

Chapitre 4 : Définition des données

Ce chapitre décrit les instructions SQL qui constituent l'aspect **LDD** (langage de définition des données). A cet effet, nous verrons notamment comment déclarer une table avec ses éventuels index et contraintes.

4.1. Tables relationnelles

Une table est créée en SQL par l'instruction CREATE TABLE, modifiée au niveau de sa structure par l'instruction ALTER TABLE et supprimée par la commande DROP TABLE.

4.1.1. Création d'une table [CREATE TABLE]

Remarque : pour pouvoir créer une table dans votre base, il faut que vous ayez reçu le privilège CREATE.

La syntaxe SQL est la suivante :

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] [nomBase.]nomTable
    (colonne1 type1 [NOT NULL | NULL] [DEFAULT valeur 1] [COMMENT 'chaine1']
    [, colonne2 type2 [NOT NULL | NULL] [DEFAULT valeur2] [COMMENT 'chaine2']]
    [CONSTRAINT nomContrainte1 typeContrainte1] ...)
    [ENGINE= InnoDB | MyISAM | ...];
```

- ✚ **TEMPORARY :** permet de créer une table qui **n'existera que durant la session courante** (la table sera supprimée à la déconnexion). Deux connexions peuvent ainsi créer deux tables temporaires de même nom sans risquer de conflit. *Pour utiliser cette option, il est indispensable de disposer du CREATE TEMPORARY TABLES.*
- ✚ **IF NOT EXISTS :** permet d'éviter qu'une erreur se produise si la table existe déjà (si c'est le cas, elle n'est aucunement affectée par la tentative de création).
- ✚ **nomBase :** (jusqu'à 64 caractères permis dans un nom de répertoire ou de fichier sauf « / », « \ » et « . ») s'il est omis, il sera assimilé à la base connectée. S'il est précisé, il désigne soit la base connectée soit une autre base (dans ce cas, il faut que l'utilisateur courant ait le droit de créer une table dans l'autre base). Nous aborderons ces points dans le cours suivant « Gestion et exploitation de bases de données. » et nous considérerons jusque-là que nous travaillons dans la base courante.
- ✚ **nomTable :** mêmes limitations que pour le nom de la base.
- ✚ **colonne type :** nom d'une colonne (mêmes caractéristiques que pour les noms des tables) et son type (INTEGER, CHAR, DATE, ...). Nous verrons quels types sont disponibles sous MySQL. La directive DEFAULT fixe une valeur par défaut. La directive NOT NULL interdit que la valeur de la colonne soit nulle.
- ✚ **COMMENT :** (jusqu'à 60 caractères) permet de commenter une colonne. Ce texte sera ensuite automatiquement affiché à l'aide des commandes SHOW CREATE TABLE et SHOW FULL COLUMNS (nous aborderons ces points dans le cours suivant).

- ✚ **nomContrainte typeContrainte** : nom de la contrainte et son type (clé primaire, clé étrangère, etc.).
- ✚ **ENGINE** : définit le type de table (par défaut InnoDB, bien adapté à la programmation de transactions). MyISAM correspond au type par défaut des versions 3, parfaitement robuste mais ne supportant pas pour l'heure l'intégrité référentielle. D'autres types existent, citons MEMORY pour les tables temporaires, ARCHIVE, etc.
- ✚ **« ; »** : symbole par défaut qui termine une instruction MySQL en mode ligne de commande

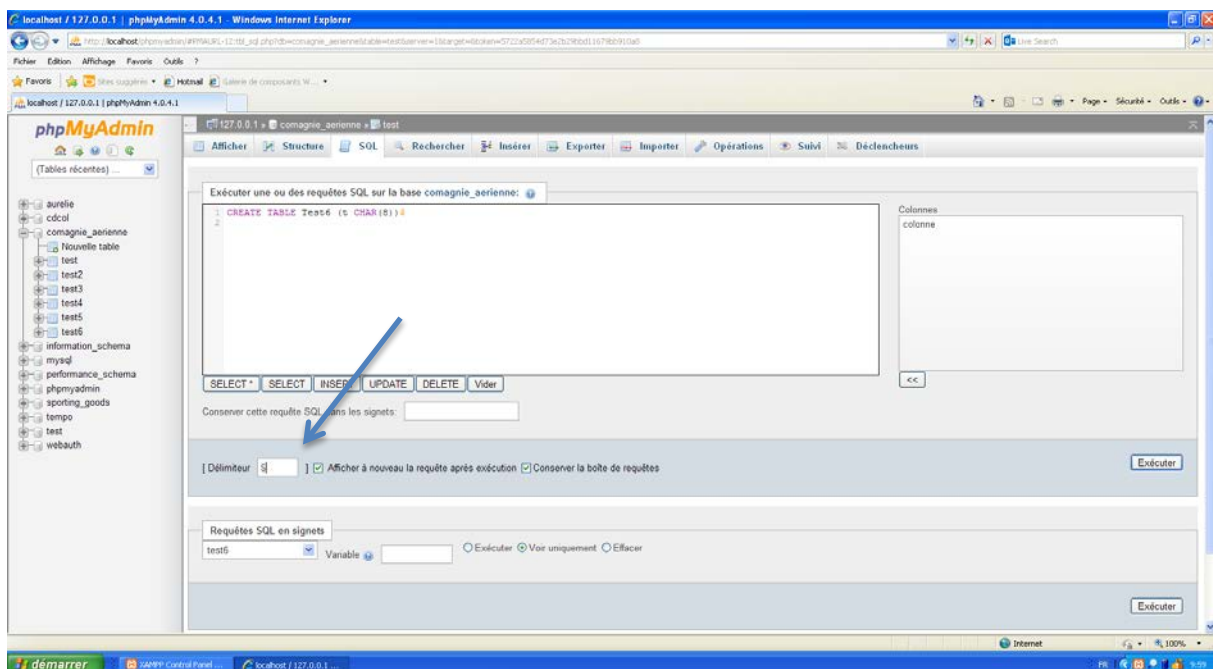
Remarque : NULL représente une valeur que l'on peut considérer comme non disponible, non affectée, inconnue ou inapplicable. Elle est différente d'un espace pour un caractère ou d'un zéro pour un nombre.

4.1.2. Délimiteurs

En mode ligne de commande, il est possible (par la directive delimiter) de choisir le symbole qui terminera chaque instruction. Dans l'exemple suivant, on choisit le dollar. Dans le cadre du cours, nous resterons avec le symbole par défaut de MySQL à savoir « ; ».

