# BASE DE DONNEES

**Erreur safe update :**

SET SQL_SAFE_UPDATES = 0;

**Les types de données**

| | | |
|---|---|---|
| TINYINT | -128 | 127 |
| SMALLINT | -32768 | 32767 |
| MEDIUMINT | -8388608 | 8388608 |
| INT | -2147483648 | 2147483648 |
| BIGINT | -9223372036854775808 | 9223372036854775808 |
| UNSIGNED TINYINT | 0 | 255 |
| UNSIGNED SMALLINT | 0 | 65535 |
| UNSIGNED MEDIUMINT | 0 | 16777215 |
| UNSIGNED INT | 0 | 4294967295 |
| UNSIGNED BIGINT | 0 | 1.846744e+19 |
| INT(4) ZEROFILL | Ajoute des 0 si la taille du int ne contient pas au minimum 4 chiffre (pour 45, ça donnera 0045) | |
| DECIMAL(5,3) ou NUMERIC(5,3) | Stockage en forme de chaine de caractère de 5 chiffre maximum, dont de 3 après la virgule (arrondissement automatique si trop de chiffre après la virgule) | |
| FLOAT ou DOUBLE | Nombre décimal sans paramètre | |
| CHAR(x) | A prendre **si on connais** exactement le nombre de caractère qu'il doit s'y trouver | |
| VARCHAR(x) | A prendre **si on ne connais pas** exactement le nombre de caractère qu'il doit s'y trouver | |
| TINYTEXT<br>TEXT<br>MEDIUMTEXT<br>LONGTEXT | 2^8 octets<br>2^16 octets<br>2^24 octets<br>2^32 octets | |
| especе ENUM('chat', 'chien', 'tortue')<br><br>especе SET('chat', 'chien', 'tortue') | Seul chat chien ou tortue peuvent être entré, sinon stockera une chaine vide par défault<br><br>Comme ENUM mais peut stocket plusieurs en même temps (chat, chien – chat, tortue – chien, chat, tortue…) | |
| DATETIME | AAAA-MM-JJ HH:MM:SS | |
| DATE | AAAA-MM-JJ | |
| TIME | HH:MM:SS | |
| YEAR | Seulement l'année (de 1901 à 2155) | |

**Création de base de données**

| | |
|---|---|
| CREATE DATABASE nom_base SET 'utf8'; | Cree la base de données du nom nom_base d'encodage utf8 |
| DROP DATABASE nom_base; | Supprime la base nom_base |
| DROP DATABASE IF EXISTS nom_base; | Supprime la base nom_base à condition qu'elle existe |
| USE nom_base | A partir de cette ligne, les commandes entrée seront pour la base nom_base |

**Création de tables**

| | |
|---|---|
| CREATE TABLE IF NOT EXISTS Nom_table (<br>colonne1 description_colonne1,<br><br>colonne2 description_colonne2,<br>colonne3 description_colonne3,<br>[PRIMARY KEY (colonne_clé_primaire)]<br>)<br>[ENGINE= INNODB]; | Cree table Nom_table si elle n'existe pas<br>Nom colonne1 type (id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,)<br>Nom colonne2 type (espece VARCHAR(40) NOT NULL,)<br>Nom colonne3 type (sexe CHAR(1),)<br>Clé primaire (PRIMARY KEY (id)) |
| DROP TABLE Nom_table; | Supprime la table Nom_table |

**MODIFICATION D'UNE TABLE**

| | |
|---|---|
| ALTER TABLE nom_table<br>ADD COLUMN nom_colonne description_colonne; | Dans la table nom_table<br>Ajoute colonne nom_colonne |
| ALTER TABLE Commande<br>ADD CONSTRAINT fk_client_numero FOREIGN KEY (client) REFERENCES Client(numero); | Dans la table Commande<br>Ajoute nom_foreign_key la foreign key client qui fait reference a la table client(numero) |
| ALTER TABLE nom_table<br>DROP COLUMN nom_colonne; | Dans la table nom_table<br>supprime colonne nom_colonne |
| ALTER TABLE nom_table<br>CHANGE ancien_nom nouveau_nom description_colonne; | Dans la table nom_table<br>Change nom de colonne par un autre (CHANGE nom prenom VARCHAR(10) NOT NULL;) |
| ALTER TABLE nom_table<br>MODIFY nom_colonne nouvelle_description; | Dans la table nom_table<br>Modifie la description (MODIFY nom VARCHAR(30) NOT NULL DEFAULT 'Blabla') |
| TRUNCATE nom_table ; | Vide **entièrement** la table nom_table |

