Table des matières

[Création de base de données 4](#_Toc437849158)

[Creation de Table 4](#_Toc437849159)

[Modification d’une Table 4](#_Toc437849160)

[Sélection de données 5](#_Toc437849161)

[Suppression et modification de donnée 6](#_Toc437849162)

[Types de données 7](#_Toc437849163)

[Nombres entiers 7](#_Toc437849164)

[Chaînes de type texte 7](#_Toc437849165)

[Insertion de données 8](#_Toc437849166)

[Index 9](#_Toc437849167)

[Clés primaire et étrangères 10](#_Toc437849168)

[Lors de la création de la table 10](#_Toc437849169)

[Après création de la table 11](#_Toc437849170)

[Jointure 12](#_Toc437849171)

[Jointure Interne 12](#_Toc437849172)

[Jointure externe 13](#_Toc437849173)

[Jointures par la gauche 13](#_Toc437849174)

[Jointures par la droite 13](#_Toc437849175)

[Sous requête 14](#_Toc437849176)

[JOINTURES ET SOUS-REQUÊTES : MODIFICATION DE DONNÉES 14](#_Toc437849177)

[Modification 15](#_Toc437849178)

[Suppression 15](#_Toc437849179)

[UNION DE PLUSIEURS REQUÊTES 15](#_Toc437849180)

[OPTIONS DES CLÉS ÉTRANGÈRES 16](#_Toc437849181)

[VIOLATION DE CONTRAINTE D'UNICITÉ 16](#_Toc437849182)

[OBTENIR LA DATE/L'HEURE ACTUELLE 17](#_Toc437849183)

[Fonction Scalaire 18](#_Toc437849184)

[https://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql/fonctions-scalaires 18](#_Toc437849185)

[**Manipulation de nombres** 18](#_Toc437849186)

[**Manipulation de chaînes de caractères** 18](#_Toc437849187)

[Fonctions d’agrégation 19](#_Toc437849188)

[Fonctions Statique 19](#_Toc437849189)

[Concaténation 21](#_Toc437849190)

[Regroupement 23](#_Toc437849191)

[Regroupement sur un ou plusieurs critères 23](#_Toc437849192)

[Super-agrégats 23](#_Toc437849193)

[Conditions sur les fonctions d’agrégation 24](#_Toc437849194)

[OBTENIR LA DATE/L'HEURE ACTUELLE 25](#_Toc437849195)

[FORMATER UNE DONNÉE TEMPORELLE 26](#_Toc437849196)

[Extraire une information précise 26](#_Toc437849197)

[Formater une date facilement 26](#_Toc437849198)

[Créer une date à partir d'une chaîne de caractères 26](#_Toc437849199)

[CALCULS SUR LES DONNÉES TEMPORELLES 26](#_Toc437849200)

[Différence entre deux dates/heures 26](#_Toc437849201)

[Ajout et retrait d'un intervalle de temps 26](#_Toc437849202)

[Divers 26](#_Toc437849203)

[TRANSACTIONS 26](#_Toc437849204)

[Les Verrous 27](#_Toc437849205)

[REQUÊTES PRÉPARÉES 28](#_Toc437849206)

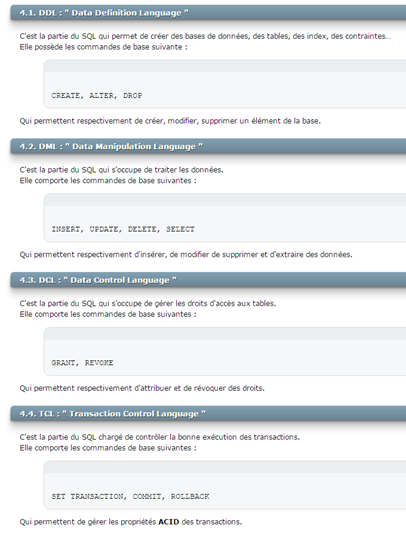
[Procédures Stockées 29](#_Toc437849207)

[STRUCTURER SES INSTRUCTIONs 31](#_Toc437849208)

[GESTIONNAIRES D'ERREURS, CURSEURS ET UTILISATION AVANCÉE 31](#_Toc437849209)

[Les Triggers 31](#_Toc437849210)

**SQL**



# Création de base de données

*-Pour créer une base de données, on utilise la commande CREATE DATABASE (IF EXISTS) nom base.*

*-Pour supprimer une base de données : DROP DATABASE nom base.*

*-À chaque connexion à MySQL, il faut préciser avec quelle base on va travailler, avec USE nom base.*

# Creation de Table

*CREATE TABLE [IF NOT EXISTS]* *Nom\_table* (

colonne1 description\_colonne1,

[colonne2 description\_colonne2,

colonne3 description\_colonne3,

...,]

*[PRIMARY KEY* (colonne\_clé\_primaire)]

*[INDEX*[nom\_index] (colonne1\_index [, colonne2\_index, ...]]

)

[ENGINE=moteur];

CREATE TABLE Animal (

id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,

espece VARCHAR(40) NOT NULL,

sexe CHAR(1),

date\_naissance DATETIME NOT NULL,

nom VARCHAR(30),

commentaires TEXT,

PRIMARY KEY (id)

)

ENGINE=INNODB;

DROP TABLE Animal; -- permet de supprimer une table

# Modification d’une Table

ALTER TABLE nom\_table ADD ... -- permet d'ajouter quelque chose (une colonne par exemple)

ALTER TABLE nom\_table DROP ... -- permet de retirer quelque chose

ALTER TABLE nom\_table CHANGE ...

