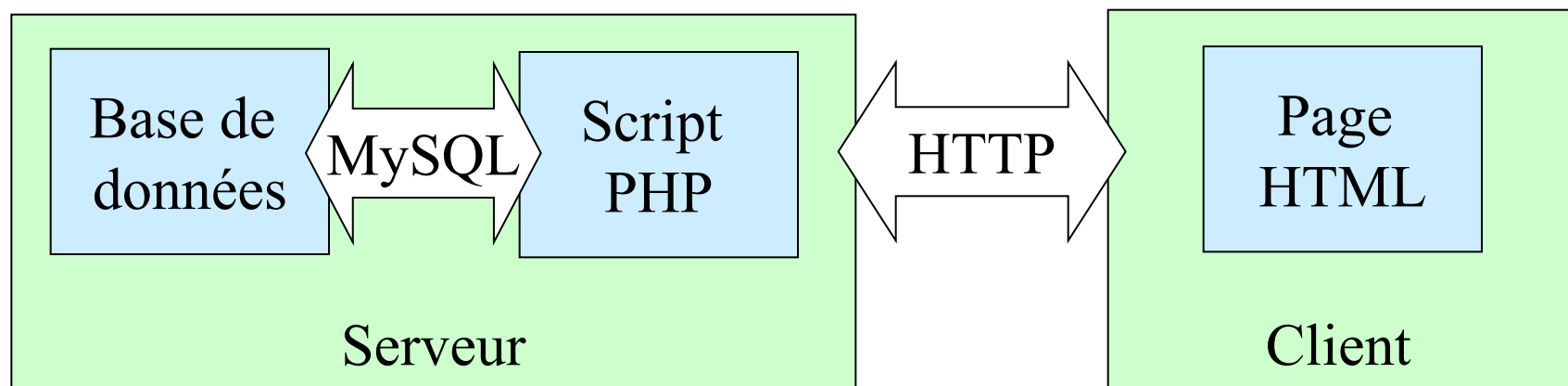


# ***Introduction aux SGBD Relationnels avec MySQL***

# Introduction

MySQL dérive directement de SQL (Structured Query Language) qui est un langage de requête vers les bases de données exploitant le modèle relationnel.

Le serveur de base de données MySQL est très souvent utilisé avec le langage de création de pages web dynamiques : PHP. Il sera discuté ici des commandes MySQL utilisables via PHP dans les conditions typiques d'utilisation dans le cadre de la gestion de sites Internet/Intranet.



# Sommaire

---

1. Théorie des bases de données relationnelles
2. Syntaxe de MySQL
3. Fonctions de MySQL
4. Administration avec l'outil phpMyAdmin

# 1

# Théorie des bases de données

# Algèbre relationnelle

L'algèbre relationnelle regroupe toutes les opérations possibles sur les tables. Voici la liste des opérations possibles :

Projection : on ne sélectionne qu'un ou plusieurs attributs d'une table (on ignore les autres). Par exemple n'afficher que les colonnes *nom* et *prénom* de la table **Personnes**.

Jointure : on fabrique une nouvelle table à partir de 2 ou plusieurs autres en prenant comme pivot 1 ou plusieurs attributs. Par exemple, on concatène la table du carnet d'adresse et celle des inscrits à la bibliothèque en fonction du nom de famille (c'est typiquement du recoupement de fichiers).

Sélection : on sélectionne tous les tuples ou bien seulement une partie en fonction de critères de sélection qui portent sur les valeurs des attributs. Par exemple n'afficher que les lignes de la table **Personnes** qui vérifient la condition suivante : le nom ne commence pas par la lettre 'C'.

Cette algèbre est facilement possible avec les commandes de MySQL (SELECT... FROM... WHERE...).


# Projection

**Personnes**

<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
Martin	Pierre	7 allée des vers	0258941236
Dupond	Jean	32 allé Poivrot	0526389152
Dupond	Marc	8 rue de l'octet	0123456789

On projette la table **Personnes** sur les colonnes *nom* et *prénom*.

  
**SELECT** *nom, prénom*  
**FROM** **Personnes**



<i>nom</i>	<i>prénom</i>
Martin	Pierre
Dupond	Jean
Dupond	Marc

⇒ Le caractère “\*” Permet de sélectionner toutes les colonnes

# Jointure

**Personnes**

<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
Martin	Pierre	7 allée des vers	0258941236
Dupond	Jean	32 allé Poivrot	0526389152

**Bibliothèque**

<i>nom</i>	<i>Dernierlivre</i>
Dupond	Robinson
Jospin	Faust
Martin	Misère

**SELECT** *Personnes.prénom*, *dernierlivre*  
**FROM** *Personnes*, *Bibliothèque*  
**WHERE** *Personnes.nom* =  
*Bibliothèque.nom*

<i>prénom</i>	<i>Dernierlivre</i>
Jean	Robinson
Pierre	Misère

On joint les deux tables, grâce à la colonne *nom*.

Et on combine cette jointure à une projection sur les attributs *nom* et *dernierlivre*.

Attention à lever toute ambiguïté sur les noms d'attribut dans le cas où deux tables possèdent des colonnes de même nom.

# Sélection

**Personnes**

<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
Martin	Pierre	7 allée des vers	0258941236
Dupond	Jean	32 allé Poivrot	0526389152
Dupond	Marc	8 rue de l'octet	0123456789



```

SELECT *
FROM Personnes
WHERE nom = "Dupond"

```



<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
Dupond	Jean	32 allé Poivrot	0526389152
Dupond	Marc	8 rue de l'octet	0123456789

On ne sélectionne que les tuples dont l'attribut *nom* est égale à 'Dupond'.



# 2

## Syntaxe de MySQL

# Types des attributs (I)

Les propriétés de vos objets peuvent être de types très différents :

- Nombre entier signé ou non (température, quantité commandée, âge)
- Nombre à virgule (prix, taille)
- Chaîne de caractères (nom, adresse, article de presse)
- Date et heure (date de naissance, heure de parution)

Il s'agit de choisir le plus adapté à vos besoins.