**Insertion de données**

| | |
|---|---|
| INSERT INTO Animal (espece, sexe, date_naissance)<br><br>VALUES ('tortue', 'F', '2009-08-03 05:12:00'); | Insert dans Animal et colonne espece, sexe, date_naissance<br>Les valeurs respectivement 'tortue', 'F', '2009-08-03 05:12:00' |
| INSERT INTO Animal (espece, sexe, date_naissance, nom)<br>VALUES ('chien', 'F', '2008-12-06 05:18:00', 'Caroline'),<br>('chat', 'M', '2008-09-11 15:38:00', 'Bagherra'),<br>('tortue', NULL, '2010-08-23 05:18:00', NULL); | Insertion multiple |
| ~~INSERT INTO Animal<br>SET nom='Bobo', espece='chien', sexe='M',<br>date_naissance='2010-07-21 15:41:00';~~ | ~~Autre façon d'insérer des donnée (propre à SQL et interdit l'insertion multiple)~~ |

**SÉLECTION DE DONNÉES**

| | |
|---|---|
| SELECT nom_colonne<br>FROM Animal | Selectionne nom_colonne<br>Dans la table nom_table |
| UNION | Relie 2 select (à mettre à la fin du premier et avant la seconde) |
| SELECT DISTINCT espece<br>FROM Animal; | Selectionne sans les doublons (donnera une seule fois le mot chien si il y en a plusieur par exemple) |
| WHERE espece='chien'; | Condition espèce = chien |
| WHERE espece LIKE c%<br><br>WHERE espece LIKE _<br><br>WHERE espece LIKE BINARY '%Lu%';<br><br>WHERE espece NOT LIKE %a% | Condition espèce contient c… (n'importe quel chaine après le c<br>Condition espèce contient 1seule caractère (P_rl_ pourrais etre egale a Parle – Perle – Perla)<br>La majuscule est obligatoire<br><br>Espece ne contient pas la lettre a |
| WHERE date_naissance BETWEEN '2008-01-05' AND '2009-03-23'; | Date de naissance compris entre 2008-01-05 et 2009-03-23 |
| WHERE nom IN ('Moka', 'Bilba', 'Tortilla', 'Balou', 'Dana', 'Redbul', 'Gingko'); | Condition nom = Moka ou Bilba ou Tortilla… |
| **Comparaisons where :**<br>=<br><<br><=<br>><br>>=<br><> ou !=<br><=> | **Comparaisons where :**<br>Égal<br>Inférieur<br>Inférieur ou égal<br>Supérieur<br>Supérieur ou égal<br>Différent<br>égal (valable pour NULL aussi) |
| **Plusieurs valeurs dans where**<br>AND<br>OR<br>XOR<br>NOT | **Plusieurs valeurs dans where**<br>ET<br>OU<br>OU exclusif<br>NON |
| **APRES LE WHERE SEULEMENT**<br>ORDER BY nom_colonne<br>ORDER BY nom_colonne DESC<br>ORDER BY nom_colonne, nom_colonne2; | **APRES LE WHERE SEULEMENT**<br>Trie par nom_colonne croissant<br>Trie par nom_colonne décroissant<br>Trie par nom_colonne et ensuite par nom_colonne2 |
| LIMIT 6 OFFSET 3;.. | Nombre de ligne sélectionner = 3 à partir de la ligne 3 |
| SHOW TABLES; | Pour voir TOUTES les tables dans un select |
| DESCRIBE nom_table; | Montre TOUTES les colonnes d'une table (ou vue) |