```
delimiter $
CREATE TABLE Test (t CHAR(8))$
```

Vous pouvez également le modifier au sein de l'environnement graphique :



4.1.3. Sensibilité à la casse

Alors que MySQL est sensible par défaut à la casse (au niveau des noms de base et de table) dans la plupart des distributions Unix, il ne l'est pas pour Windows. En revanche, concernant les noms de colonnes, index, alias de colonnes, déclencheurs et procédures cataloguées, MySQL n'est pas

sensible à la casse tous systèmes confondus. En fait, tous ces noms sont stockés en minuscules dans le dictionnaire de données.

La variable `lower_case_table_names` permet de forcer la sensibilité à la casse pour les noms des tables et des bases de données. Si elle vaut :

- ✚ 0 ➔ la sensibilité à la casse est active et les noms sont stockés en minuscules
- ✚ 1 ➔ pas de sensibilité à la casse et les noms sont stockés en minuscules
- ✚ 2 ➔ pas de sensibilité à la casse et les noms sont stockés en respectant la casse

Je vous invite à positionner cette variable à 0 de manière à homogénéiser le codage et à contrôler un peu plus l'écriture de vos instructions SQL. De plus, c'est l'option par défaut sur Linux. Dans le fichier `my.ini`, sous la section serveur identifiée par `[mysqld]`, ajouter la ligne et le commentaire suivants :

```
# Rend sensible à la CASSE les noms de tables et de database
lower_case_table_names=0
```

Refusez ce type de programmation (rendue impossible d'ailleurs si la variable `lower_case_table_names` est positionnée à 0).

```
SELECT * FROM Avion WHERE AVION.capacite > 150;
```

```
# Example MySQL config file for small systems.
#
# This is for a system with little memory (<= 64M) where MySQL is only used
# from time to time and it's important that the mysqld daemon
# doesn't use much resources.
#
# You can copy this file to
# C:/xampp/mysql/bin/my.cnf to set global options,
# mysql-data-dir/my.cnf to set server-specific options (in this
# installation this directory is C:/xampp/mysql/data) or
# ~/.my.cnf to set user-specific options.
#
# In this file, you can use all long options that a program supports.
# If you want to know which options a program supports, run the program
# with the "--help" option.

# The following options will be passed to all MySQL clients
[client]
# password      = your_password
port            = 3306
socket          = "C:/xampp/mysql/mysql.sock"

# Here follows entries for some specific programs

# The MySQL server
[mysqld]
# Rend sensible à la CASSE les noms de tables et de database <== ajt
lower_case_table_names=0                                <== ajt
|
port= 3306
socket = "C:/xampp/mysql/mysql.sock"
basedir = "C:/xampp/mysql"
tmpdir = "C:/xampp/tmp"
datadir = "C:/xampp/mysql/data"
pid_file = "mysql.pid"
# enable-named-pipe
key_buffer = 16M
max_allowed_packet = 1M
sort_buffer_size = 512K
net_buffer_length = 8K
```

Par ailleurs, la casse devrait toujours avoir (quel que soit le SGBD concerné) une incidence majeure dans les expressions de comparaison entre colonnes et valeurs que ce soit dans une instruction SQL ou un test dans un programme.

Ainsi l'expression « nomComp = 'Air France' » a la même signification que l'expression « nomComp = 'AIR France' » avec MySQL. Horreur, oui.

Donc, si vous désirez vérifier la casse au sein même des données, il faudra utiliser la fonction `BINARY()` qui convertit en bits une expression. En effet, « `BINARY ('AIR France')` » est différent de « `BINARY ('Air France')` » et « `BINARY (nomComp) = BINARY ('Air France')` » renverra vrai si tel est le cas en respectant la casse.




4.1.4. Commentaires

Dans toute instruction SQL (déclaration, manipulation, interrogation et contrôle des données), il est possible d'inclure des retours chariot, des tabulations, espaces et commentaires (sur une ligne précédée de deux tirets « - - », en fin de ligne à l'aide du dièse « # », au sein d'une ligne ou sur plusieurs lignes entre « / * » et « */ »). Les scripts suivants décrivent la déclaration d'une même table en utilisant différentes conventions :

Sans commentaire	Avec commentaires
<pre> 1 CREATE TABLE Test (2 colonne DECIMAL(38,8) 3); </pre>	<pre> 1 CREATE TABLE -- nom table 2 test4 (#test 3 col integer 4); </pre>
	<pre> 1 CREATE TABLE -- nom table 2 test3 (-- test 3 col integer 4) 5 -- fin 6 ; </pre>
	<pre> 1 CREATE TABLE Test /*nom de la table*/ 2 (3 /* definition des champs plusieurs */ 4 colonne /* type */ DECIMAL(38,8) 5); </pre>

Tableau 1-1 : Différentes écritures SQL

Bonne pratique : dans le cadre de ce cours, il est préférable d'utiliser les conventions suivantes :

-  **Tous les mots-clés de SQL sont notés en majuscules.**
-  **Les noms de tables sont notés en Minuscules (excepté la première lettre, ces noms seront quand même stockés dans le système en minuscules).**
-  **Les noms de colonnes et de contraintes en minuscules.**

L'adoption de ces conventions rendra vos requêtes, scripts et programmes plus lisibles (un peu comme en programmation).

4.1.5. Premier exemple

Le tableau ci-après décrit l'instruction SQL qui permet de créer la table *Compagnie* illustrée par la figure suivante dans la base *compagnie_aerienne* (l'absence du préfixe « *compagnie_aerienne* » conduirait au même résultat si *compagnie_aerienne* était la base connectée lors de l'exécution du script).

Compagnie				
comp	nrue	rue	ville	nomComp

Figure 1-1 : Table à créer

Instruction SQL