ALTER TABLE nom\_table MODIFY ... -- permettent de modifier une colonne

# Sélection de données

* *La commande SELECT permet d'afficher des données.*
* *La clause WHERE permet de préciser des critères de sélection.*
* *Il est possible de trier les données grâce à ORDER BY, selon un ordre ascendant (ASC) ou descendant (DESC) ou (id).*
* *Pour éliminer les doublons, on utilise le mot-clé DISTINCT, juste après SELECT.*
* *Pour afficher les lignes, on utilise le mot-clé COUNT(\*) après SELECT.*
* *La clause GROUP BY  permet de grouper des critères.*
* *LIMIT nb\_lignes OFFSET decalage permet de sélectionner uniquement nb\_lignes de*
* *résultats, avec un certain décalage.*
* SELECT \*
* FROM Animal
* WHERE espece='chien'
* AND nom IS NOT NULL
* ORDER BY nom DESC;
* *L'opérateur LIKE permet de faire des recherches approximatives, grâce aux deux caractères "joker" : '%' (qui représente 0 ou plusieurs caractères) et '\_' (qui représente 1 caractère).*
* *L'opérateur BETWEEN permet de faire une recherche sur un intervalle. WHERE colonne BETWEEN a AND b étant équivalent à WHERE colonne >= a AND colonne <= b.*
* *Enfin, l'opérateur IN permet de faire une recherche sur une liste de valeurs.*
* [*AVG()*](http://sql.sh/fonctions/agregation/avg)*pour calculer la moyenne d’un set de valeur. Permet de connaître le prix du panier moyen pour de chaque client*
* [*COUNT()*](http://sql.sh/fonctions/agregation/count)*pour compter le nombre de lignes concernées. Permet de savoir combien d’achats a été effectué par chaque client*
* ***MAX()****pour récupérer la plus haute valeur. Pratique pour savoir l’achat le plus cher*
* ***MIN()****pour récupérer la plus petite valeur. Utile par exemple pour connaître la date du premier achat d’un client*
* ***SUM()****pour calculer la somme de plusieurs lignes. Permet par exemple de connaître le total de tous les achats d’un client*

| * **Opérateur** | **Signification** |
| --- | --- |
| = | égal |
| < | inférieur |
| <= | inférieur ou égal |
| > | supérieur |
| >= | supérieur ou égal |
| <> ou != | différent |
| <=> | égal (valable pour NULL aussi) |

| **Opérateur** | **Symbole** | **Signification** |
| --- | --- | --- |
| AND | && | ET |
| OR | || | OU |
| XOR |  | OU exclusif |
| NOT | ! | NON |

# Suppression et modification de donnée

* *Le client mysqldump est un programme qui permet de sauvegarder facilement ses bases de données.*
* *La commande DELETE permet de supprimer des données.*
* *La commande UPDATE permet de modifier des données.*
* On utilise la clause WHERE de la même manière qu'avec la commande SELECT pour préciser quelles lignes doivent être supprimées.
* DELETE FROM Animal
* WHERE nom = 'Zoulou';

La modification des données se fait grâce à la commande UPDATE, dont la syntaxe est la suivante :

UPDATE Animal

SET sexe='F', nom='Pataude'

WHERE id=21;

# Types de données

## Nombres entiers

Les types de données qui acceptent des nombres entiers comme valeur sont désignés par le mot-clé INT, et ses déclinaisons TINYINT, SMALLINT, MEDIUMINT et BIGINT. La différence entre ces types est le nombre d'octets (donc la place en mémoire) réservés à la valeur du champ. Voici un tableau reprenant ces informations, ainsi que l'intervalle dans lequel la valeur peut être comprise pour chaque type.

Il est possible de préciser le nombre de chiffres minimum **à l'affichage** d'une colonne de type INT (ou un de ses dérivés). Il suffit alors de préciser ce nombre entre parenthèses : INT(x).

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

## Chaînes de type texte

Pour stocker un texte relativement court (moins de 255 caractères), vous pouvez utiliser les types CHAR et VARCHAR. Ces deux types s'utilisent avec un paramètre qui précise la taille que peut prendre votre texte (entre 1 et 255). La différence entre CHAR et VARCHAR est la manière dont ils sont stockés en mémoire. Un CHAR(x) stockera toujours x caractères, en remplissant si nécessaire le texte avec des espaces vides pour le compléter, tandis qu'un VARCHAR(x) stockera jusqu'à x caractères (entre 0 et x), et stockera en plus en mémoire la taille du texte stocké.  
Si vous entrez un texte plus long que la taille maximale définie pour le champ, celui-ci sera tronqué.

Petit tableau explicatif, en prenant l'exemple d'un CHAR ou d'un VARCHAR de 5 caractères maximum :

| **Texte** | **CHAR(5)** | **Mémoire requise** | **VARCHAR(5)** | **Mémoire requise** |
| --- | --- | --- | --- | --- |
| '' | '   ' | 5 octets | '' | 1 octet |
| 'tex' | 'tex  ' | 5 octets | 'tex' | 4 octets |
| 'texte' | 'texte' | 5 octets | 'texte' | 6 octets |
| 'texte trop long' | 'texte' | 5 octets | 'texte' | 6 octets |

*Et si je veux pouvoir stocker des textes de plus de 255 caractères ?*

Il suffit alors d'utiliser le type TEXT, ou un de ses dérivés TINYTEXT, MEDIUMTEXT ou LONGTEXT. La différence entre ceux-ci étant la place qu'ils permettent d'occuper en mémoire. Petit tableau habituel :

| **Type** | **Longueur maximale** | **Mémoire occupée** |
| --- | --- | --- |
| TINYTEXT | 2^8 octets | Longueur de la chaîne + 1 octet |
| TEXT | 2^16 octets | Longueur de la chaîne + 2 octets |
| MEDIUMTEXT | 2^24 octets | Longueur de la chaîne + 3 octets |
| LONGTEXT | 2^32 octets | Longueur de la chaîne + 4 octets |

# Insertion de données

INSERT INTO Animal (espece, sexe, date\_naissance)

VALUES ('tortue', 'F', '2009-08-03 05:12:00');

INSERT INTO Animal (nom, commentaires, date\_naissance, espece)

VALUES ('Choupi', 'Né sans oreille gauche', '2010-10-03 16:44:00', 'chat');

INSERT INTO Animal (espece, date\_naissance, commentaires, nom, sexe)

VALUES ('tortue', '2009-06-13 08:17:00', 'Carapace bizarre', 'Bobosse', 'F');

Si vous avez plusieurs lignes à introduire, il est possible de le faire en une seule requête de la manière suivante :

INSERT INTO Animal (espece, sexe, date\_naissance, nom)