Ces types requièrent une plus ou moins grande quantité de données à stocker. Par exemple, ne pas choisir un LONGTEXT pour stocker un prénom mais plutôt un VARCHAR(40) !

# Types des attributs (II) – entiers

nom	borne inférieure	borne supérieure
TINYINT	-128	127
TINYINT UNSIGNED	0	255
SMALLINT	-32768	32767
SMALLINT UNSIGNED	0	65535
MEDIUMINT	-8388608	8388607
MEDIUMINT UNSIGNED	0	16777215
INT*	-2147483648	2147483647
INT* UNSIGNED	0	4294967295
BIGINT	-9223372036854775808	9223372036854775807
BIGINT UNSIGNED	0	18446744073709551615

(\*) : **INTEGER** est un synonyme de **INT**.

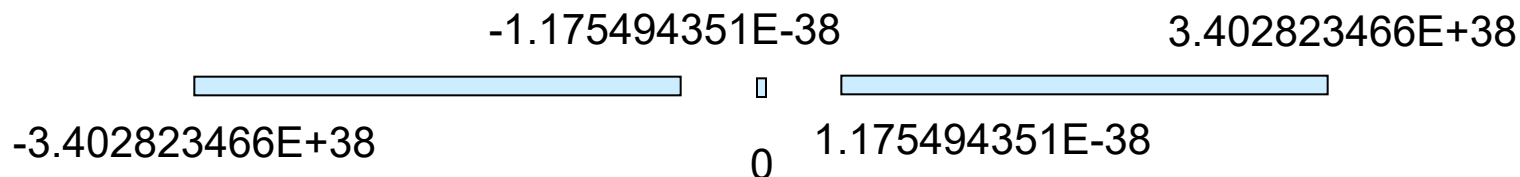
**UNSIGNED** permet d'avoir un type non signé.

**ZEROFILL** : remplissage des zéros non significatifs.

# Types des attributs (III) – flottants

Les flottants – dits aussi nombres réels – sont des nombres à virgule. Contrairement aux entiers, leur domaine n'est pas continu du fait de l'impossibilité de les représenter avec une précision absolue.

Exemple du type **FLOAT** :



nom	domaine négatif : borne inférieure borne supérieure	Domaine positif : borne inférieure borne supérieure
<b>FLOAT</b>	$-3.402823466E+38$ $-1.175494351E-38$	$1.175494351E-38$ $3.402823466E+38$
<b>DOUBLE*</b>	$-1.7976931348623157E+308$ $-2.2250738585072014E-308$	$2.2250738585072014E-308$ $1.7976931348623157E+308$

(\*) : **REAL** est un synonyme de **DOUBLE**.

# Types des attributs (IV) – chaînes

nom	longueur
<b>CHAR(M)</b>	Chaîne de taille fixée à M, où $1 < M < 255$ , complétée avec des espaces si nécessaire.
<b>CHAR(M) BINARY</b>	Idem, mais insensible à la casse lors des tris et recherches.
<b>VARCHAR(M)</b>	Chaîne de taille variable, de taille maximum M, où $1 < M < 255$ , complété avec des espaces si nécessaire.
<b>VARCHAR(M) BINARY</b>	Idem, mais insensible à la casse lors des tris et recherches.
<b>TINYTEXT</b>	Longueur maximale de 255 caractères.
<b>TEXT</b>	Longueur maximale de 65535 caractères.
<b>MEDIUMTEXT</b>	Longueur maximale de 16777215 caractères.
<b>LONGTEXT</b>	Longueur maximale de 4294967295 caractères.
<b>DECIMAL(M,D)*</b>	Simule un nombre flottant de D chiffres après la virgule et de M chiffres au maximum. Chaque chiffre ainsi que la virgule et le signe moins (pas le plus) occupe un caractère.

(\*) : **NUMERIC** est un synonyme de **DECIMAL**.

# Types des attributs (V) – chaînes

Les types TINYTEXT, TEXT, MEDIUMTEXT et LONGTEXT peuvent être judicieusement remplacés respectivement par TINYBLOB, BLOB, MEDIUMBLOB et LONGBLOB.

Ils ne diffèrent que par la sensibilité à la casse qui caractérise la famille des BLOB. Alors que la famille des TEXT sont insensibles à la casse lors des tris et recherches.

Les BLOB peuvent être utilisés pour stocker des données binaires.

Les VARCHAR, TEXT et BLOB sont de taille variable. Alors que les CHAR et DECIMAL sont de taille fixe.

# Types des attributs (VI) – dates et heures

nom	description
<b>DATE</b>	Date au format anglophone AAAA-MM-JJ.
<b>DATETIME</b>	Date et heure au format anglophone AAAA-MM-JJ HH:MM:SS.
<b>TIMESTAMP</b>	Affiche la date et l'heure sans séparateur : AAAAMMMJJHHMMSS.
<b>TIMESTAMP(M)</b>	Idem mais M vaut un entier pair entre 2 et 14. Affiche les M premiers caractères de <b>TIMESTAMP</b> .
<b>TIME</b>	Heure au format HH:MM:SS.
<b>YEAR</b>	Année au format AAAA.

# Identificateurs

Les noms des bases, tables, attributs, index et alias sont constitués de caractères alphanumériques et des caractères `_` et `$`.

Un nom comporte au maximum **64** caractères.

Comme les bases de données et les tables sont codées directement dans le système de fichiers, la sensibilité à la casse de MySQL dépend de celle du système d'exploitation sur lequel il repose. Sous **Windows**, la casse n'a pas d'importance ; alors que sous **Unix**, elle en a !

Le point `.` est un caractère réservé utilisé comme séparateur entre le nom d'une base et celui d'une table, entre le nom d'une table et celui d'un attribut.

Exemple :

```
SELECT base1.table25.attribut5  
FROM base1.table25
```



# Exemple

Imaginons que l'on veuille construire la version web d'un journal papier. Nous devons créer une table pour stocker les articles de presse. Les informations relatives à un article sont les suivantes : titre, texte, date de parution, auteur, rubrique.