```
1 CREATE TABLE Compagnie(  
2     comp      VARCHAR(4) PRIMARY KEY,  
3     nrue      INTEGER(3),  
4     rue       VARCHAR(20),  
5     ville     VARCHAR(15) DEFAULT 'Paris' COMMENT 'Par défaut : Paris',  
6     nomComp   VARCHAR(15) NOT NULL,  
7 );
```

Ou

```
1 CREATE TABLE Compagnie(  
2     comp      VARCHAR(4),  
3     nrue      INTEGER(3),  
4     rue       VARCHAR(20),  
5     ville     VARCHAR(15) DEFAULT 'Paris' COMMENT 'Par défaut : Paris',  
6     nomComp   VARCHAR(15) NOT NULL,  
7     CONSTRAINT pk_Compagnie PRIMARY KEY(comp)  
8 );
```

Commentaires

La table contient cinq colonnes (quatre chaînes de caractères et un numérique de trois chiffres). La colonne *ville* est commentée. La table inclut en plus deux contraintes :

- ✚ DEFAULT qui fixe *Paris* comme valeur par défaut de la colonne *ville*
- ✚ NOT NULL qui impose une valeur non nulle dans la colonne *nomComp*

4.1.6. Contraintes

Les contraintes ont pour but de programmer des règles de gestion au niveau des colonnes des tables. Elles peuvent alléger un développement côté client (si on déclare qu'une note doit être comprise entre 0 et 20, les programmes de saisie n'ont plus à tester les valeurs en entrée mais seulement le code retour après connexion à la base, on déporte les contraintes côté serveur).

Les contraintes peuvent être déclarées de deux manières :

- ✚ En même temps que la colonne (valable pour les contraintes monocolonnes) : ces contraintes sont dites « en ligne » (inline constraints). L'exemple précédent en déclare deux.
- ✚ Après que la colonne soit déclarée : ces contraintes ne sont pas limitées à une colonne et peuvent être personnalisées par un nom (out-of-line constraints).

Il est recommandé de déclarer les contraintes NOT NULL en ligne, les autres peuvent soit être déclarées en ligne, soit être nommées. Etudions à présent les types de contraintes nommées (out-of-line). Les quatre types de contraintes les plus utilisées sont les suivants :

```
CONSTRAINT nomContrainte  
UNIQUE (colonne1 [, colonne2]...)  
PRIMARY KEY (colonne1 [, colonne2]...)  
FOREIGN KEY (colonne1 [, colonne2]...)  
REFERENCES nomTablePere [(colonne1 [, colonne2]...)]  
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]  
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]  
CHECK (condition)
```

- ✚ La contrainte **UNIQUE** impose une valeur distincte au niveau de la table (les valeurs nulles font exception à moins que NOT NULL soit aussi appliquée sur les colonnes).
- ✚ La contrainte **PRIMARY KEY** déclare la clé primaire de la table. Un index est généré automatiquement sur la ou les colonnes concernées. Les colonnes clés primaires ne peuvent être ni nulles ni identiques (en totalité si elles sont composées de plusieurs colonnes).
- ✚ La contrainte **FOREIGN KEY** déclare une clé étrangère entre une table enfant (*child*) et une table père (*parent*). Ces contraintes définissent l'intégrité référentielle que nous aborderons plus tard. Les directives ON UPDATE et ON DELETE disposent de quatre options que nous détaillerons plus loin dans le cours.

Remarque : la contrainte **CHECK** impose un domaine de valeurs à une colonne ou une condition (exemples: **CHECK** (note BETWEEN 0 AND 20), **CHECK** (grade='Copilote' OR grade='Commandant')). Il est permis de déclarer des contraintes **CHECK** lors de la création d'une table mais le mécanisme de vérification des valeurs n'est toujours pas implémenté.

Il est recommandé de ne pas définir de contraintes sans les nommer (bien que cela soit possible) car il sera difficile de les faire évoluer (désactivation, réactivation, suppression) et la lisibilité des programmes en sera affectée.

4.1.7. Conventions recommandées

Adoptez les conventions d'écriture suivantes pour vos contraintes :

- ✚ Préfixez par « **pk_** » le nom d'une contrainte clé primaire, « **fk_** » une clé étrangère, « **ck_** » une vérification, « **un_** » une unicité.
- ✚ Pour une contrainte clé primaire, suffixez du nom de la table la contrainte (exemple **pk_Avion**).

- ✚ Pour une contrainte clé étrangère, renseignez (ou abrégez) les noms de la table source, de la clé et de la table cible (exemple `fk_Pil_compa_Comp`).

En respectant nos conventions, déclarons les tables de l'exemple suivant (Compagnie avec sa clé primaire et Avion avec sa clé primaire et sa clé étrangère). Du fait de l'existence de la clé étrangère, la table Compagnie est dite « parent » (ou « père ») de la table Avion « enfant » (ou « fils »).

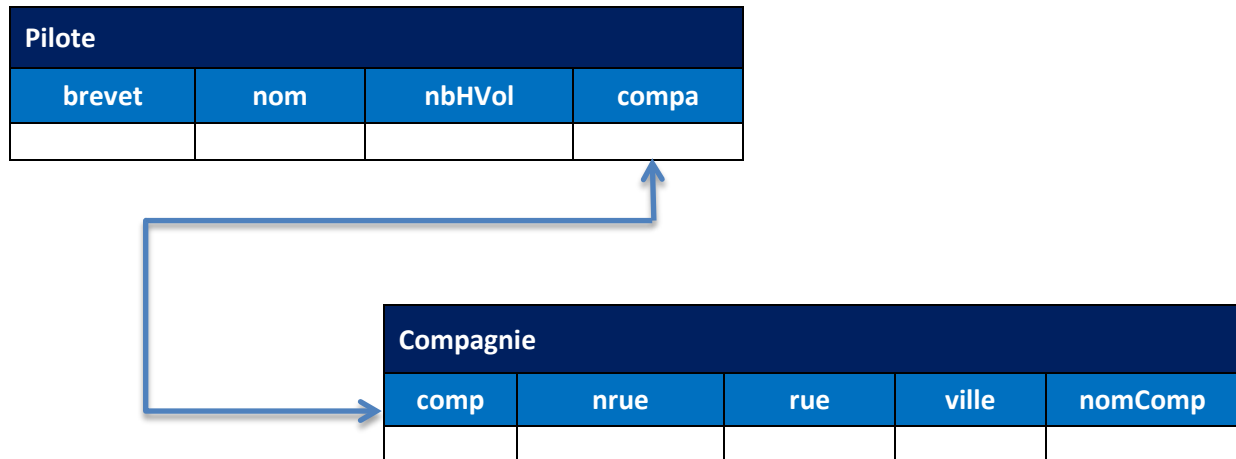


Figure 1-2 : Deux tables reliées à créer

Table : Compagnie