VALUES ('chien', 'F', '2008-12-06 05:18:00', 'Caroline'),

('chat', 'M', '2008-09-11 15:38:00', 'Bagherra'),

('tortue', NULL, '2010-08-23 05:18:00', NULL);

INSERT INTO Animal

SET nom='Bobo', espece='chien', sexe='M', date\_naissance='2010-07-21 15:41:00';

# Index

* *Un index permet d'accélérer les recherches faites sur les colonnes constituant celui-ci.*
* *Un index UNIQUE ne peut contenir qu'une seule fois chaque valeur (ou combinaison de valeurs si l'index est composite, c'est-à-dire sur plusieurs colonnes).*
* *Un index FULLTEXT permet de faire des recherches complexes sur le contenu des colonnes le constituant.*

ALTER TABLE nom\_table

ADD INDEX [nom\_index] (colonne\_index [, colonne2\_index ...]); --Ajout d'un index simple

ALTER TABLE nom\_table

ADD UNIQUE [nom\_index] (colonne\_index [, colonne2\_index ...]); --Ajout d'un index UNIQUE

ALTER TABLE nom\_table

ADD FULLTEXT [nom\_index] (colonne\_index [, colonne2\_index ...]); --Ajout d'un index FULLTEXT

INDEX ind\_nom (nom(10)) -- index sur le nom (le chiffre entre parenthèses étant le nombre de caractères pris en compte)

ALTER TABLE nom\_table -- suppression

DROP INDEX nom\_index;

*Voici la syntaxe utilisée pour faire une recherche FULLTEXT :*

SELECT \* -- Vous mettez évidemment les colonnes que vous voulez.

FROM nom\_table

WHERE MATCH(colonne1[, colonne2, ...]) -- La (ou les) colonne(s) dans laquelle (ou lesquelles) on veut faire la recherche (index FULLTEXT correspondant nécessaire).

AGAINST ('chaîne recherchée'); -- La chaîne de caractères recherchée, entre guillemets bien sûr.

##### *Recherche avec booléens*

SELECT \*

FROM Livre

WHERE MATCH(titre)

AGAINST ('chaîne recherchée' IN BOOLEAN MODE);

SELECT \*

FROM Livre

WHERE MATCH(titre)

AGAINST ('+bonheur -ogres' IN BOOLEAN MODE);

SELECT \*

FROM Livre

WHERE MATCH(titre)

AGAINST ('"Terre la Lune"' IN BOOLEAN MODE);

# Clés primaire et étrangères

* Une clé primaire permet d'identifier chaque ligne de la table de manière unique : c'est à la fois une contrainte d'unicité et une contrainte NOT NULL.
* Chaque table **doit définir une clé primaire**, et une table ne peut avoir qu'**une seule** clé primaire.

##### Une clé étrangère permet de définir une relation entre deux tables, et d'assurer la cohérence des données.

##### 

## Lors de la création de la table

* Création de clé Primaire

CREATE TABLE Animal (

id SMALLINT AUTO\_INCREMENT PRIMARY KEY,

espece VARCHAR(40) NOT NULL,

sexe CHAR(1),

date\_naissance DATETIME NOT NULL,

nom VARCHAR(30),

commentaires TEXT

)

* Création de clé étrangère

CREATE TABLE [IF NOT EXISTS] Nom\_table (

colonne1 description\_colonne1,

[colonne2 description\_colonne2,

colonne3 description\_colonne3,

...,]

[ [CONSTRAINT [symbole\_contrainte]] FOREIGN KEY (colonne(s)\_clé\_étrangère) REFERENCES table\_référence (colonne(s)\_référence)]

)

CREATE TABLE Commande (

numero INT UNSIGNED PRIMARY KEY AUTO\_INCREMENT,

client INT UNSIGNED NOT NULL,

produit VARCHAR(40),

quantite SMALLINT DEFAULT 1,

CONSTRAINT fk\_client\_numero -- On donne un nom à notre clé

FOREIGN KEY (client) -- Colonne sur laquelle on crée la clé

REFERENCES Client(numero) -- Colonne de référence

)

ENGINE=InnoDB; -- MyISAM interdit, je le rappelle encore une fois !

## Après création de la table

* Création de clé primaire

ALTER TABLE nom\_table

ADD [CONSTRAINT [symbole\_contrainte]] PRIMARY KEY (colonne\_pk1 [, colonne\_pk2, ...]);

#### Suppression de la clé primaire

ALTER TABLE nom\_table

DROP PRIMARY KEY

ALTER TABLE Commande

ADD CONSTRAINT fk\_client\_numero FOREIGN KEY (client) REFERENCES Client(numero);

##### Après création de la table

##### Création de clé étrangère

ALTER TABLE Commande

ADD CONSTRAINT fk\_client\_numero FOREIGN KEY (client) REFERENCES Client(numero);

#### Suppression d'une clé étrangère

ALTER TABLE nom\_table

DROP FOREIGN KEY symbole\_contrainte

# Jointure

## Jointure Interne

SELECT (reduc table).colonne

FROM Table (reduc table1)

INNER JOIN Table2 (reduc table2)

ON colonneT = colonneT2

WHERE colonne = 'valeur';

SELECT Espece.description

FROM Espece

INNER JOIN Animal

ON Espece.id = Animal.espece\_id

WHERE Animal.nom = 'Cartouche';

Décomposons !

SELECT Espece.description : je sélectionne la colonne description de la table Espece.

FROM Espece: je travaille sur la table Espece.

INNER JOIN Animal : je la joins (avec une jointure interne) à la table Animal.

ON Espece.id = Animal.espece\_id : la jointure se fait sur les colonnes id de la table Espece et espece\_id de la table Animal, qui doivent donc correspondre.

WHERE Animal.nom = 'Cartouche': dans la table résultant de la jointure, je sélectionne les lignes qui ont la valeur "Cartouche" dans la colonne nom venant de la table Animal.

Si vous avez compris ça, vous avez tout compris !

SELECT e.id,

e.description,

a.nom

FROM Espece AS e -- On donne l'alias "e" à Espece

INNER JOIN Animal AS a -- et l'alias "a" à Animal.