**SUPPRESSION ET MODIFICATION DE DONNÉES**

| | |
|---|---|
| DELETE FROM nom_table<br>WHERE critères; | Supprime dans nom_table<br>Les éléments dont la condition est… |
| UPDATE Animal<br>SET sexe='F', nom='Pataude'<br>WHERE id=21; | Modifie dans la classe Animal<br>Sexe = F et nom = Pataude<br>A condition que l'ID soit 21 |

**Jointures**

| APRES FROM SEULEMENT | APRES FROM SEULEMENT |
|---|---|
| INNER JOIN Animal<br>ON Espece.id = Animal.espece_id | Joins avec la table Animal<br>Ou l'id d'espece = l'id d'Animal |
| LEFT JOIN Animal<br>ON Espece.id = Animal.espece_id | A utiliser pour voir les table qui ont au moins **1 null** dans la sélection |
| WHERE espece.id = Animal.espece_id | Equivalent à inner join avec la condition where |

**Sous requête**

| | |
|---|---|
| SELECT MIN(date_naissance)<br>FROM (<br>SELECT Animal.id, Animal.nom<br>FROM Animal<br>INNER JOIN Espece<br>ON Espece.id = Animal.espece_id<br>WHERE sexe = 'F'<br>AND Espece.nom_courant IN ('Tortue d''Hermann',<br>'Perroquet amazone')<br>) AS tortues_perroquets_F; | Sélectionne la date de naissance la plus basse<br>Dans… (sous-requête dans from)<br><br><br><br><br><br><br>que l'on appellera Tortues_perroquer_F |
| SELECT id, nom, espece_id<br>FROM Race<br>WHERE espece_id < (<br>SELECT id<br>FROM Espece<br>WHERE nom_courant = 'Tortue d''Hermann'); | Sélectionne id, nom, espèce<br>Dans la classe Race<br>Condition l'id d'espèce est égale a (sous requête) |

# Fonctions : nombres, chaînes et agrégats

**Fonctions scalaire (agit indépendamment sur chaque ligne)**

| | |
|---|---|
| ROUND(prix) | Fait un arrondi au nombre décimal le plus proche |
| CEIL(prix) | Arrondit a l'entier supérieur |
| FLOOR(prix) | Arrondit a l'entier inférieur |
| POW(2, 5) | Exposant (ici 2^5) |
| SQRT(4) | Racine carré |
| RAND() | Nombre aléatoire entre 0 et 1 |
| SIGN(nombre) | Renvoie le signe du nombre<br>Négatif : renvoie -1<br>zéro : renvoie 0<br>positif : renvoie +1 |
| ABS(nombre) | Renvoie la valeur absolu du nombre |
| MOD(nombre, division) | Retourne le reste de la division |
| BIT_LENGTH(chaine) :<br>CHAR_LENGTH(chaine) :<br>LENGTH(chaine) : | Retourne le nombre de bits de la chaîne.<br>Retourne le nombre de caractères de la chaîne.<br>retourne le nombre d'octets de la chaîne. |
| STRCMP(chaine1, chaine2) | Compare les deux chaînes passées en paramètres<br>Retourne :<br>0 si les chaînes sont les mêmes,<br>-1 si la première chaîne est classée avant dans l'ordre alphabétique<br>1 dans le cas contraire. |
| LPAD("chaineAModifier", taille, 'supplément caractère')<br>RPAD("chaineAModifier", taille, 'supplément caractère') | Modifie la taille de la chaine et rajoute des caractères à gauche ou à droite si la chaine est trop courte.<br>Gauche<br>Droite |
| TRIM(LEADING FROM 'chaine')<br>TRIM(TRAILING FROM 'chaine')<br>TRIM(BOTH 'e' FROM 'chaine') | Supprime les caracteres inutile (par default les espaces)<br><br>Supprime les espaces à droite<br>Supprime les espaces à gauche<br>Supprime les e à gauche ET à droite |
| SUBSTRING('texte',3)<br>SUBSTRING('texte', 2, 3) | Récupère une partie de la chaine<br>A partir de la 3eme lettre incluse (xte)<br>De la 2eme lettre, de longueur 3 (ext) |
| INSTR(chaine, rech) | Retourne la position de la recherche dans chaine<br>("texte","xt" donnera : 3" |
| LOWER(chaine) | Met en minuscule |
| UPPER(chaine) | Met en majuscule |
| LEFT('123456789', 5), | Prendre les 5 premiers chiffres à gauche (12345) |
| RIGHT('123456789', 5) | Prendre les 5 premiers chiffres à droite (56789) |
| REVERSE('abcde'); | Inverse la chaine |
| REPLACE('texte', 'e', 'a') | Remplace les e par des a |
| CONCAT('My', 'SQL', '!') | Concatène |
| WS('-', 'My', 'SQL', '!'); | Concatène avec séparateur (ici le "-") |
| FIELD('Bonjour', 'Bonjour !', 'Au revoir', 'Bonjour', 'Au revoir !') | Recherche Bonjours **(valeur 1)**, dans les chaines suivantes, renvoi sa position **(ici 3)** |
| ASCII('T'), | Renvoi la valeur ascii de T **(ici 84)** |
| Char('84') | Renvoi la valeur char de 84 **(ici T)** |