```

1 CREATE TABLE Compagnie(
2     comp      VARCHAR(4),
3     nrue      INTEGER(3),
4     rue       VARCHAR(20),
5     ville     VARCHAR(15) DEFAULT 'Paris' COMMENT 'Par défaut : Paris',
6     nomComp   VARCHAR(15) NOT NULL,
7     CONSTRAINT pk_Compagnie PRIMARY KEY(comp)
8 );

```

Contraintes de la table Compagnie

Deux contraintes en ligne et une contrainte nommée clé primaire

Table : Pilote

```

3 CREATE TABLE Pilote(
4     brevet    VARCHAR(6),
5     nom       VARCHAR(16),
6     nbHVol    DECIMAL(7,2),
7     compa     VARCHAR(4),
8     CONSTRAINT pk_Pilote PRIMARY KEY(brevet),
9     CONSTRAINT ck_nbHVol CHECK(nbHVol BETWEEN 0 AND 20000),
10    CONSTRAINT fk_Pil_compa_Comp FOREIGN KEY(compa) REFERENCES Compagnie(comp)
11 );

```

Contraintes de la table Pilote

Une contrainte en ligne et quatre contraintes nommées :

- ✚ Clé primaire

- ✚ NOT NULL
- ✚ CHECK (nombre d'heures de vol compris entre 0 et 20 000)
- ✚ UNIQUE (homonymes interdits)
- ✚ Clé étrangère

Remarque :

- ➡ L'ordre n'est pas important dans la déclaration des contraintes nommées.
- ➡ PRIMARY KEY équivaut à : UNIQUE + NOT NULL + index.
- ➡ L'ordre de création des tables est important quand on définit les contraintes en même temps que les tables (on peut différer la création ou l'activation des contraintes nous le verrons plus loin). Il faut créer d'abord les tables « pères » puis les tables « fils ». Le script de destruction des tables suit le raisonnement inverse.

4.1.8. Types de colonne

Pour décrire les colonnes d'une table, MySQL fournit les types prédéfinis suivants :

- ✚ caractères (CHAR, VARCHAR, TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT)
- ✚ valeurs numériques (TINYINT, SMALLINT, MEDIUMINT, INT, INTEGER, BIGINT, FLOAT, DOUBLE, REAL, DECIMAL, NUMERIC et BIT)
- ✚ date/heure (DATE, DATETIME, TIME, YEAR, TIMESTAMP)
- ✚ données binaires (BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB)
- ✚ énumérations (ENUM, SET).

Détaillons à présent ces types. Nous verrons comment utiliser les plus courants au fil du cours.

4.1.8.1. Caractères

Le type **CHAR** permet de stocker des chaînes de caractères de taille fixe. Les valeurs sont stockées en ajoutant, s'il le faut, des espaces à concurrence de la taille définie. Ces espaces ne seront pas considérés après extraction à partir de la table.

Le type **VARCHAR** permet de stocker des chaînes de caractères de taille variable. Les valeurs sont stockées sans l'ajout d'espaces à concurrence de la taille définie. Depuis la version 5.0.3 de MySQL, les éventuels espaces de fin de chaîne seront stockés et extraits en conformité avec la norme SQL. Des caractères Unicode (méthode de codage universelle qui fournit une valeur de code unique pour chaque caractère quel que soit la plate-forme, le programme ou la langue) peuvent aussi être stockés.

Les types **BINARY** et **VARBINARY** sont similaires à **CHAR** et **VARCHAR**, excepté par le fait qu'ils contiennent des chaînes d'octets sans tenir compte d'un jeu de caractères en particulier.

Les quatre types permettant aussi de stocker du texte sont **TINYTEXT**, **TEXT**, **MEDIUMTEXT** et **LONGTEXT**. Ces types sont associés à un jeu de caractères. Il n'y a pas de mécanisme de suppression d'espaces de fin, ni de possibilité d'y associer une contrainte DEFAULT.

Type	Description	Commentaire pour une colonne
CHAR(n) [BINARY ASCII UNICODE]	Chaîne fixe de n octets ou caractères.	Taille fixe (maximum de 255 caractères).
VARCHAR(n) [BINARY]	Chaîne variable de n caractères ou octets.	Taille variable (maximum de 65 535 caractères).
BINARY(n)	Chaîne fixe de n octets.	Taille fixe (maximum de 255 octets).
VARBINARY(n)	Chaîne variable de n octets.	Taille variable (maximum de 255 octets).
TINYTEXT(n)	Flot de n octets.	Taille fixe (maximum de 255 octets).
TEXT(n)	Flot de n octets.	Taille fixe (maximum de 65 535 octets).
MEDIUMTEXT(n)	Flot de n octets.	Taille fixe (maximum de 16 mégaoctets).
LONGTEXT(n)	Flot de n octets.	Taille fixe (maximum de 4,29 gigaoctets).

Tableau 1-4 : Types de données caractères

4.1.8.2. Valeurs numériques

De nombreux types sont proposés par MySQL pour définir des valeurs exactes (entiers ou décimaux, positifs ou négatifs : **INTEGER** et **SMALLINT**) et des valeurs à virgule fixe ou flottante (**FLOAT**, **DOUBLE** et **DECIMAL**). En plus des spécifications de la norme SQL, MySQL propose les types d'entiers restreints (**TINYINT**, **MEDIUMINT**, **BIGINT**). Le tableau suivant décrit ces types :

- ✚ n indique le nombre de positions de la valeur à l'affichage (le maximum est de 255). Ainsi, il est possible de déclarer une colonne **TINYINT(4)** sachant que seules 3 positions sont nécessaires en fonction du domaine de valeurs permises.
- ✚ La directive **UNSIGNED** permet de considérer seulement des valeurs positives.
- ✚ La directive **ZEROFILL** complète par des zéros à gauche une valeur (par exemple : soit un **INTEGER(5)** contenant 4, si **ZEROFILL** est appliqué, la valeur extraite sera « 00004 »). En déclarant une colonne **ZEROFILL**, MySQL l'affecte automatiquement aussi à **UNSIGNED**.