ON e.id = a.espece\_id

WHERE a.nom LIKE 'Ch%';

Comme vous le voyez, le code est plus compact. Ici encore, c'est quelque chose que j'utilise souvent pour de petites requêtes ponctuelles. Par contre, pour de grosses requêtes, je préfère les noms explicites ; c'est ainsi plus facile de s'y retrouver.

INNER JOIN permet donc de faire une jointure interne sur deux tables. Mais que signifie donc ce "interne" ?

Lorsque l'on fait une jointure interne, cela veut dire qu'on exige qu'il y ait des données de part et d'autre de la jointure. Donc, si l'on fait une jointure sur la colonne a de la table A et la colonne b

Ceci retournera **uniquement** les lignes pour lesquelles *A.a* et *B.b* correspondent.

## Jointure externe

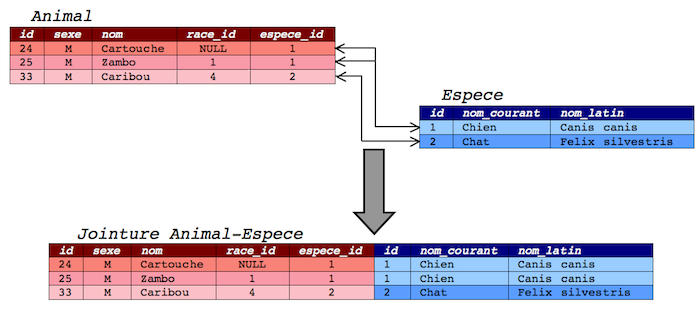
### Jointures par la gauche

Lorsque l'on fait une jointure par la gauche (grâce aux mots-clés LEFT JOIN ou LEFT OUTER JOIN), cela signifie que l'on veut toutes les lignes de la table de gauche (sauf restrictions dans une clause WHERE bien sûr), même si certaines n'ont pas de correspondance avec une ligne de la table de droite.

Alors, table de gauche, table de droite, laquelle est laquelle ? C'est très simple, nous lisons de gauche à droite, donc la table de gauche est la première table mentionnée dans la requête, c'est-à-dire, en général, la table donnée dans la clause FROM.

### Jointures par la droite

Les jointures par la droite (RIGHT JOIN ou RIGHT OUTER JOIN), c'est évidemment le même principe, sauf que ce sont toutes les lignes de la table de droite qui sont sélectionnées même s'il n'y a pas de correspondance dans la table de gauche.



# Sous requête

* Une sous-requête est une requête imbriquée dans une autre requête.
* Il est obligatoire de donner un alias au résultat d'une sous-requête lorsqu'on l'utilise dans une clause FROM.
* IN, ANY, SOME et ALL s'utilisent uniquement avec des sous-requêtes renvoyant une seule colonne.
* Une sous-requête corrélée est une sous-requête utilisant une table référencée uniquement dans la requête dont elle dépend, et non dans la sous-requête elle-même.
* EXISTS renvoie vrai si la sous-requête qui y est associée renvoie au moins un résultat.

Exemple

SELECT MIN(date\_naissance)

FROM (

SELECT Animal.id, Animal.sexe, Animal.date\_naissance, Animal.nom, Animal.espece\_id,

Espece.id AS espece\_espece\_id -- On renomme la colonne id de Espece, donc il n'y a plus de doublons.

FROM Animal -- Attention de ne pas la renommer espece\_id, puisqu'on sélectionne aussi la colonne espece\_id dans Animal !

INNER JOIN Espece

ON Espece.id = Animal.espece\_id

WHERE sexe = 'F'

AND Espece.nom\_courant IN ('Tortue d''Hermann', 'Perroquet amazone')

# JOINTURES ET SOUS-REQUÊTES : MODIFICATION DE DONNÉES

INSERT INTO Animal

(nom, sexe, date\_naissance, race\_id, espece\_id)

-- Je précise les colonnes puisque je ne donne pas une valeur pour toutes.

SELECT 'Yoda', 'M', '2010-11-09', id AS race\_id, espece\_id

-- Attention à l'ordre !

FROM Race WHERE nom = 'Maine coon';

Sélectionnons maintenant les Maine coon de notre base, pour vérifier que l'insertion s'est faite correctement.

SELECT Animal.id, Animal.sexe, Animal.nom, Race.nom AS race, Espece.nom\_courant as espece

FROM Animal

INNER JOIN Race ON Animal.race\_id = Race.id

INNER JOIN Espece ON Race.espece\_id = Espece.id

WHERE Race.nom = 'Maine coon';

## Modification

Voici la syntaxe que vous devriez utiliser pour le faire avec une jointure :

UPDATE Animal -- Classique !

INNER JOIN Espece -- Jointure.

ON Animal.espece\_id = Espece.id

-- Condition de la jointure.

SET Animal.commentaires = Espece.description

-- Ensuite, la modification voulue.

WHERE Animal.commentaires IS NULL

-- Seulement s'il n'y a pas encore de commentaire.

AND Espece.nom\_courant IN ('Perroquet amazone', 'Tortue d''Hermann');

-- Et seulement pour les perroquets et les tortues.

## Suppression

DELETE Animal -- Je précise de quelles tables les données doivent être supprimées

FROM Animal -- Table principale

INNER JOIN Espece ON Animal.espece\_id = Espece.id

-- Jointure

WHERE Animal.nom = 'Carabistouille'

AND Espece.nom\_courant = 'Chat';

# UNION DE PLUSIEURS REQUÊTES

La syntaxe d'UNION est simplissime : vous avez deux requêtes SELECT dont vous voulez additionner les résultats ; il vous suffit d'ajouter UNION entre ces deux requêtes.

SELECT ...

UNION

SELECT ...

##### En résumé

* Pour grouper les résultats de deux requêtes SELECT, il suffit de placer UNION entre les deux requêtes.
* UNION est équivalente à UNION DISTINCT. Si l'on ne veut pas éliminer les doublons, il faut utiliser UNION ALL.
* Il est possible de limiter les résultats d'un UNION (avec LIMIT), mais également de limiter les résultats de chaque requête composant l'UNION.
* Par contre, ORDER BY n'aura d'influence que s'il est utilisé sur le résultat final d'un UNION.