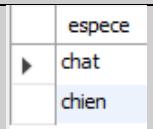
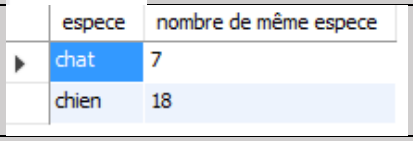
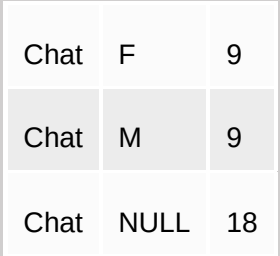
**Fonction d'agrégation (renvoi une seule ligne, le reste aura la même valeur si plusieurs lignes)**
**SAUF SI GROUP BY UTILISE**

| | |
|---|---|
| COUNT(*) | Renvoi le nombre de ligne sélectionné |
| COUNT(race_id) | Renvoi le nombre de race id des lignes sélectionné |
| MIN(prix) | Renvoi le minimum sur toute la sélection |
| MAX(prix) | Renvoi le maximum sur toute la sélection |
| SUM(prix) | Renvoi la somme |
| AVG(prix) | Renvoi la moyenne |
| PI() | Renvoie pi avec décimales |
| REPEAT('fort ! Trop ', 4); | Répète 4 fois, fort ! Trop |
| GROUP_CONCAT(nom_colonne) | Concatène tous les nom_trouvé de la colonne séparé par une virgule |

**Enlève des colonnes si besoin :**

| | |
|---|---|
| DISTINCT selection | Enlève les doublons |

**Regroupement :**

| | |
|---|---|
| GROUP BY espece_id; | Regroupe par espece_id renverra qu'une fois l'espece (à mettre après where)<br>Avec COUNT : écrit le nombre d'une espèce par ligne |
| select espece<br>from animal<br>group by espece; |  |
| select espece, count(espece)<br>from animal<br>group by espece; |  |
| GROUP BY nom_courant, sexe WITH ROLLUP; | Sous-groupe de groupe exemple pour :<br>GROUP BY nom_courant, sexe WITH ROLLUP<br>  |
| COALESCE(nom_courant, 'Total') | Renvoi nom_courant si il n'est pas null, sinon Total (peu prendre autant de paramètre, et renvoi le **premier paramètre** not null) |
| HAVING COUNT(*) > 15; | APRES GROUP BY, fonctionne comme where, obligatoire pour les count **(group by doit groupé avant de pouvoir faire la condition), préférable a utiliser que pour les fonctions d'agregation** |

# Fonctions : manipuler les dates

**OBTENIR LA DATE/L'HEURE ACTUELLE**

Les TYPES

| | |
|---|---|
| DATE | Doit être de la forme **AAAA-MM-JJ**<br>**de '1001-01-01' à '9999-12-31'** |
| TIME | La valeur doit être de la forme **HH:MM:SS**<br>de '**-838:59:59**' à '**838:59:59**' |
| DATETIME | Doit être de la forme **AAAA-MM-JJ HH:MM:SS**<br>de '**1001-01-01 00:00:00**' à '**9999-12-31 23:59:59**' |
| TIMESTAMP | Doit être un **int** de forme **AAAAMMJJHHMMSS**<br>du **1er janvier 1970 00h00min00s** au **19 janvier 2038 03h14min07s.** |
| YEAR | Doit être un **int** de forme **AAAA**<br>De **1901** et **2155** |