Synonymes et alias

- ✚ **INT** est synonyme de **INTEGER**.

- ✚ *DOUBLE PRECISION et REAL sont synonymes de DOUBLE.*
- ✚ *DEC NUMERIC et FIXED sont synonymes de DECIMAL.*
- ✚ *SERIAL est un alias pour BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.*
- ✚ *Dans toute instruction SQL, écrivez la virgule par un point (7/2 retourne 3.5).*

Type	Description
BIT[(n)]	Ensemble de bits. Taille de 1 à 64 (par défaut 1).
TINYINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur un octet) de -128 à 127 signé, 0 à 255 non signé.
BOOL et BOOLEAN	Synonymes de TINYINT(1), la valeur zéro est considérée comme fausse. Le non-zéro est considéré comme vrai. Dans les prochaines versions, le type <i>boolean</i> comme le préconise la norme SQL sera réellement pris en charge.
SMALLINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 2 octets) de -32.768 à 32.767 signé, 0 à 65.535 non signé.
MEDIUMINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 3 octets) de -8.388.608 à 8.388.607 signé, 0 à 16.777.215 non signé.
INTEGER[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 4 octets) de -2.147.483.648 à 2.147.483.647 signé, 0 à 4.294.967.295 non signé.
BIGINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 8 octets) de -9.223.372.036.854.775.808 à 9.223.372.036.854.775.807 signé, 0 à 18.446.744.073.709.551.615 non signé.
FLOAT[(n,p)] [UNSIGNED] [ZEROFILL]	Flottant (de 4 à 8 octets) <i>p</i> désigne la précision simple (jusqu'à 15 décimales) de $-3,4 \cdot 10^{+38}$ à $-1,1 \cdot 10^{-38}$, 0 signé, et de $1,1 \cdot 10^{-38}$ à $3,4 \cdot 10^{+38}$ non signé.
DOUBLE[(n ,p)] [UNSIGNED] [ZEROFILL]	Flottant (sur 8 octets) <i>p</i> désigne la précision double jusqu'à 15 décimales) de $-1,7 \cdot 10^{-308}$ à $-2,2 \cdot 10^{-308}$, 0 signé, et de $2,2 \cdot 10^{-308}$ à $1,7 \cdot 10^{+308}$ non signé.
DECIMAL[(n,p)] [UNSIGNED] [ZEROFILL]	Décimal (sur 4 octets) à virgule fixe, <i>p</i> désigne la précision (nombre de chiffres après la virgule, maximum 30). Par défaut <i>n</i> vaut 10, <i>p</i> vaut 0. Nombre maximal de chiffres pour un nombre décimal est : 65.

Tableau 1-5 : Types de données numériques

4.1.8.3. Dates et heures

Les types suivants permettent de stocker des moments ponctuels (dates, dates et heures, années, heures). Les fonctions *NOW()* et *SYSDATE()* retournent la date et l'heure courantes. Dans une procédure ou un déclencheur *SYSDATE* est réévaluée en temps réel, alors que *NOW()* désignera toujours l'instant de début de traitement.

Type	Description	Commentaire pour une colonne
DATE	Dates du 1er janvier de l'an 1000 au 31 décembre 9999 après J.-C.	Sur 3 octets. L'affichage est au format 'YYYY-MM-DD'.
DATETIME	Dates et heures (de 0 h de la première date à 23h59 minutes 59 secondes de la dernière date).	Sur 8 octets. L'affichage est au format 'YYYY-MM-DD HH:MM:SS'.
YEAR [(2 4)]	Sur 4 positions, de 1901 à 2155 (incluant 0000). Sur 2 positions, de 70 à 69 (désignant 1970 à 2069).	Sur 1 octet, l'année est considérée sur 2 ou 4 positions (4 par défaut). Le format d'affichage est 'YYYY'.
TIME	Heures de -838 h 59 minutes 59 secondes à 838 h 59 minutes 59 secondes.	L'heure au format 'HHH: MM: SS' sur 3 octets.
TIMESTAMP	Instants du 1er Janvier 1970 0 h 0 minute 0 seconde à l'année 2037.	Estampille sur 4 octets (au format 'YYYY - MM-DD HH: MM: SS'), mise à jour à chaque modification sur la table.

Tableau 1-6 : Types de données dates et heures

4.1.8.4. Données binaires

Les types *BLOB* (*Binary Large Object*) permettent de stocker des données non structurées comme le multimédia (images, sons, vidéo, etc.). Les quatre types de colonnes *BLOB* sont *TINYBLOB*, *BLOB*, *MEDIUMBLOB* et *LOBLOB*. Ces types sont traités comme des flots d'octets sans jeu de caractères associé.

Type	Description	Commentaire pour une colonne
TINYBLOB(n)	Flot de <i>n</i> octets.	Taille fixe (maximum de 255 octets).
BLOB(n)		Taille fixe (maximum de 65 535 octets).
MEDIUMBLOB(n)		Taille fixe (maximum de 16 mégaoctets).
LOBLOB(n)		Taille fixe (maximum de 4,29 gigaoctets).

Tableau 1-7 : Types de données binaires

4.1.8.5. Énumération

Deux types de collections sont proposés par MySQL.

- Le type ENUM définit une liste de valeurs permises (chaînes de caractères).
- Le type SET permettra de comparer une liste à une combinaison de valeurs permises à partir d'un ensemble de référence (chaînes de caractères).

Type	Description
ENUM('valeur1', 'valeur2', ...)	Liste de 65.535 valeurs au maximum.
SET('valeur1', 'valeur2', ...)	Ensemble de référence (maximum de 64 valeurs).

4.1.9. Structure d'une table (DESCRIBE)

DESCRIBE (écriture autorisée DESC) est une commande qui vient de SQL*Plus d'Oracle et qui a été reprise par MySQL. Elle permet d'extraire la structure brute d'une table ou d'une vue.

DESCRIBE [nomBase .] nomTableouVue [colonne] ;

Si la base n'est pas indiquée, il s'agit de celle en cours d'utilisation. Retrouvons la structure des tables Compagnie et Pilote précédemment créées. Le type de chaque colonne apparaît :

Résultat