# OPTIONS DES CLÉS ÉTRANGÈRES

* Lorsque l'on crée (ou modifie) une clé étrangère, on peut lui définir deux options : ON DELETE, qui sert en cas de suppression de la référence ; et ON UPDATE, qui sert en cas de modification de la référence.
* RESTRICT et NO ACTION désignent le comportement par défaut : la référence ne peut être ni supprimée, ni modifiée si cela entraîne des données incohérentes vis-à-vis de la clé étrangère.
* SET NULL fait en sorte que les données de la clé étrangère ayant perdu leur référence (suite à une modification ou une suppression) soient mises à NULL.
* CASCADE répercute la modification ou la suppression d'une référence de clé étrangère sur les lignes impactées.

# VIOLATION DE CONTRAINTE D'UNICITÉ

##### En résumé

* Le mot-clé IGNORE, utilisé dans des requêtes INSERT, UPDATE ou LOAD DATA, permet de ne pas déclencher d'erreur en cas de violation d'une contrainte d'unicité : la ligne posant problème ne sera simplement pas insérée/modifiée.
* Utiliser REPLACE au lieu de INSERT (ou dans LOAD DATA) supprime les lignes existantes qui provoquent une violation de la contrainte d'unicité à l'insertion, puis insère la nouvelle ligne.
* Il est possible, en ajoutant une clause ON DUPLICATE KEY UPDATE à une requête INSERT INTO, de provoquer soit une insertion (si aucune contrainte d'unicité n'est violée), soit une modification de certaines valeurs de la ligne déjà existante (dans le cas contraire).

# OBTENIR LA DATE/L'HEURE ACTUELLE

* La date du jour peut s'obtenir avec CURDATE() et CURRENT\_DATE().
* L'heure actuelle peut s'obtenir avec CURTIME() et CURRENT\_TIME().
* L'heure et la date actuelles peuvent s'obtenir avec NOW(), SYSDATE(), LOCALTIME(),CURRENT\_TIMESTAMP(), LOCALTIMESTAMP().
* On peut insérer un DATETIME dans une colonne DATE ou TIME. MySQL ôtera la partie inutile.

# Fonction Scalaire

<https://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql/fonctions-scalaires>

**Manipulation de nombres**

Arrondis

Exposants et racines

Hasard

Divers

**Manipulation de chaînes de caractères**

Longueur et comparaison

Retrait et ajout de caractères

Récupérer une sous-chaîne

Recherche et remplacement

Concaténation

Code ASCII

En résumé

Concaténer deux chaînes de caractères signifie les mettre bout à bout.

Il ne faut pas hésiter, pour obtenir le résultat voulu, à combiner plusieurs fonctions ensemble.

Pour savoir si un nombre est multiple d'un autre, on peut utiliser le modulo.

# Fonctions d’agrégation

## Fonctions Statique

La plupart des fonctions d'agrégation vont vous permettre de faire des statistiques sur vos données.

#### Nombre de lignes

La fonction COUNT() permet de savoir combien de lignes sont sélectionnées par la requête.

-- Combien de races avons-nous ? --

-- ---------------------------------

SELECT COUNT(\*) AS nb\_races

FROM Race;

-- Combien de chiens avons-nous ? --

-- ---------------------------------

SELECT COUNT(\*) AS nb\_chiens

FROM Animal

INNER JOIN Espece ON Espece.id = Animal.espece\_id

WHERE Espece.nom\_courant = 'Chien';

| **nb\_races** |
| --- |
| 8 |

| **nb\_chiens** |
| --- |
| 21 |

##### COUNT(\*) ou COUNT(colonne)

Vous l'avez vu, j'ai utilisé COUNT(\*) dans les exemples ci-dessus. Cela signifie que l'on compte tout simplement les lignes, sans se soucier de ce qu'elles contiennent.

Par contre, si on utilise COUNT(colonne), seules les lignes dont la valeur de colonnen'est pas NULL seront prises en compte.

**Exemple** : comptons les lignes de la table Animal, avec COUNT(\*) et COUNT(race\_id).

SELECT COUNT(race\_id), COUNT(\*)

FROM Animal;

| **COUNT(race\_id)** | **COUNT(\*)** |
| --- | --- |
| 31 | 60 |

Il n'y a donc que 31 animaux sur nos 60 pour lesquels la race est définie.

##### Doublons

Comme dans une requête SELECT tout à fait banale, il est possible d'utiliser le mot-clé DISTINCT pour ne pas prendre en compte les doublons.

**Exemple** : comptons le nombre de races distinctes définies dans la table Animal.

SELECT COUNT(DISTINCT race\_id)

FROM Animal;

| **COUNT(DISTINCT race\_id)** |
| --- |
| 7 |

Parmi nos 31 animaux dont la race est définie, on trouve donc 7 races différentes.

#### Minimum et maximum

Nous avons déjà eu l'occasion de rencontrer la fonction MIN(x), qui retourne la plus petite valeur de x. Il existe également une fonction MAX(x), qui renvoie la plus grande valeur de x.

SELECT MIN(prix), MAX(prix)

FROM Race;

| **MIN(prix)** | **MAX(prix)** |
| --- | --- |
| 485.00 | 1235.00 |

Notez que MIN() et MAX() ne s'utilisent pas uniquement sur des données numériques. Si vous lui passez des chaînes de caractères, MIN() récupérera la première chaîne dans l'ordre alphabétique, MAX() la dernière ; avec des dates, MIN()renverra la plus vieille et MAX() la plus récente.

**Exemple** :

SELECT MIN(nom), MAX(nom), MIN(date\_naissance), MAX(date\_naissance)

FROM Animal;

| **MIN(nom)** | **MAX(nom)** | **MIN(date\_naissance)** | **MAX(date\_naissance)** |
| --- | --- | --- | --- |
| Anya | Zonko | 2006-03-15 14:26:00 | 2010-11-09 00:00:00 |

#### Somme et moyenne

##### Somme

La fonction SUM(x) renvoie la somme de x.

SELECT SUM(prix)

FROM Espece;

| **SUM(prix)** |
| --- |
| 1200.00 |

##### Moyenne

La fonction AVG(x) (du mot anglais average) renvoie la valeur moyenne de x.