Récupération :

| | |
|---|---|
| SET lc_time_names = 'fr_FR'; | Met les date (écrit en lettre) en français |
| SELECT CURDATE() | Récupère la date : 2011-10-25 |
| SELECT CURTIME() | Récupère l'heure : 18:04:20 |
| SELECT NOW() | Récupère data et heure : 2011-10-26 09:40:18 |
| SELECT UNIX_TIMESTAMP(); | Nombre de seconde depuis le premier janvier 1970, à 00:00:00 |
| DAY(date) | Donne le jour du mois **(sous forme de nombre entier de 1 à 31)** |
| DAYOFWEEK(date) | Donne l'index du jour de la semaine **(nombre de 1 à 7 avec 1 = dimanche, 2 = lundi,…7 = samedi)** |
| WEEKDAY(date) | Donne l'index du jour de la semaine, **(nombre de 0 à 6 avec 0 = lundi, 1 = mardi,…6 = dimanche)** |
| DAYNAME(date) | Donne le nom du jour de la semaine |
| DAYOFYEAR(date) | Retourne le numéro du jour par rapport à l'année **(de 1 à 366 donc)** |
| WEEK(date) | Donne uniquement le numéro de la semaine **(un nombre entre 0 et 53)** |
| YEARWEEK(date) | Donne également l'année **(200709)** |
| MONTH(date) | Donne le numéro du mois **(nombre de 1 à 12)** |
| MONTHNAME(date) | Donne le nom du mois. |
| YEAR(date) | extrait l'année. |
| TIME(datetime) | Extrait l'heure complète (le TIME) ; |
| HOUR(heure) | Extrait l'heure ; |
| MINUTE(heure) | Extrait les minutes ; |
| SECOND(heure) | Extrait les secondes. |
| DATE_FORMAT(date_naissance, 'le %W %e %M %Y') | Ecrit la date sous le forme souhaiter<br>%d Jour du mois (nombre à deux chiffres, de 00 à 31)<br>%e Jour du mois (nombre à un ou deux chiffres, de 0 à 31)<br>%w jour de la semaine (dimanche = 0,…, samedi = 6)<br>%W Nom du jour de la semaine<br>%a Nom du jour de la semaine en abrégé<br>%m Mois (nombre de deux chiffres, de 00 à 12)<br>%c Mois (nombre d'un ou deux chiffres, de 0 à 12)<br>%M Nom du mois<br>%b Nom du mois en abrégé<br>%y Année, sur deux chiffres<br>%Y Année, sur quatre chiffres<br>%r Heure complète, format 12h (hh:mm:ss AM/PM)<br>%T Heure complète, format 24h (hh:mm:ss)<br>%h Heure sur deux chiffres et sur 12 heures (de 00 à 12)<br>%H Heure sur deux chiffres et sur 24 heures (de 00 à 23)<br>%l Heure sur un ou deux chiffres, sur 12 heures (de 0 à 12)<br>%k Heure sur un ou deux chiffres, sur 24 heures (de 0 à 23)<br>%i Minutes (de 00 à 59)<br>%s Secondes (de 00 à 59)<br>%p AM/PM |

**CALCULS SUR LES DONNÉES TEMPORELLES**

| | |
|---|---|
| DATEDIFF(date1,date2) | Nombre de jours entre date1 et date2 |
| TIMEDIFF(time1, time2) | Durée entre time1 et time2 **(type time ou datetime)** |
| ADDDATE(now(), INTERVAL '1 -01:00:00' DAY_SECOND) | Date avec décalage **(mettre signe – pour négatif)** jour heure:minute:seconde |
| MAKEDATE(2012, 60) | crée une DATE à partir d'une année et d'un numéro de jour **(1 étant le premier janvier, 32 le premier février, etc.).** |
| MAKETIME(3, 45, 34) | crée un TIME à partir d'une heure et d'un nombre de minutes et de secondes |
| LAST_DAY('2012-02-03') | Donne le dernier jour du mois (ici : 2012-02-29) |

