Honda';
```

je définis la relation :

```
ALTER TABLE voitures ADD FOREIGN KEY (Marque) REFERENCES pays(Marque);
```

et désormais il ne veut plus du INSERT précédent. Il faut donc en premier :

```
INSERT INTO pays VALUES('Honda','Japon');
```

et désormais je peux insérer autant de Honda que je veux. Tant que j'ai des Honda dans la table voitures, il m'empêchera de le supprimer dans la table pays :

```
DELETE FROM pays WHERE Marque='Honda';
```

En conclusion, le fait de définir proprement les relations dans une base empêchera les incohérences dans les données.

4.8) Divers

4.8.1) Gestion des tables

On supprime des lignes par `DELETE FROM table WHERE condition;` (on ne peut plus restaurer, attention à ce qui est derrière WHERE car on peut vite tout enlever!)

On peut aussi alimenter une table par un SELECT, en fonction de données existantes:

```
INSERT INTO table(col1,col2) SELECT col1,col2 FROM xxx WHERE......;
```

Modifier tout le contenu d'une table :

```
UPDATE nomtable SET colonne='valeur'... WHERE conditions;
exemple : UPDATE produits SET TVA=21.6 WHERE TVA=20.6;
```

4.8.2) Requêtes particulières (peut être spécifiques MySQL)

Comment afficher l'age quand je dispose de la date de naissance ? Ajouter dans le Select le champ :

```
(YEAR(CURRENT_DATE)-YEAR(naissance)) - (RIGHT(CURRENT_DATE,5) < RIGHT(naissance,5)) AS age
```

Plutôt que d'imbriquer des sous requêtes, on peut utiliser des variables internes :

```
SELECT @ma_variable:=MAX(Prix) FROM voitures;
SELECT * FROM voitures WHERE Prix = @ma_variable;
```

Ou alors, on peut passer par un tableau temporaire (on peut utiliser le mot clef TEMPORARY mais ce n'est pas obligé si on n'oublie pas le DROP TABLE) :

```
CREATE TEMPORARY TABLE tmp
      (Marque VARCHAR(15),Modele VARCHAR(15), Nb INTEGER);
INSERT INTO tmp SELECT Marque,Modele,COUNT(Numero) FROM voitures
      GROUP BY Marque,Modele;
SELECT V.* FROM voitures V,tmp
      WHERE V.Marque=tmp.Marque AND V.Modele=tmp.Modele AND tmp.Nb>1;
DROP TABLE tmp;
```

Table des matières