SELECT AVG(prix)

FROM Espece;

| **AVG(prix)** |
| --- |
| 240.000000 |

## Concaténation

#### Principe

Avec les fonctions d'agrégation, on regroupe plusieurs lignes. Les fonctions statistiques nous permettent d'avoir des informations fort utiles sur le résultat d'une requête, mais parfois, il est intéressant d'avoir également les valeurs concernées.  
Ceci est faisable avec GROUP\_CONCAT(nom\_colonne). Cette fonction concatène les valeurs de nom\_colonnepour chaque groupement réalisé.

**Exemple** : on récupère la somme des prix de chaque espèce, et on affiche les espèces concernées par la même occasion.

SELECT SUM(prix), GROUP\_CONCAT(nom\_courant)

FROM Espece;

| **SUM(prix)** | **GROUP\_CONCAT(nom\_courant)** |
| --- | --- |
| 1200.00 | Chien,Chat,Tortue d'Hermann,Perroquet amazone,Rat brun |

#### Syntaxe

Voici la syntaxe de cette fonction :

GROUP\_CONCAT(

[DISTINCT] col1 [, col2, ...]

[ORDER BY col [ASC | DESC]]

[SEPARATOR sep]

)

* DISTINCT : sert comme d'habitude à éliminer les doublons.
* col1 : est le nom de la colonne dont les valeurs doivent être concaténées. C'est le **seul argument obligatoire**.
* col2,… : sont les éventuelles autres colonnes (ou chaînes de caractères) à concaténer.
* ORDER BY : permet de déterminer dans quel ordre les valeurs seront concaténées.
* SEPARATOR : permet de spécifier une chaîne de caractères à utiliser pour séparer les différentes valeurs. Par défaut, c'est une virgule.

#### Exemples

-- --------------------------------------

-- CONCATENATION DE PLUSIEURS COLONNES --

-- --------------------------------------

SELECT SUM(Race.prix), GROUP\_CONCAT(Race.nom, Espece.nom\_courant)

FROM Race

INNER JOIN Espece ON Espece.id = Race.espece\_id;

-- ---------------------------------------------------

-- CONCATENATION DE PLUSIEURS COLONNES EN PLUS JOLI --

-- ---------------------------------------------------

SELECT SUM(Race.prix), GROUP\_CONCAT(Race.nom, ' (', Espece.nom\_courant, ')')

FROM Race

INNER JOIN Espece ON Espece.id = Race.espece\_id;

-- ---------------------------

-- ELIMINATION DES DOUBLONS --

-- ---------------------------

SELECT SUM(Espece.prix), GROUP\_CONCAT(DISTINCT Espece.nom\_courant)

-- Essayez sans le DISTINCT pour voir

FROM Espece

INNER JOIN Race ON Race.espece\_id = Espece.id;

-- --------------------------

-- UTILISATION DE ORDER BY --

-- --------------------------

SELECT SUM(Race.prix), GROUP\_CONCAT(Race.nom, ' (', Espece.nom\_courant, ')' ORDER BY Race.nom DESC)

FROM Race

INNER JOIN Espece ON Espece.id = Race.espece\_id;

-- ----------------------------

-- CHANGEMENT DE SEPARATEUR --

-- ----------------------------

SELECT SUM(Race.prix), GROUP\_CONCAT(Race.nom, ' (', Espece.nom\_courant, ')' SEPARATOR ' - ')

FROM Race

INNER JOIN Espece ON Espece.id = Race.espece\_id;

##### En résumé

* La plupart des fonctions d'agrégation permettent de faire des statistiques sur les données : nombre de lignes, moyenne d'une colonne,…
* COUNT(\*) compte toutes les lignes quel que soit leur contenu, tandis que COUNT(colonne\_x) compte les lignes pour lesquelles colonne\_x n'est pas NULL.
* GROUP\_CONCAT(colonne\_x) permet de concaténer les valeurs de colonne\_x dont les lignes ont été groupées par une autre fonction d'agrégation.

# Regroupement

## Regroupement sur un ou plusieurs critères

Pour regrouper les lignes selon un critère, il faut utiliser GROUP BY, qui se place après l'éventuelle clause WHERE (sinon directement après FROM), suivi du nom de la colonne à utiliser comme critère de regroupement.

SELECT ...

FROM nom\_table

[WHERE condition]

GROUP BY nom\_colonne;

## Super-agrégats

Parlons maintenant de l'option WITH ROLLUP de GROUP BY. Cette option va afficher des lignes supplémentaires dans la table de résultats. Ces lignes représenteront des "super-groupes" (ou super-agrégats), donc des "groupes de groupes". Deux petits exemples, et vous aurez compris !

##### Exemple avec un critère de regroupement

SELECT nom\_courant, COUNT(\*) as nb\_animaux

FROM Animal

INNER JOIN Espece ON Espece.id = Animal.espece\_id

GROUP BY nom\_courant WITH ROLLUP;

| **nom\_courant** | **nb\_animaux** |
| --- | --- |
| Chat | 20 |
| Chien | 21 |
| Perroquet amazone | 4 |
| Tortue d'Hermann | 15 |
| NULL | 60 |

##### NULL, c'est pas joli

Il est possible d'éviter d'avoir ces NULL dans les lignes des super-groupes. Pour cela, on peut utiliser la fonction COALESCE().  
Cette fonction prend autant de paramètres que l'on veut, et renvoie le premier paramètre non NULL.

**Exemples** :

SELECT COALESCE(1, NULL, 3, 4); -- 1

SELECT COALESCE(NULL, 2); -- 2

SELECT COALESCE(NULL, NULL, 3); -- 3

Voici comment l'utiliser dans le cas des super-agrégats.

SELECT COALESCE(nom\_courant, 'Total'), COUNT(\*) as nb\_animaux

FROM Animal

INNER JOIN Espece ON Espece.id = Animal.espece\_id

GROUP BY nom\_courant WITH ROLLUP;

| **COALESCE(nom\_courant, 'Total')** | **nb\_animaux** |
| --- | --- |
| Chat | 20 |
| Chien | 21 |
| Perroquet amazone | 4 |
| Tortue d'Hermann | 15 |
| Total | 60 |

Tant qu'il s'agit de simples groupes, nom\_courant contient bien le nom de l'espèce. COALESCE() renvoie donc celui-ci. Par contre, quand il s'agit du super-groupe, la colonne nom\_courant du résultat contient NULL, et donc COALESCE() va renvoyer "Total".