# Sécuriser et automatiser ses actions

**Transaction :**

Par default, MySQL est en mode autocommit, il valide automatiquement les requêtes et ne peut donc pas revenir en arrière

| | |
|---|---|
| SET autocommit=0; | Enleve l'autocommit, on devra nous même dire de valider ou non les requêtes. (à remettre en début de chaque session si utilisé)<br><br>**Si il y a une coupure avant un commit, les dernières insertion seront annulé** |
| COMMIT;<br>ROLLBACK; | Pour valider les requêtes<br>Pour annuler les requêtes |
| START TRANSACTION; | Permet de commencer une transaction sans avoir a mettre autocommit a 0<br><br>**Se stop à partir d'une validation ou d'une annulation (commit, rollback). Les prochaines requêtes seront donc autocommit jusqu'au prochain start** |
| SAVEPOINT jalon1; | Caree un point de sauvagarde jalon1 |
| ROLLBACK TO SAVEPOINT jalon1; | Annule les requêtes faites après la sauvegarde du nom de jalon1 |
| | |

**Verrous :**

| | |
|---|---|
| LOCK TABLES nom_table [AS alias_tab] [READ \| WRITE] | Met un verrou sur nom_table<br>**READ** : verrou de lecture. Les autres sessions pourront toujours lire les données des tables verrouillées, mais ne pourront plus les modifier.<br>**WRITE** : verrou d'écriture. Les autres sessions ne pourront plus ni lire ni modifier les données des tables verrouillées.<br><br>**AS** : si on n'en met pas, on ne pourra accédé seulement sans alias<br>si on le met, on pourra accédé seulement avec un alias du même nom que alias_tab dans **from** |
| SELECT * FROM Animal WHERE espece_id = 5 LOCK IN SHARE MODE; | Je suis en train de lire ces données. Vous pouvez venir les lire aussi, mais pas les modifier tant que je n'ai pas terminé |
| **Attention : le verrou précèdent s'annule seulement si un autre select ne relie pas de valeur appartenant à une même ligne demandé.** | |
| SELECT * FROM Animal WHERE espece_id = 5 FOR UPDATE; | Je suis en train de lire ces données dans le but probable de faire une modification. Ne les lisez pas avant que j'aie fini (et bien sûr, ne les modifiez pas). |
| **Attention : le verrou précèdent se fait seulement dans une transaction, le verrou est levé quand il y a un commit ou un rollback** | |
| show index from animal; | Pour voir les index de la table animal<br><br>Un verrou prendra toutes les lignes si le where contient une condition sans index |

| | |
|---|---|
| SET [GLOBAL \| SESSION] TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED \| READ COMMITTED \| REPEATABLE READ \| SERIALIZABLE } | Définie un niveau d'isolation<br><br>**GLOBAL :** définit le niveau d'isolation pour toutes les sessions MySQL qui seront créées dans le futur. Les sessions existantes ne sont pas affectées.<br>**SESSION :** définit le niveau d'isolation pour la session courante.<br>Si l'on ne précise ni GLOBAL, ni SESSION, le niveau d'isolation défini ne concernera que la prochaine transaction que l'on ouvrira dans la session courante.<br><br>**REPEATABLE READ : niveau par défaut**, celui avec lequel vous travaillez depuis le début. *Repeatable read* signifie "lecture répétable", c'est-à-dire que si l'on fait plusieurs requêtes de sélection (**non-verrouillante**) de suite, elles donneront toujours le même résultat, quels que soient les changements effectués par d'autres sessions.<br>Si l'on pense à bien utiliser les verrous là où c'est nécessaire, c'est un niveau d'isolation tout à fait suffisant.<br>**READ COMMITTED :**<br>Avec ce niveau d'isolation, chaque requête SELECT (non-verrouillante) va reprendre une "photo" à jour de la base de données, même si plusieurs SELECTS se font dans la même transaction. Ainsi, un SELECT verra toujours les derniers changements comités, même s'ils ont été faits dans une autre session, après le début de la transaction.<br><br>**SERIALIZABLE :** Ce niveau d'isolation se comporte comme REPEATABLE READ, sauf que lorsque le mode autocommit est désactivé, tous les SELECT simples sont implicitement convertis en SELECT ... LOCK IN SHARE MODE. |