```
mysql> describe Pilote;
```

Field	Type	Null	Key	Default	Extra
brevet	char(6)	NO	PRI		
nom	char(15)	NO		NULL	
nbHVol	decimal(7,2)	YES		NULL	
compa	char(4)	YES	MUL	NULL	


```
mysql> describe Compagnie;
```

Field	Type	Null	Key	Default	Extra
comp	char(4)	NO	PRI		
nrue	int(3)	YES		NULL	
rue	char(20)	YES		NULL	
ville	char(15)	YES		Paris	
nonComp	char(15)	NO		NULL	

Commentaires

Les clés primaires sont NOT NULL (désignées par **PRI** dans la colonne Key).

Les unicités sont désignées par **UNI** dans la colonne Key.

Les occurrences multiples possibles sont désignées par **MUL** dans la colonne key.

Les contraintes NOT NULL nommées (définies via les contraintes CHECK) n'apparaissent pas.

La colonne *Extra* indique notamment les séquences (AUTO_INCREMENT).

Tableau 1-9 : Structure brute des tables

Restrictions

- ➔ Les contraintes CHECK définies ne sont pas encore opérationnelles.
- ➔ La fraction de seconde du type TIME n'est pas encore prise en charge 'D HH:MM:SS.fraction'.
- ➔ Les noms des colonnes doivent être uniques pour une table donnée (il est en revanche possible d'utiliser le même nom de colonne dans plusieurs tables).
- ➔ Les colonnes de type SET sont évaluées par des chaînes de caractères séparés par des « , » ('Airbus, Boeing'). En conséquence aucune valeur d'un SET ne doit contenir le symbole « , ».
- ➔ Les noms des objets (base, tables, colonnes, contraintes, vues, etc.) ne doivent pas emprunter des mots-clés de MySQL : TABLE, SELECT, INSERT, IF ...

4.2. Index

Comme l'index d'un ouvrage vous aide à atteindre les pages concernées par un mot recherché, un index MySQL permet d'accélérer l'accès aux données d'une table. Le but principal d'un index est d'éviter de parcourir une table séquentiellement du premier enregistrement jusqu'à celui visé. Le principe d'un index est l'association de l'adresse de chaque enregistrement avec la valeur des colonnes indexées.

Sans index et pour n enregistrements, le nombre moyen d'accès nécessaire pour trouver un élément est égal à $n/2$. Avec un index, ce nombre tendra vers $\log(n)$ et augmentera donc bien plus faiblement en fonction de la montée en charge des enregistrements. Si une table contient 1.000 enregistrements, alors l'usage d'un index accélérera l'accès d'un facteur 100 par rapport à un accès séquentiel.

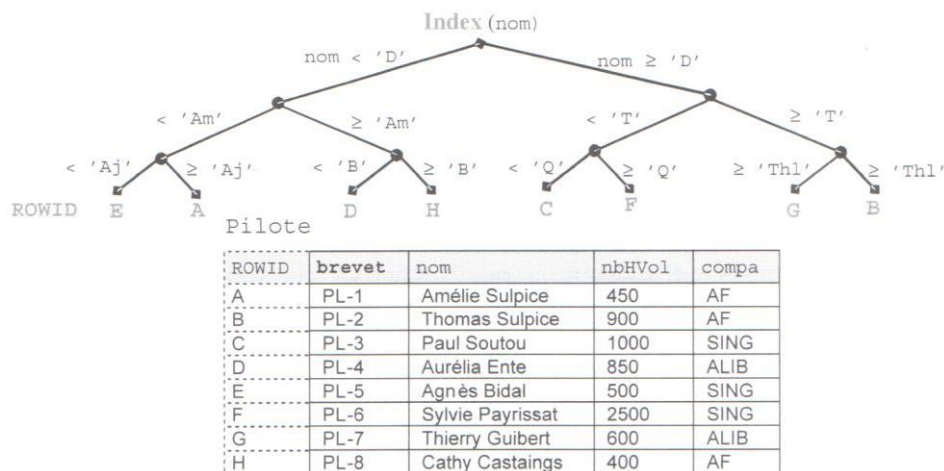
4.2.1. Arbres balancés

La figure suivante illustre un index sous la forme d'un arbre. Cet index est basé sur la colonne nom de la table Pilote. Cette figure est caricaturale car un index n'est pas un arbre binaire (plus de deux liens peuvent partir d'un nœud). Dans cet exemple, trois accès à l'index seront nécessaires pour adresser directement un pilote via son nom au lieu d'en analyser huit au plus.

Un index est associé à une table et peut être défini sur une ou plusieurs colonnes (dites « indexées »). Une table peut « héberger » plusieurs index. Ils sont mis à jour automatiquement après rafraîchissement de la table (ajouts et suppressions d'enregistrements ou modification des colonnes indexées). Un index peut être déclaré unique si on sait que les valeurs des colonnes indexées seront toujours uniques.

La plupart des index de MySQL (PRIMARY KEY, UNIQUE, INDEX et FULLTEXT) sont stockés dans des arbres équilibrés (*balanced trees* : *B-trees*). D'autres types d'index existent, citons ceux qui portent sur des colonnes SPATIAL (*reverse key* : *R-trees*) et ceux appliqués aux tables MEMORY (tables de hachage : *hash*).