## Conditions sur les fonctions d’agrégation

Il n'est pas possible d'utiliser la clause WHERE pour faire des conditions sur une fonction d'agrégation. Donc, si l'on veut afficher les espèces dont on possède plus de 15 individus, la requête suivante ne fonctionnera pas.

SELECT nom\_courant, COUNT(\*)

FROM Animal

INNER JOIN Espece ON Espece.id = Animal.espece\_id

WHERE COUNT(\*) > 15

GROUP BY nom\_courant;

ERROR 1111 (HY000): Invalid use of group function

Il faut utiliser une clause spéciale : HAVING. Cette clause se place juste après le GROUP BY.

SELECT nom\_courant, COUNT(\*)

FROM Animal

INNER JOIN Espece ON Espece.id = Animal.espece\_id

GROUP BY nom\_courant

HAVING COUNT(\*) > 15;

| **nom\_courant** | **COUNT(\*)** |
| --- | --- |
| Chat | 20 |
| Chien | 21 |

Il est également possible d'utiliser un alias dans une condition HAVING :

SELECT nom\_courant, COUNT(\*) as nombre

FROM Animal

INNER JOIN Espece ON Espece.id = Animal.espece\_id

GROUP BY nom\_courant

HAVING nombre > 15;

| **nom\_courant** | **nombre** |
| --- | --- |
| Chat | 20 |
| Chien | 21 |

# OBTENIR LA DATE/L'HEURE ACTUELLE

les différents types de données temporelles, qui sont au nombre de cinq : DATE, TIME, DATETIME, TIMESTAMP et YEAR.

##### En résumé

* La date du jour peut s'obtenir avec CURDATE() et CURRENT\_DATE().
* L'heure actuelle peut s'obtenir avec CURTIME() et CURRENT\_TIME().
* L'heure et la date actuelles peuvent s'obtenir avec NOW(), SYSDATE(), LOCALTIME(),CURRENT\_TIMESTAMP(), LOCALTIMESTAMP().
* On peut insérer un DATETIME dans une colonne DATE ou TIME. MySQL ôtera la partie inutile.

# FORMATER UNE DONNÉE TEMPORELLE

<https://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql/formater-une-donnee-temporelle>

## Extraire une information précise

## Formater une date facilement

## Créer une date à partir d'une chaîne de caractères

# CALCULS SUR LES DONNÉES TEMPORELLES

<https://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql/calculs-sur-les-donnees-temporelles>

## Différence entre deux dates/heures

## Ajout et retrait d'un intervalle de temps

## Divers

# TRANSACTIONS

<https://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql/transactions>

##### En résumé

* Les transactions permettent de **grouper plusieurs requêtes**, lesquelles seront validées (COMMIT) ou annulées (ROLLBACK) toutes en même temps.
* Tous les changements de données (insertion, suppression, modification) faits par les requêtes à l'intérieur d'une transaction sont **invisibles pour les autres sessions tant que la transaction n'est pas validée**.
* Les transactions permettent d'**exécuter un traitement nécessitant plusieurs requêtes en une seule fois**, ou de l'annuler complètement si une des requêtes pose problème ou si la transaction est interrompue.
* Certaines commandes SQL provoquent une **validation implicite des transactions**, notamment toutes les commandes DDL, c'est-à-dire les commandes qui créent, modifient ou suppriment des objets dans la base de données (tables, index,…).
* Les critères ACID sont les critères qu'un système appliquant les transactions doit respecter pour être fiable :**Atomicité, Cohérence, Isolation, Durabilité**.

# Les Verrous

<https://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql/verrous>

##### En résumé

* Les verrous permettent de **restreindre, voire interdire, l'accès à une partie des données**.
* Les **verrous de table** peuvent s'utiliser sur des tables transactionnelles et non-transactionnelles, contrairement aux**verrous de ligne** qui ne sont disponibles que pour des tables transactionnelles.
* Les **verrous de lecture (tables) et partagés (lignes)** permettent aux autres sessions de lire les données verrouillées, mais pas de les modifier. Les **verrous d'écriture (tables) et exclusif (lignes)** par contre, ne permettent aux autres sessions ni de lire, ni de modifier les données verrouillées.
* Les verrous de ligne s'utilisent **avec les transactions**, et dépendent des **index**.
* Les requêtes de **suppression, modification et insertion posent automatiquement un verrou** de ligne exclusif de clé suivante sur les lignes concernées par la requête. Les requêtes de sélection par contre, ne posent pas de verrou par défaut, il faut en poser un explicitement.
* Le comportement par défaut des verrous de ligne est défini par le **niveau d'isolation des transactions**, qui est modifiable.

# REQUÊTES PRÉPARÉES

#### Créer et modifier une variable utilisateur

##### SET

La manière la plus classique de créer ou modifier une variable utilisateur est d'utiliser la commande SET.

SET @age = 24;

-- Ne pas oublier le @

SET @salut = 'Hello World !', @poids = 7.8;

-- On peut créer plusieurs variables en même temps

Si la variable utilisateur existe déjà, sa valeur sera modifiée, sinon, elle sera créée.

SELECT @age, @poids, @salut;

| **@age** | **@poids** | **@salut** |
| --- | --- | --- |
| 24 | 7.8 | Hello World ! |

##### Opérateur d'assignation

Il est également possible d'assigner une valeur à une variable utilisateur directement dans une requête, en utilisant l'opérateur d'assignation **:=** (n'oubliez pas les **:**, sinon il s'agit de l'opérateur de comparaison de valeurs).