Un titre ayant une longueur raisonnable, il sera de type VARCHAR(80), le texte pourra être très grand : TEXT (65535 caractères !), la date sera au format DATE (YYYY:MM:JJ). L'auteur pourra être codé sur un VARCHAR(80). Et la rubrique pourrait être un ENUM.

```
CREATE TABLE article (  
  id MEDIUM INT UNSIGNED PRIMARY KEY,  
  titre VARCHAR(80),  
  texte TEXT,  
  parution DATE,  
  auteur VARCHAR(80),  
  rubrique ENUM('économie','sports','international','politique','culture')  
)
```

# Créer une table (I)

La création d'une table utilise la commande **CREATE TABLE** selon la syntaxe suivante :

```
CREATE [TEMPORARY] TABLE nom_table [IF NOT EXISTS] (  
    nom_attribut TYPE_ATTRIBUT [OPTIONS]  
    ...  
)
```

TEMPORARY donne pour durée de vie à la table : le temps de la connexion de l'utilisateur au serveur, après, elle sera détruite. En l'absence de cette option, la table sera permanente à moins d'être détruite par la commande DROP TABLE. L'option IF NOT EXIST permet de ne créer cette table que si une table de même nom n'existe pas encore.

A l'intérieur des parenthèses, il sera listé tous les attributs, clés et indexs de la table.

## Créer une table (II)

Le type de l'attribut doit être d'un type vu précédemment.  
Les options seront vues au fur et à mesure du cours.

Exemple du carnet d'adresse :

```
CREATE TABLE Personne (  
    nom VARCHAR(40),  
    'prénom' VARCHAR(40),  
    adresse TINYTEXT,  
    'téléphone' DECIMAL(10,0)  
)
```

Notre carnet d'adresse est stocké dans un tableau (appelé **table**) de nom *Personne* qui comporte les colonnes (dites aussi **attributs**) suivantes : *nom* (chaîne de 40 caractères maximum), *prénom* (idem), *adresse* (texte de longueur variable mais inférieure à 255 caractères) et *téléphone* (chaîne de 10 caractères). Chacune des personnes à ajouter au carnet d'adresse occupera une ligne de cette table. Une ligne est dite **enregistrement** dans le jargon des bases de données.

# Clé primaire (I)

Pour des raisons pratiques, si nous souhaitons pouvoir associer à chacun des enregistrements de la table un identifiant numérique unique qui puisse être passé en paramètre à nos scripts PHP.

Pour cela on rajoute un nouvelle attribut de type entier. Pour nous facilité la tâche, cet entier ne devra pas être signé mais être suffisamment grand pour identifier tous nos enregistrements car destiné à un décompte (donc débute forcément à 1 et pas à -127 par exemple).

Dans notre exemple, le carnet d'adresse ne devrait pas excéder plusieurs centaines de personnes. Ainsi un attribut de type **SMALLINT UNSIGNED** devrait faire l'affaire. Nous le nommerons par la suite : *id*.

Cet attribut devra ne jamais être vide, il faut donc préciser l'option **NOT NULL** pour le forcer à prendre une valeur de son domaine (entre 0 et 65535).

Il devra aussi être unique, c'est-à-dire que deux enregistrements ne pourront pas avoir une valeur identique de *id*. Il faut alors faire la déclaration suivante : **UNIQUE (id)** à la suite de la liste des attributs.

Pour simplifier, on utilisera l'option **PRIMARY KEY** qui regroupe **NOT NULL** et **UNIQUE** en remplacement des deux dernières déclarations.

Et pour finir, il faut signifier que cette valeur doit s'incrémenter automatiquement à chaque insertion d'un enregistrement grâce à l'option **AUTO\_INCREMENT**.

## Clé primaire (II)

Notre exemple devient :

```
CREATE TABLE Personne (  
    id SMALLINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,  
    nom VARCHAR(40),  
    'prénom' VARCHAR(40),  
    adresse TINYTEXT,  
    'téléphone' DECIMAL(10,0)  
)
```

Cet identifiant numérique unique auto-incrémental, s'appelle une « *clé primaire* ».

La numérotation des clés primaires, débute à 1 et pas à 0.

### *Personnes*

<i>Id</i>	<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
1	Dupond	Marc	8 rue de l'octet	0123456789

## Clé primaire (III)

Notre clé primaire peut être associée simultanément à plusieurs attributs mais selon une syntaxe différente.

Si au lieu de créer un identifiant numérique unique, on souhaite simplement interdire d'avoir des doublon sur le couple (*nom*,*prénom*) et d'en interdire la nullité, on va créer une clé primaire sur ce couple.

La connaissance des seuls nom et prénom suffit à identifier sans ambiguïté un et un seul enregistrement.

Mauvaise syntaxe :

```
CREATE TABLE Personne (  
    nom VARCHAR(40) PRIMARY KEY,  
    'prénom' VARCHAR(40) PRIMARY KEY,  
    adresse TINYTEXT,  
    'téléphone' DECIMAL(10,0)  
)
```

Bonne syntaxe :

```
CREATE TABLE Personne (  
    nom VARCHAR(40),  
    'prénom' VARCHAR(40),  
    adresse TINYTEXT,  
    'téléphone' DECIMAL(10,0),  
    PRIMARY KEY (nom, 'prénom')  
)
```

# Attribut non nul

Considérons que l'on souhaite que certains attributs aient obligatoirement une valeur. On utilisera l'option **NOT NULL**.

Dans ce cas, si malgré tout, aucune valeur n'est fournie, la valeur par défaut – si elle est déclarée à la création de la table – sera automatiquement affectée à cet attribut dans l'enregistrement.

Si aucune valeur par défaut n'est déclarée :

- la chaîne vide "" sera affectée à l'attribut s'il est de type chaîne de caractères
- la valeur zéro 0 s'il est de type nombre
- la date nulle 0000-00-00 et/ou l'heure nulle 00:00:00 s'il est de type date, heure ou date et heure.

Exemple :

*adresse* **TINYTEXT NOT NULL**

Au contraire, on utilisera l'option **NULL** si on autorise l'absence de valeur.