La particularité des index *B-tree* est qu'ils conservent en permanence une arborescence symétrique (balancée). Toutes les feuilles sont à la même profondeur. Le temps de recherche est ainsi à peu près constant quel que soit l'enregistrement cherché. Le plus bas niveau de l'index (*leaf blocks*) contient les valeurs des colonnes indexées et le *rowid*. Toutes les feuilles de l'index sont chaînées entre elles. Pour les index non uniques (par exemple si on voulait définir un index sur la colonne compa de la table Pilote), le *rowid* est inclus dans la valeur de la colonne indexée. Ces index, premiers apparus, sont désormais très fiables et performants, ils ne se dégradent pas lors de la montée en charge de la table.



4.2.2. Création d'un index (CREATE INDEX)

Pour pouvoir créer un index dans sa base, la table à indexer doit appartenir à la base. Si l'utilisateur a le privilège INDEX, il peut créer et supprimer des index dans sa base. Un index est créé par l'instruction CREATE INDEX et supprimé par DROP INDEX.

La syntaxe de création d'un index est la suivante :

CREATE [UNIQUE | FULLTEXT | SPATIAL] **INDEX** nomIndex
 [USING BTREE | HASH]
 ON nomTable (colonne1 [(taille1)]) (ASC | DESC), ...)

- ➡ UNIQUE permet de créer un index qui n'accepte pas les doublons.
- ➡ FULLTEXT permet de bénéficier de fonctions de recherche dans des textes (flot de caractères).
- ➡ SPATIAL permet de profiter de fonctions pour les données géographiques.
- ➡ ASC et DESC précisent l'ordre (croissant ou décroissant).

Créons deux index sur la table pilote.

Instruction SQL	Commentaires
<pre>1 CREATE UNIQUE INDEX idx_Pilote_nom3 2 USING BTREE 3 ON Pilote (nom(3) DESC);</pre>	Index <i>B-tree</i> , ordre décroissant sur les trois premiers caractères du nom des pilotes.
<pre>1 CREATE INDEX idx_Pilote_compa 2 USING BTREE 3 ON Pilote (compa);</pre>	Index <i>B-tree</i> , ordre croissant sur la colonne clé étrangère compa.

Remarques

- ➡ Un index ralentit les rafraîchissements de la base (conséquence de la mise à jour de l'arbre ou des *bitmaps*). En revanche, il accélère les accès.

- ➡ Il est conseillé de créer des index sur des colonnes (majoritairement des clés étrangères) utilisées dans les clauses de jointures.
- ➡ Il est possible de créer des index pour toutes les colonnes d'une table (jusqu'à concurrence de 16).
- ➡ Les index sont pénalisants lorsqu'ils sont définis sur des colonnes très souvent modifiées ou si la table contient peu de lignes.

4.3. Destruction d'un schéma

Il vous sera utile d'écrire un script de destruction d'un schéma (j'entends « schéma » comme ensemble de tables, contraintes et index composant une base de données et non pas en tant qu'ensemble de tous les objets d'un utilisateur) pour pouvoir recréer une base propre. Bien entendu, si des données sont déjà présentes dans les tables et que vous souhaitez les garder, il faudra utiliser une stratégie pour les réimporter dans les nouvelles tables. A ce niveau du cours vous n'en êtes pas là, et le script de destruction va vous permettre de corriger vos erreurs de syntaxe du script de création des tables.

Nous avons vu qu'il fallait créer d'abord les tables « pères » puis les tables « fils » (si des contraintes sont définies en même temps que les tables). L'ordre de destruction des tables pour des raisons de cohérence est l'inverse (il faut détruire les tables « fils » puis les tables « pères »). Dans notre exemple, il serait malvenu de supprimer la table Compagnie avant la table Pilote. En effet, la clé étrangère *compa* n'aurait plus de sens.

4.3.1. Suppression d'une table (DROP TABLE)

Pour pouvoir supprimer une table dans une base, il faut posséder le privilège DROP sur cette base. L'instruction DROP TABLE entraîne la suppression des données, de la structure, de la description dans le dictionnaire des données, des index, des déclencheurs associés (*triggers*) et la récupération de la place dans l'espace de stockage.