SELECT @age := 32, @poids := 48.15, @perroquet := 4;

| **@age := 32** | **@poids := 48.15** | **@perroquet := 4** |
| --- | --- | --- |
| 32 | 48.15 | 4 |

PREPARE nom\_requete

FROM 'requete\_preparable';

**Exemples**

-- Sans paramètre

PREPARE select\_race

FROM 'SELECT \* FROM Race';

-- Avec un paramètre

PREPARE select\_client

FROM 'SELECT \* FROM Client WHERE email = ?';

-- Avec deux paramètres

PREPARE select\_adoption

FROM 'SELECT \* FROM Adoption WHERE client\_id = ? AND animal\_id = ?';

Plusieurs choses importantes :

* Le nom de la requête préparée ne doit pas être entre guillemets. Par contre la requête à préparer, si. **La requête à préparer doit être passée comme une chaîne de caractères**.
* Que le paramètre soit un nombre (client\_id = ?) ou une chaîne de caractères (email = ?), cela ne change rien. **On ne met pas de guillemets autour du ?** à l'intérieur de la requête à préparer.
* La chaîne de caractères contenant la requête à préparer ne peut contenir qu'**une seule requête** (et non plusieurs séparées par un **;**).
* **Les paramètres ne peuvent représenter que des données, des valeurs**, pas des noms de tables ou de colonnes, ni des morceaux de commandes SQL.

Comme la requête à préparer est donnée sous forme de chaîne de caractères, il est également possible d'utiliser une variable utilisateur, dans laquelle on enregistre tout ou partie de la requête à préparer.

# Procédures Stockées

#### Procédure avec une seule requête

Voici une procédure toute simple, sans paramètres, qui va juste afficher toutes les races d'animaux.

CREATE PROCEDURE afficher\_races\_requete()

-- pas de paramètres dans les parenthèses

SELECT id, nom, espece\_id, prix FROM Race;

#### Procédure avec un bloc d'instructions

Pour délimiter un bloc d'instructions (qui peut donc contenir plus d'une instruction), on utilise les mots BEGIN et END.

BEGIN

-- Série d'instructions

END;

CREATE PROCEDURE nom\_procedure ([parametre1 [, parametre2, ...]])

corps de la procédure;

#### Création d'une procédure stockée

DELIMITER | -- On change le délimiteur

CREATE PROCEDURE afficher\_races()

-- toujours pas de paramètres, toujours des parenthèses

BEGIN

SELECT id, nom, espece\_id, prix

FROM Race; -- Cette fois, le ; ne nous embêtera pas

END| -- Et on termine bien sûr la commande CREATE PROCEDURE par notre nouveau délimiteur

Cette fois-ci, tout se passe bien. La procédure a été créée.

#### Utilisation d'une procédure stockée

Pour appeler une procédure stockée, c'est-à-dire déclencher l'exécution du bloc d'instructions constituant le corps de la procédure, il faut utiliser le mot-clé CALL, suivi du nom de la procédure appelée, puis de parenthèses (avec éventuellement des paramètres).

CALL afficher\_races()| -- le délimiteur est toujours | !!!

#### Exemples

##### Procédure avec un seul paramètre entrant

Voici une procédure qui, selon l'id de l'espèce qu'on lui passe en paramètre, affiche les différentes races existant pour cette espèce.

DELIMITER | -- Facultatif si votre délimiteur est toujours |

CREATE PROCEDURE afficher\_race\_selon\_espece (IN p\_espece\_id INT)

-- Définition du paramètre p\_espece\_id

BEGIN

SELECT id, nom, espece\_id, prix

FROM Race

WHERE espece\_id = p\_espece\_id; -- Utilisation du paramètre

END |

DELIMITER ; -- On remet le délimiteur par défaut

Notez que, suite à la création de la procédure, j'ai remis le délimiteur par défaut **;**. Ce n'est absolument pas obligatoire, vous pouvez continuer à travailler avec **|** si vous préférez.

Pour l'utiliser, il faut donc passer une valeur en paramètre de la procédure. Soit directement, soit par l'intermédiaire d'une variable utilisateur.

CALL afficher\_race\_selon\_espece(1);

SET @espece\_id := 2;

CALL afficher\_race\_selon\_espece(@espece\_id);

On peut passer des **paramètres** à une procédure stockée, qui peuvent avoir trois sens : IN (entrant), OUT (sortant) ou INOUT (les deux).

# STRUCTURER SES INSTRUCTIONS

##### En résumé

* Un bloc d'instructions est délimité par BEGIN et END. Il est possible d'imbriquer plusieurs blocs d'instructions.
* Une variable locale est définie dans un bloc d'instructions grâce à la commande DECLARE. Une fois la fin du bloc d'instructions atteinte, toutes les variables locales qui y ont été déclarées sont supprimées.
* Une structure conditionnelle permet d'exécuter une série d'instructions si une condition est respectée. Les deux structures conditionnelles de MySQL sont IF et CASE.
* Une boucle est une structure qui permet de répéter une série d'instructions un certain nombre de fois. Il existe trois types de boucle pour MySQL : WHILE, REPEAT et LOOP.
* L'instruction LEAVE permet de quitter un bloc d'instructions ou une boucle.
* L'instruction ITERATE permet de relancer une itération d'une boucle.

# GESTIONNAIRES D'ERREURS, CURSEURS ET UTILISATION AVANCÉE

<https://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql/gestionnaires-d-erreurs-curseurs-et-utilisation-avancee>

# Les Triggers

<https://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql/triggers>

**En résumé**

* Un trigger est un objet **stocké dans la base de données**, à la manière d'une table ou d'une procédure stockée. La seule différence est qu'un trigger est **lié à une table**, donc en cas de suppression d'une table, les triggers liés à celle-ci sont supprimés également
* Un trigger **définit une ou plusieurs instructions**, dont l'exécution est **déclenchée par une insertion, une modification ou une suppression** de données dans la table à laquelle le trigger est lié.
* Les instructions du trigger peuvent être exécutées **avant la requête ayant déclenché celui-ci, ou après**. Ce comportement est à définir à la création du trigger.
* Une table ne peut posséder qu'un seul trigger par combinaison événement/moment (BEFORE UPDATE, AFTER DELETE,…)
* Les triggers sous MySQL sont soumis à d'importantes (et potentiellement très gênantes) **restrictions**.

Sécuriser une base de données et automatiser les traitements ne se limite bien sûr pas à ce que nous venons de voir. Les deux prochaines parties vous donneront de nouveaux outils pour avoir une base de données bien construite, sûre et efficace.

…