# Valeur par défaut

Pour donner une valeur par défaut à un attribut, on utilise l'option **DEFAULT**.  
Lors de l'ajout d'un enregistrement cette valeur sera affectée à l'attribut si aucune valeur n'est donnée.

Exemple :

*'téléphone'* **DECIMAL(10,0) DEFAULT '0123456789'**

Les attributs de type chaîne de caractères de la famille TEXT et BLOB ne peuvent pas avoir de valeur par défaut.



# Attribut sans doublon (I)

Pour interdire l'apparition de doublon pour un attribut, on utilise l'option **UNIQUE**.

Syntaxe :

**UNIQUE** [*nom dela contrainte*](*liste des attributs*)

Exemple, pour interdire tout doublon de l'attribut *nom* :

**UNIQUE**(*nom*)

Pour interdire les doublons sur l'attribut *nom* mais les interdire aussi sur '*prénom*', tout en les laissant indépendants :

**UNIQUE**(*nom*)

**UNIQUE**('prénom')

<i>nom</i>	<i>prénom</i>
Dupond	<b>Marc</b>
Dupont	Pierre
Martin	<b>Marc</b>

enregistrement interdit  
car 'Marc' est un doublon  
dans la colonne 'prénom'

# Attribut sans doublon (II)

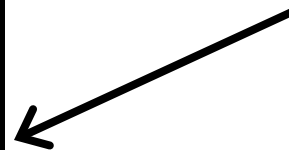
Pour interdire tout doublon à un ensemble d'attributs (tuple), on passe en paramètre à **UNIQUE** la liste des attributs concernés.

Pour interdire tout doublon du couple (*nom*, '*prénom*') :

**UNIQUE**(*nom*, '*prénom*')

<i>nom</i>	<i>prénom</i>
Dupond	Marc
Dupont	Pierre
<b>Martin</b>	<b>Marc</b>
Martin	Pierre
<b>Martin</b>	<b>Marc</b>

enregistrement interdit car le couple ('Martin', 'Marc') est un doublon du couple (nom, '*prénom*')



# Index (I)

Lors de la recherche d'informations dans une table, MySQL parcourt la table correspondante dans n'importe quel ordre. Dans le cas d'un grand nombre de lignes, cette **recherche** est très très longue du fait du parcours de **TOUTE** la table.

Pour y remédier, une **optimisation** possible et FORTEMENT recommandée, est d'utiliser des indexs.

La création d'un index associé à un attribut ou à un ensemble ordonné d'attributs va créer une **liste ordonnée** des valeurs de ces attributs et de l'adresse de la ligne associée. C'est sur les valeurs de cette liste que se fera **les recherches et les tris**. Les algorithmes de recherche et de tri sur des ensembles ordonnés sont énormément plus rapides !

Ainsi, d'une recherche à coût prohibitif, on passe à une recherche sur un ensemble déjà trié. On gagne donc énormément en temps d'accès aux informations. Bien que cela **ralentisse les mises à jour** (insertion, suppression, modification de clé).

On choisira de créer des indexs sur les **attributs** qui seront **les plus sollicités** par les recherches ou utilisés comme critère de jointure. Par contre, on épargnera les attributs qui contiennent peu de valeurs différentes les unes des autres et ceux dont les valeurs sont très **fréquemment modifiées**.

## Index (II)

Syntaxe :

**INDEX *index* (*liste des attributs*)**

Exemple, pour créer un index sur les 3 premiers caractères seulement de l'attribut *nom* :

**INDEX *idx\_nom* (*nom*(3))**

Exemple, pour créer un index sur le couple (*nom*, '*prénom*') :

**INDEX *idx\_nom\_prenom* (*nom*, '*prénom*')**

Un index peut porter sur 15 colonnes maximum.

Une table peut posséder au maximum 16 indexs.

Un index peut avoir une taille d'au maximum 256 octets et ne doit porter que sur des attributs NOT NULL.

Il suffit de suffixer l'attribut (CHAR, VARCHAR) pour dire de ne prendre que les M premiers caractères pour l'indexation.

# Supprimer une table

La commande **DROP TABLE** prend en paramètre le nom de la table à supprimer. Toutes les données qu'elle contient sont supprimées et sa définition aussi.

Syntaxe :

**DROP TABLE *table***

Exemple :

**DROP TABLE *Personnes***

Si un beau jour on s'aperçoit qu'une table a été mal définie au départ, plutôt que de la supprimer et de la reconstruire bien comme il faut, on peut la modifier très simplement. Cela évite de perdre les données qu'elle contient.

# Modifier une table

La création d'une table par **CREATE TABLE** n'en rend pas définitives les spécifications. Il est possible d'en modifier la définition par la suite, à tout moment par la commande **ALTER TABLE**.

Voici ce qu'il est possible de réaliser :

- ajouter/supprimer un attribut
- créer/supprimer une clé primaire
- ajouter une contrainte d'unicité (interdire les doublons)
- changer la valeur par défaut d'un attribut
- changer totalement la définition d'un attribut
- changer le nom de la table
- ajouter/supprimer un index

# Ajouter un attribut

Syntaxe :

```
ALTER TABLE table ADD definition [ FIRST | AFTER attribut]
```

Ajoutons l'attribut *fax* qui est une chaîne représentant un nombre de 10 chiffres:

```
ALTER TABLE Personnes ADD fax DECIMAL(10,0)
```

Nous aurions pu forcer la place où doit apparaître cet attribut. Pour le mettre en tête de la liste des attributs de la table, il faut ajouter l'option **FIRST** en fin de commande. Pour le mettre après l'attribut '*téléphone*', il aurait fallu ajouter **AFTER** '*téléphone*'.

Note : il ne doit pas déjà avoir dans la table un attribut du même nom !

# Supprimer un attribut (I)

Attention, supprimer un attribut implique la suppression des valeurs qui se trouvent dans la colonne qui correspond à cet attribut, sauf à utiliser l'option IGNORE.