**REQUÊTES PRÉPARÉES : stocké que pour la session**

Variable utilisateur : (seulement pour la session, les autres ne voient pas cette variable)

| | |
|---|---|
| SET @age = 24;<br>SET @salut = 'Hello World !', @poids = 7.8; | Créé une variable |
| SELECT @age, @poids, @salut; | Affiche les variables |
| **une variable ne peut pas être utilisé pour stocker un nom de table ou de colonne qu'on introduirait directement dans la requête. Il faut l'inclure dans une requête préparé avant.** | |

**Requête prépare:**

| | |
|---|---|
| PREPARE select_adoption<br>FROM 'SELECT * FROM Adoption WHERE client_id = ? AND animal_id = ?'; | Nom de la requête prépare : select_adoption<br>La requête qui sera faite quand on appellera select_adoption |
| SET @colonne = 'nom';<br>SET @req_animal = CONCAT('SELECT ', @colonne, ' FROM Animal WHERE id = ?');<br>PREPARE select_col_animal<br>FROM @req_animal; | Syntaxe quand on utilise une variable,<br>penser à concaténé dans une autre variable toute le chaine.<br>Préparation de la requête<br><br>Résultat obtenu : 'SELECT nom FROM Animal WHERE id = ?' |
| EXECUTE select_adoption USING @client, @id; | Exécute la requête préparé avec les variables client et id |
| **En une fois :**<br><br>PREPARE select_adoption<br>FROM 'SELECT * FROM Adoption WHERE client_id = ? AND animal_id = ?';<br><br>SET @id = 3;<br>SET @client = 2;<br>EXECUTE select_adoption USING @client, @id; | **En une fois :**<br><br>Prépare une requête avec le nom select_adoption<br>Les '?' sont les parties qui seront remplacé<br><br><br>Cree la variable id<br>Cree la variable client<br>Exécute la requête select_adoption en utilisant les variables client et id qui remplacement dans l'ordre les '?' |
| DEALLOCATE PREPARE select_adoption; | Supprime la requête préparé select_adoption |

**Procédure stocké : stocké permanant**

| | |
|---|---|
| DELIMITER \|<br><br>CREATE PROCEDURE afficher_races(PARAMETRES)<br>BEGIN<br>  SELECT id, nom, espece_id, prix<br>  FROM Race;<br><br>END\|<br>DELIMITER ; | Change le délimiteur ; par \|<br><br>Créé une procédure du nom afficher_race()<br>Début de la procédure<br><br>On peut faire autant de requête que l'on veut<br><br>Fin de la procédure<br>Rechange le délimiteur pour les requêtes normal<br><br>PARAMETRES : variable type (exemple : p_espece_id INT) |
| CALL afficher_races(PARAMETRES A ENVOYER); | Appelle la procédure afficher_races()<br><br>PARAMETRES : variable qu'on envoie (exemple : @espece_id) |
| DELIMITER \|<br>CREATE PROCEDURE compter_races_selon_espece (IN p_espece_id INT, OUT p_nb_races INT)<br>BEGIN<br>  SELECT COUNT(*) INTO p_nb_races<br>  FROM Race<br>  WHERE espece_id = p_espece_id;<br>END \|<br>DELIMITER ; | IN : valeur entrant (utilisable que par la procédure)<br>OUT : valeur de sortie (la procédure modifie seulement)<br>INOUT : les deux au-dessus en même temps<br><br>Le select est inséré dans la variable p_nb_races qui peut être réutilisé n'importe ou **(attention, il faut une seule ligne)** |
| DROP PROCEDURE afficher_races; | Supprime une procedure |