```
DROP [TEMPORARY] TABLE [IF EXISTS]
[nomBase.] nomTable1 [, [nomBase2.]nomTable2,...]
[RESTRICT | CASCADE]
```

- ✚ TEMPORARY : pour supprimer des tables temporaires. Les transactions en cours ne sont pas affectées. L'utilisation de TEMPORARY peut être un bon moyen de s'assurer qu'on ne détruit pas accidentellement une table non temporaire...
- ✚ IF EXISTS : permet d'éviter qu'une erreur se produise si la table n'existe pas.
- ✚ RESTRICT (par défaut) : permet de vérifier qu'aucun autre élément n'utilise la table (autre table via une clé étrangère, vue, déclencheur, etc.).
- ✚ CASCADE : option qui n'est pas encore opérationnelle, doit répercuter la destruction à toutes les autres tables référencées par des clés étrangères.

Les éléments qui utilisaient la table (vues, synonymes, fonctions et procédures) ne sont pas supprimés mais sont temporairement inopérants. Attention, une suppression ne peut pas être par la suite annulée.

4.3.2. Ordre des suppressions

Remarque : Il suffit de relire à l'envers le script de création de vos tables pour en déduire l'ordre de suppression à écrire dans le script de destruction de votre schéma.

Le tableau suivant présente deux écritures possibles pour détruire le schéma contenant les tables *Pilote* et *Compagnie* et deux index. La première écriture ne se soucie pas de l'ordre des tables (il faut alors supprimer les clés étrangères, les index, puis les tables dans un ordre cohérent). La seconde écriture suit l'ordre des tables enfants puis parents.

Sans ordre des tables	En respectant l'ordre des tables
<pre>1 DROP INDEX idx_Pilote_nom3 ON Pilote; 2 ALTER TABLE Pilote 3 DROP FOREIGN KEY fk_Pil_compa_Comp; 4 DROP INDEX idx_Pilote_compa ON Pilote; 5 DROP TABLE Compagnie; 6 DROP TABLE Pilote;</pre>	<pre>1 DROP TABLE Pilote; 2 DROP Table Compagnie;</pre>

Tableau 1-11 : Scripts de destruction

4.4. Exercices d'application.

L'objectif de ces exercices est de créer des tables, leur clé primaire et des contraintes de vérification (NOT NULL et CHECK).

Exercice 1

Créez la base de données « Téléphonie ». Créez la table « Carte_Appel » avec les attributs et hypothèses indiquées ci-après. Choisissez les types de données les plus appropriés.

Attribut : *Nom_Societe*, *Numero_Carte*, *Valeur_Depart*, *Valeur_Restante* et *Numero-Pin*

Hypothèse : l'attribut *Nom_Societe* doit pouvoir comporter jusqu'à 25 caractères. Les attributs *Valeur_Depart* et *Valeur_Restante* sont exprimées en Euro. L'attribut *Numero_Carte* peut comporter jusqu'à 15 caractères. L'attribut *Numero-Pin* comporte toujours 12 caractères.

Exercice 2

Réécrivez l'instruction CREATE TABLE de l'exemple précédent avec l'attribut *Numero_Carte* défini comme clé primaire et l'attribut *Numero_Pin* défini comme étant unique.

Exercice 3

Réécrivez la commande CREATE TABLE de l'exemple précédent au moyen de contraintes nommées.

Exercice 4

Décrivez la structure de la table *Carte_Appel*.

Exercice 5 : Présentation de la base de données

Une entreprise désire gérer son parc informatique à l'aide d'une base de données. Le bâtiment est composé de trois étages. Chaque étage possède son réseau (ou segment distinct) Ethernet. Ces réseaux traversent des salles équipées de postes de travail. Un poste de travail est une machine sur laquelle sont installés certains logiciels. Quatre catégories de postes de travail sont recensées (stations Unix, terminaux X, PC Windows et PC NT). La base de données devra aussi décrire les installations de logiciels.

Les noms et types des colonnes sont les suivants :

Colonne	Commentaire	Type
indIP	trois premiers groupes IP (exemple : 130.120.80)	VARCHAR(11)
nomSegment	nom du segment	VARCHAR(20)
etage	étage du segment	TINYINT(1)
nSalle	numéro de la salle	VARCHAR(7)
nomSalle	nom de la salle	VARCHAR(20)
nbPoste	nombre de postes de travail dans la salle	TINYINT(2)
nPoste	code du poste de travail	VARCHAR(7)
nomPoste	nom du poste de travail	VARCHAR(20)
ad	dernier groupe de chiffres IP (exemple : 11)	VARCHAR(3)
typePoste	type du poste (UNIX, TX, PCWS, PCNT)	VARCHAR(9)
dateIns	date d'installation du logiciel sur le poste	dateTime
nLog	code du logiciel	VARCHAR(5)
nomLog	nom du logiciel	VARCHAR(20)
dateAch	date d'achat du logiciel	dateTime
version	version du logiciel	VARCHAR(7)
typeLog	type du logiciel (UNIX, TX, PCWS, PCNT)	VARCHAR(9)
prix	prix du logiciel	DECIMAL (6, 2)
numIns	numéro séquentiel des installations	INTEGER(5)
dateIns	date d'installation du logiciel	TIMESTAMP
delai	intervalle entre achat et installation	SMALLINT
typeLP	types des logiciels et des postes	VARCHAR(9)
nomType	noms des types (Terminaux X, PC Windows...)	VARCHAR(20)

Tableau 1-12 : Caractéristiques des colonnes

Exercice 6 : Création des tables

Ecrivez puis exécutez le script SQL (que vous appellerez creParc.sql) de création des tables avec leurs clés primaires (en gras dans le schéma suivant) et les contraintes suivantes :

- Les noms des segments, des salles et des postes sont non nuls.
- Le domaine de valeurs de la colonne **ad** s'étend de 0 à 255.
- La colonne **prix** est supérieure ou égale à 0.
- La colonne **dateIns** est égale à la date du jour par défaut.

Segment

indIP	nomSegment	etage
--------------	------------	-------

Salle

nSalle	nomSalle	nbPoste	indIP
---------------	----------	---------	-------

Poste

nPoste	nomPoste	indIP	ad	typePoste	nSalle
---------------	----------	-------	----	-----------	--------

Logiciel

nLog	nomLog	dateAch	version	typeLog	prix
-------------	--------	---------	---------	---------	------

Installer

nPoste	nLog	numIns	dateIns	delai
--------	------	---------------	---------	-------

Types

typeLP	nomType
---------------	---------

Figure 1-4 : Composition des tables**Exercice 7 : Structure des tables**

Ecrivez puis exécutez le script SQL (que vous appellerez descParc.sql) qui affiche la description de toutes ces tables (en utilisant des commandes DESCRIBE). Comparez le résultat obtenu avec le schéma ci-dessus.

Exercice 8 : Destruction des tables

Ecrivez puis exécutez le script SQL de destruction des tables (que vous appellerez dropParc.sql). Lancez ce script puis celui de la création des tables à nouveau.