Syntaxe :

**ALTER TABLE *table* DROP *attribut***

Exemple :

**ALTER TABLE *Personnes* DROP 'prénom'**



# Supprimer un attribut (II)

La suppression d'un attribut peut provoquer des erreurs sur les contraintes clé primaire (**PRIMARY KEY**) et unique (**UNIQUE**).

```
CREATE TABLE Personne (  
    id SMALLINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,  
    nom VARCHAR(40),  
    'prénom' VARCHAR(40),  
    adresse TINYTEXT,  
    'téléphone' DECIMAL(10,0),  
    UNIQUE(nom, 'prénom')  
)
```

ALTER TABLE *Personnes* DROP '*prénom*'

<i>nom</i>	<i>prénom</i>
Dupond	Marc
Martin	Marc
Martin	Pierre

Refus d'opérer la suppression, car cela contredirait la contrainte d'unicité qui resterait sur l'attribut *nom*.

<i>Nom</i>
Dupond
<b>Martin</b>
<b>Martin</b>

# Créer une clé primaire

La création d'une clé primaire n'est possible qu'en l'absence de clé primaire dans la table.

Syntaxe :

**ALTER TABLE *table* ADD PRIMARY KEY (*attribut*)**

Exemple :

**ALTER TABLE *Personnes* ADD PRIMARY KEY (*nom*, '*prénom*')**

# Supprimer une clé primaire

Comme une clé primaire est unique, il n'y a aucune ambiguïté lors de la suppression.

Syntaxe :

**ALTER TABLE *table* DROP PRIMARY KEY**

Exemple :

**ALTER TABLE *Personnes* DROP PRIMARY KEY**

S'il n'y a aucune clé primaire lorsque cette commande est exécutée, aucun message d'erreur ne sera généré, la commande sera simplement ignorée.

# Ajout d'une contrainte d'unicité

Il est possible (facultatif) de donner un nom à la contrainte.

Cette contrainte peut s'appliquer à plusieurs attributs.

Si les valeurs déjà présentes dans la table sont en contradiction avec cette nouvelle contrainte, alors cette dernière ne sera pas appliquée et une erreur sera générée.

Syntaxe :

```
ALTER TABLE table ADD UNIQUE [contrainte] (attributs)
```

Exemple pour interdire tout doublon sur l'attribut *fax* de la table **Personnes** :

```
ALTER TABLE Personnes ADD UNIQUE u_fax (fax)
```

Autre exemple fictif :

```
ALTER TABLE Moto ADD UNIQUE u_coul_vitre (couleur,vitre)
```

# Changer la valeur par défaut d'un attribut

Pour changer ou supprimer la valeur par défaut d'un attribut.

Attention aux types qui n'acceptent pas de valeur par défaut (les familles **BLOB** et **TEXT**).

Syntaxe :

```
ALTER TABLE table ALTER attribut { SET DEFAULT valeur | DROP  
DEFAULT }
```

Changer sa valeur par défaut :

```
ALTER TABLE Personnes ALTER 'téléphone' SET DEFAULT  
'9999999999'
```

Supprimer sa valeur par défaut :

```
ALTER TABLE Personnes ALTER 'téléphone' DROP DEFAULT
```

Le changement ou la suppression n'affecte en rien les enregistrements qui ont eu recours à cette valeur lors de leur insertion.

# Changer le nom de la table

---

Syntaxe :

**ALTER TABLE *table* RENAME *nouveau\_nom***

Exemple :

**ALTER TABLE *Personnes* RENAME *Carnet***

Cela consiste à renommer la table, et donc le fichier qui la stocke.

# Ajouter un index

Une table ne peut comporter que 32 indexs.

Et un index ne peut porter que sur 16 attributs maximum à la fois.

Syntaxe :

```
ALTER TABLE table ADD INDEX index (attributs)
```

Exemple :

```
ALTER TABLE Personnes ADD INDEX nom_complet  
(nom,prénom)
```

Dans cet exemple, on a ajouté à la table ***Personnes*** un index que l'on nomme *nom\_complet* et qui s'applique aux deux attributs *nom* et '*prénom*'. Ainsi, les recherches et les tris sur les attributs *nom* et '*prénom*' seront grandement améliorés. Car un index apporte les changements sous-jacents permettant d'optimiser les performances du serveur de base de données.

# Supprimer un index

Syntaxe :

**ALTER TABLE *table* DROP INDEX *index***

Exemple :

**ALTER TABLE *Personnes* DROP INDEX *nom\_complet***

Cette exemple permet de supprimer l'index nommé *nom\_complet* de la table ***Personnes***.



# Ajouter un enregistrement (I)

## insertion étendue

Ajouter un enregistrement à une table revient à ajouter une ligne à la table. Pour cela, pour chacun des attributs, il faudra en préciser la valeur. Si certaines valeurs sont omises, alors les valeurs par défauts définies lors de la création de la table seront utilisées. Si on ne dispose pas non plus de ces **valeurs par défaut**, alors MySQL mettra 0 pour un nombre, "" pour une chaîne, 0000-00-00 pour une date, 00:00:00 pour une heure, 0000000000000000 pour un timestamp (si le champ porte la contrainte **NOT NULL**). Dans le cas où l'attribut porte la contrainte NULL (par défaut) alors la valeur par défaut de l'attribut – quel soit son type – sera la suivante : NULL.

Syntaxe d'une « insertion étendue » :

**INSERT INTO *table*(liste des attributs) VALUES(liste des valeurs)**

Exemple :

**INSERT INTO *Personnes*(nom,prénom) VALUES('Martin','Jean')**

REPLACE est un synonyme de INSERT, mais sans doublon. Pratique pour respecter les contraintes d'unicité (UNIQUE, PRIMARY KEY).

# Ajouter un enregistrement (II)

## *insertion standard*

Une syntaxe plus courte mais plus ambiguë permet d'insérer un enregistrement dans une table. Elle consiste à omettre la liste des noms d'attribut à la suite du nom de la table. Cela impose que la liste des valeurs suivant le mot clé VALUES soit exactement celle définie dans la table et qu'elles soient dans l'ordre défini dans la définition de la table ; sinon des erreurs se produiront.

Syntaxe d'une « insertion standard » :

**INSERT INTO *table* VALUES**(*liste exhaustive et ordonnée des valeurs*)

Exemple :

```
CREATE TABLE Ballon (  
    taille INT NOT NULL,  
    couleur VARCHAR(40)  
)
```

**INSERT INTO *Ballon* VALUES(20, 'rouge')** **ok**

**INSERT INTO *Ballon* VALUES('rouge', 20)** **faux**

**INSERT INTO *Ballon* VALUES('rouge')** **faux**

# Ajouter un enregistrement (III)

## insertion complète

Dans le cas où l'on souhaite procéder à l'insertion de plusieurs enregistrements les uns à la suite des autres, il y a deux méthodes :

- faire une boucle qui envoie autant d'INSERT que nécessaire au serveur
- faire une insertion dite « complète »

Syntaxe d'une « insertion complète » :

**INSERT INTO *table* VALUES** (*liste des valeurs*), (*liste d'autres valeurs*), (*liste d'encore d'autres valeurs*), ...

Exemple :

**INSERT INTO *Ballon* VALUES** (20, 'rouge'), (35, 'vert fluo'), (17, 'orange'), (28, 'céruleen')

Cet exemple est équivalent aux requêtes suivantes :

**INSERT INTO *Ballon* VALUES**(20, 'rouge')

**INSERT INTO *Ballon* VALUES**(35, 'vert fluo')

**INSERT INTO *Ballon* VALUES**(17, 'orange')

**INSERT INTO *Ballon* VALUES**(28, 'céruleen')

# Ajouter un enregistrement (IV)

## *insertion complète*

L'insertion complète permet d'insérer plusieurs enregistrements dans une même table. Une insertion complète ne permet pas d'insérer des données dans plusieurs tables différentes.

L'insertion complète et l'insertion étendue peuvent être associées dans une même requête :

```
INSERT INTO Ballon(taille, couleur) VALUES (20, 'rouge'), (35, 'vert fluo'), (17, 'orange'), (28, 'céruleen')
```

# Modifier un enregistrement (I)

Pour modifier un ou des enregistrement(s) d'une table, il faut préciser un critère de sélection des enregistrement à modifier (clause **WHERE**), il faut aussi dire quels sont les attributs dont on va modifier la valeur et quelles sont ces nouvelles valeurs (clause **SET**).

Syntaxe :

```
UPDATE [ LOW_PRIORITY ] table SET attribut=valeur, ... [ WHERE  
condition ] [ LIMIT a ]
```

Exemple :

```
UPDATE Personnes SET téléphone='0156281469' WHERE  
nom='Martin' AND prénom = 'Pierre'
```

Cet exemple modifie le numéro de téléphone de Martin Pierre.

**LOW\_PRIORITY** est une option un peu spéciale qui permet de n'appliquer la ou les modification(s) qu'une fois que plus personne n'est en train de lire dans la table.

## Modifier un enregistrement (II)

Il est possible de modifier les valeurs d'autant d'attributs que la table en contient.

Exemple pour modifier plusieurs attributs :

```
UPDATE Personnes SET téléphone='0156281469',  
fax='0156281812' WHERE id = 102
```

Pour appliquer la modification à tous les enregistrements de la table, il suffit de ne pas mettre de clause **WHERE**.

**LIMIT a** permet de n'appliquer la commande qu'aux **a** premiers enregistrements satisfaisant la condition définie par **WHERE**.

Autre exemple :

```
UPDATE Enfants SET age=age+1
```

Il est donc possible de modifier la valeur d'un attribut relativement à sa valeur déjà existante.

# Supprimer un enregistrement

Attention, la suppression est définitive !

Syntaxe :

**DELETE [ LOW\_PRIORITY ] FROM *table* [ WHERE condition ] [ LIMIT a ]**

Exemple :

**DELETE FROM *Personnes* WHERE *nom*='Martin' AND *prénom*='Marc'**

Pour vider une table de tous ces éléments, ne pas mettre de clause WHERE. Cela efface et recrée la table, au lieu de supprimer un à un chacun des tuples de la table (ce qui serait très long).

Exemple :

**DELETE FROM *Personnes***

# Optimisation

Après la suppression de grandes parties d'une table contenant des index, les index des tuples supprimés sont conservés, rallongeant d'autant les sélections. Pour supprimer ces index obsolètes et vider les « trous », il faut l'optimiser.

*Syntaxe :*

**OPTIMIZE TABLE *table***

*Exemple :*

**OPTIMIZE TABLE *Personnes***



# Sélectionner des enregistrements (I)

Pour extraire de votre base de données des informations, comme la liste des personnes de votre carnet d'adresse qui vivent à Paris.

Syntaxe générale :

```
SELECT [ DISTINCT ] attributs  
      [ INTO OUTFILE fichier ]  
      [ FROM table ]  
      [ WHERE condition ]  
      [ GROUP BY attributs [ ASC | DESC ] ]  
      [ HAVING condition ]  
      [ ORDER BY attributs ]  
      [ LIMIT [a,] b ]
```

Exemple :

```
SELECT nom,prénom FROM Personnes WHERE adresse LIKE  
'%paris%'
```

# Sélectionner des enregistrements (II)

Nom	Description
<b>SELECT</b>	Spécifie les attributs dont on souhaite connaître les valeurs.
<b>DISTINCT</b>	Permet d'ignorer les doublons de ligne de résultat.
<b>INTO OUTFILE</b>	Spécifie le fichier sur lequel effectuer la sélection.
<b>FROM</b>	Spécifie le ou les tables sur lesquelles effectuer la sélection.
<b>WHERE</b>	Définie le ou les critères de sélection sur des attributs.
<b>GROUP BY</b>	Permet de grouper les lignes de résultats selon un ou des attributs.
<b>HAVING</b>	Définie un ou des critères de sélection sur des ensembles de valeurs d'attributs après groupement.
<b>ORDER BY</b>	Permet de définir l'ordre ( <b>ASC</b> endant par défaut ou <b>DESC</b> endant) dans l'envoi des résultats.
<b>LIMIT</b>	Permet de limiter le nombre de lignes du résultats

# Sélectionner des enregistrements (III)

*Procédons par étapes :*

Pour sélectionner tous les enregistrements d'une table :

**SELECT \* FROM *table***

Pour sélectionner toutes les valeurs d'un seul attribut :

**SELECT *attribut* FROM *table***

Pour éliminer les doublons :

**SELECT DISTINCT *attribut* FROM *table***

Pour trier les valeurs en ordre croissant :

**SELECT DISTINCT *attribut* FROM *table* ORDER BY *attribut* ASC**

Pour se limiter aux **num** premiers résultats :

**SELECT DISTINCT *attribut* FROM *table* ORDER BY *attribut* ASC  
LIMIT num**

Pour ne sélectionner que ceux qui satisfont à une condition :

**SELECT DISTINCT *attribut* FROM *table* WHERE *condition* ORDER  
BY *attribut* ASC LIMIT num**

# Sélectionner des enregistrements (IV)

table de départ :

**SELECT \* FROM Gens**

Gens		
Nom	Prenom	Age
Dupond	Pierre	24
Martin	Marc	48
Dupont	Jean	51
Martin	Paul	36
Dupond	Lionel	68
Chirac	Jacques	70

**SELECT Nom FROM Gens**

Gens	
Nom	
Dupond	
Martin	
Dupont	
Martin	
Dupond	
Chirac	

**SELECT DISTINCT Nom FROM Gens**

Gens	
Nom	
Dupond	
Martin	
Dupont	
Chirac	

# Sélectionner des enregistrements (V)

<i>Gens</i>
<i>Nom</i>
Chirac
Dupond
Dupont
Martin

```
SELECT DISTINCT Nom
FROM Gens
ORDER BY Nom ASC
```

<i>Gens</i>
<i>Nom</i>
Chirac
Dupond

```
SELECT DISTINCT Nom
FROM Gens
ORDER BY Nom ASC
LIMIT 2
```

```
SELECT DISTINCT Nom
FROM Gens
WHERE Nom <> 'Chirac'
ORDER BY Nom ASC
LIMIT 2
```

<i>Gens</i>
<i>Nom</i>
Dupond

# Jointure évoluée (I)

En début de ce document, on a vu la jointure suivante :

```
SELECT Personnes.nom, nblivres  
FROM Personnes, Bibliothèque  
WHERE Personnes.nom = Bibliothèque.nom
```

qui permet de concaténer deux table en prenant un attribut comme pivot.

Il est possible de concaténer deux table sur plusieurs attributs à la fois, ou même de concaténer X table sur Y attributs.

Les requêtes utilisant très souvent les jointures, il a été créé une syntaxe spéciale plus rapide : JOIN que la méthode vue plus haut : avec la clause WHERE.

Ainsi la jointure précédente peut s'écrire aussi :

```
SELECT Personnes.nom, nblivres  
FROM Personnes INNER JOIN Bibliothèque  
USING (nom)
```

ce qui signifie que les deux tables **Personnes** et **Bibliothèque** sont concaténées (INNER JOIN) en utilisant (USING) l'attribut *nom*.

# Jointure évoluée (II)

La syntaxe USING permet de lister les attributs servant de pivot. Ces attributs doivent porter le même nom dans chacune des tables devant être concaténées. Si les attributs pivots ne portent pas le même nom, il faut utiliser la syntaxe ON.

Ainsi la jointure précédente peut s'écrire aussi :

```
SELECT Personnes.nom, nblivres
FROM Personnes INNER JOIN Bibliothèque
ON Personnes.nom = Bibliothèque.nom
```

La méthode **INNER JOIN** n'inclus les enregistrements de la première table que s'ils ont une correspondance dans la seconde table.

Personnes

Nom	Prénom
Martin	Jean
Tartan	Pion
Dupond	Jacques

Bibliothèque

Nom	Nblivres
Martine	5
Tartan	10
Dupond	3

Résultat de la jointure

Nom	Nblivres
Tartan	10
Dupond	3

# Jointure évoluée (III)

Pour remédier aux limites de **INNER JOIN**, il existe la syntaxe **LEFT JOIN** qui inclus tous les enregistrements de la première table même s'ils n'ont pas de correspondance dans la seconde table. Dans ce cas précis, l'attribut non renseigné prendra la valeur NULL.

Là encore, le **ON** peut avantageusement être remplacé par le **USING**.

La jointure devient :

```
SELECT Personnes.nom, nblivres
FROM Personnes LEFT JOIN Bibliothèque
ON Personnes.nom = Bibliothèque.nom
```

Personnes

Nom	Prénom
Martin	Jean
Tartan	Pion
Dupond	Jacques

Bibliothèque

Nom	Nblivres
Martine	5
Tartan	10
Dupond	3

Résultat de la jointure

Nom	Nblivres
Martin	NULL
Tartan	10
Dupond	3



# Exercice / TP SQL (1)

1. Créer la requête d'insertion d'un média
2. Modifier le mail de l'utilisateur 'untel' avec un mail en 'imt-nord-europe.fr'
3. Supprimer un type de média : quelles sont les conséquences ?
4. Afficher la liste des médias triés par nom (order) ainsi que leur type
5. Afficher les médias créés par un utilisateur (where)
6. Afficher les types de médias regardés par un utilisateur
7. Afficher les playlist d'un utilisateur et les films qui lui sont associé

# 3

## Fonctions de MySQL

# Les fonctions

Bien que ces fonctions appartiennent typiquement à MySQL, la création d'un chapitre à part se justifie par le fait que l'on va se contenter ici d'énumérer les fonctions les plus courantes.

Reportez-vous au manuel MySQL pour la liste détaillée de toutes les fonctions disponibles dans votre version du serveur MySQL.

Ces fonctions sont à ajouter à vos requêtes dans un SELECT, WHERE, GROUP BY ou encore HAVING.

D'abord sachez que vous avez à votre disposition :

- les parenthèses ( ),
- les opérateurs arithmétiques (+, -, \*, /, %),
- les opérateurs logiques qui retournent 0 (faux) ou 1 (vrai) (**AND &&**, **OR ||**, **NOT !**, **BETWEEN**, **IN**),
- les opérateurs relationnels (<, <=, =, >, >=, <>).

Les opérateurs et les fonctions peuvent être composés entre eux pour donner des expressions très complexes.

# Quelques exemples

```
SELECT nom
FROM produits
WHERE prix <= 100.5
```

Liste du nom des produits dont le prix est inférieur ou égale à 100.5 EUR.

```
SELECT nom,prénom
FROM élèves
WHERE age BETWEEN 12 AND 16
```

Liste des nom et prénom des élèves dont l'âge est compris entre 12 et 16 ans.

```
SELECT modèle
FROM voitures
WHERE couleur IN ('rouge', 'blanc', 'noir')
```

Liste des modèles de voiture dont la couleur est dans la liste : rouge, blanc, noir.

```
SELECT modèle
FROM voitures
WHERE couleur NOT IN ('rose', 'violet')
```

Liste des modèles de voiture dont la couleur n'est pas dans la liste : rose, violet.

# Fonctions de comparaison de chaînes

Le mot clé **LIKE** permet de comparer deux chaînes.

Le caractère **'%'** est spécial et signifie : 0 ou plusieurs caractères.

Le caractère **'\_'** est spécial et signifie : 1 seul caractère, n'importe lequel.

L'exemple suivant permet de rechercher tous les clients dont le prénom commence par 'Jean', cela peut être 'Jean-Pierre', etc... :

```
SELECT nom  
FROM clients  
WHERE prénom LIKE 'Jean%'
```

Pour utiliser les caractères spéciaux ci-dessus en leur enlevant leur fonction spéciale, il faut les faire précéder de l'antislash : **'\''**.

Exemple, pour lister les produits dont le code commence par la chaîne **'\_XE'** :

```
SELECT *  
FROM produit  
WHERE code LIKE '\_XE%'
```

# Fonctions mathématiques

Fonction	Description
ABS(x)	Valeur absolue de X.
SIGN(x)	Signe de X, retourne -1, 0 ou 1.
FLOOR(x)	Arrondi à l'entier inférieur.
CEILING(x)	Arrondi à l'entier supérieur.
ROUND(x)	Arrondi à l'entier le plus proche.
EXP(x), LOG(x), SIN(x), COS(x), TAN(x), PI()	Bon, là c'est les fonctions de maths de base...
POW(x,y)	Retourne X à la puissance Y.
RAND(), RAND(x)	Retourne un nombre aléatoire entre 0 et 1.0 Si x est spécifié, entre 0 et X
TRUNCATE(x,y)	Tronque le nombre X à la Yème décimale.

```
SELECT nom
FROM filiales
WHERE SIGN(ca) = -1
```

Cet exemple affiche dans un ordre aléatoire le nom des filiales dont le chiffre d'affaire est négatif.

```
ORDER BY RAND()
```

A noter que :  $\text{SIGN}(ca) = -1 \Leftrightarrow ca < 0$

# Fonctions de chaînes

Fonction	Description
TRIM(x)	Supprime les espaces de début et de fin de chaîne.
LOWER(x)	Converti en minuscules.
UPPER(x)	Converti en majuscules.
LONGUEUR(x)	Retourne la taille de la chaîne.
LOCATE(x,y)	Retourne la position de la dernière occurrence de x dans y. Retourne 0 si x n'est pas trouvé dans y.
CONCAT(x,y,...)	Concatène ses arguments.
SUBSTRING(s,i,n)	Retourne les n derniers caractères de s en commençant à partir de la position i.
SOUNDEX(x)	Retourne une représentation phonétique de x.

```
SELECT UPPER(nom)
FROM clients
WHERE SOUNDEX(nom) = SOUNDEX('Dupond')
```

On affiche en majuscules le nom de tous les clients dont le nom ressemble à 'Dupond'.

# Fonctions de dates et heures

Fonction	Description
NOW()	Retourne la date et heure du jour.
TO_DAYS(x)	Conversion de la date X en nombre de jours depuis le 1er janvier 1970.
DAYOFWEEK(x)	Retourne le jour de la semaine de la date x sous la forme d'un index qui commence à 1 (1=dimanche, 2=lundi...)
DAYOFMONTH(x)	Retourne le jour du mois (entre 1 et 31).
DAYOFYEAR(x)	Retourne le jour de l'année (entre 1 et 366).
SECOND(x), MINUTE(x), HOUR(x), MONTH(x), YEAR(x), WEEK(x)	Retournent respectivement les secondes, minutes, heures, mois, année et semaine de la date.

SELECT titre

FROM article

WHERE (TO\_DAYS(NOW()) – TO\_DAYS(parution)) < 30

Cet exemple affiche le titre des articles parus il y a moins de 30 jours.



# Fonctions à utiliser dans les GROUP BY

Fonction	Description
COUNT([DISTINCT]x,y,...)	Décompte des tuples du résultat par projection sur le ou les attributs spécifiés (ou tous avec '*'). L'option DISTINCT élimine les doublons.
MIN(x), MAX(x), AVG(x), SUM(x)	Calculent respectivement le minimum, le maximum, la moyenne et la somme des valeurs de l'attribut X.

```
SELECT DISTINCT model
FROM voiture
GROUP BY model
HAVING COUNT(couleur) > 10
```

Ici on affiche le palmarès des modèles de voitures qui proposent un choix de plus de 10 couleurs.

```
SELECT COUNT(*)
FROM client
```

Affichage de tous les clients.

```
SELECT DISTINCT produit.nom, SUM(vente.qt * produit.prix) AS total
FROM produit, vente
WHERE produit.id = vente.produit_idx
GROUP BY produit.nom
```

Classement des produits par la valeur totale vendue.

# ***Exercice / TP SGBD (3)***

---

A venir...