**STRUCTURER SES INSTRUCTIONS dans les procédures :**

| | |
|---|---|
| DECLARE nom_variable type_variable [DEFAULT valeur_defaut]; | Déclare une variable LOCAL, à mettre après un BEGIN |
| IF condition THEN<br>instructions<br>ELSEIF autre_condition THEN<br>instructions<br>ELSE<br>instructions<br>END IF; | SI<br><br>SINON SI<br><br>SINON<br><br>FIN SI |
| CASE v_sexe<br>WHEN 'F' THEN -- Première possibilité<br>SELECT 'Je suis une femelle !' AS sexe;<br>WHEN 'M' THEN -- Deuxième possibilité<br>SELECT 'Je suis un mâle !' AS sexe;<br>ELSE<br>SELECT 'Je suis en plein questionnement existentiel...' AS sexe;<br>END CASE; | Variable v_sexe<br>SI v_SEXE = f<br><br>SINON SI v_sexe = M<br><br>SINON<br><br><br>FIN SI |
| WHILE condition DO<br><br>END WHILE; | Boucle TANT QUE |
| super_while: WHILE condition DO<br><br>END WHILE; | Donne le nom super_while a la boucle |
| LEAVE super_while; | Quitte la boucle super_while **(fonctionne aussi avec les procédures : corps_procedure: BEGIN)** |
| ITERATE boucle_while; | Toute les instruction après cette ligne jusqu'au prochain retour de boucle est annulé |
| SELECT id, nom, CASE<br>WHEN sexe = 'M' THEN 'Je suis un mâle !'<br>WHEN sexe = 'F' THEN 'Je suis une femelle !'<br>ELSE 'Je suis en plein questionnement existentiel...'<br>END AS message<br>FROM Animal<br>WHERE id IN (9, 8, 6); | Une condition CASE directement fais dans le select **sans procédure** |
| SELECT nom, IF(sexe = 'M', 'Je suis un mâle', 'Je ne suis pas un mâle') AS sexe<br>FROM Animal<br>WHERE espece_id = 5; | Condition if dans select :<br>condition, valeur si vrai, valeur si faut |

**Curseur :**

| | |
|---|---|
| DECLARE curseur_client CURSOR<br>FOR SELECT *<br>FROM Client; | Déclare un curseur curseur_client<br>Cree la requête select |
| OPEN nom_curseur;<br>-- Parcours du curseur et instructions diverses<br>CLOSE nom_curseur; | Ouverture/fermeture curseur |
| FETCH curs_clients INTO v_nom, v_prenom;<br>SELECT CONCAT(v_prenom, ' ', v_nom) AS 'Premier client';<br><br>FETCH curs_clients INTO v_nom, v_prenom;<br>SELECT CONCAT(v_prenom, ' ', v_nom) AS 'Second client'; | On récupère la première ligne<br>On assigne les valeurs récupérées à nos variables locales<br><br>On récupère la seconde ligne<br>on assigne les valeurs récupérées à nos variables locales |
| BEGIN<br>declare variable1 type;<br>declare variable2 type;<br>declare done type;<br><br>declare curseur CURSOR for (select IDPER,TPSETAPE from participe where IDETA = _idetape AND TPSETAPE != "00:00:00" order by TPSETAPE);<br><br>declare continue handler for not found set done = true;<br><br>open curseur;<br>repeat<br>fetch curseur into variable1, variable2;<br><br><br>until done end repeat;<br>close curseur;<br>END | Exemple de curseur<br>Déclaration des variables<br><br>Déclaration de done (obligatoire)<br><br>Déclaration du curseur (faire le select a l'intérieur)<br><br>Met done a true dès qu'il y a une erreur<br><br>Ouvre le curseur<br>Boucle<br>Lit la ligne suivante et met le résultat trouver respectivement dans les variables<br><br>Fin boucle<br>Ferme le curseur |

**Triggers :**

| | |
|---|---|
| CREATE TRIGGER nom_trigger moment_trigger evenement_trigger<br>ON nom_table FOR EACH ROW<br>~~corps_trigger~~;<br>BEGIN<br>-- Instructions<br>END \| | Déclaration d'un trigger<br><br>**nom_trigger** : nom du trigger.<br>**moment_trigger :** quand le trigger est déclenché.<br>**evenement_trigger** : comment le trigger est déclenché.<br>**nom_table** : à quelle table le trigger est attaché.<br>**FOR EACH ROW** : pour chaque ligne<br>**corps_trigger** : contenu du trigger.<br><br>**Exemple** :<br><br>CREATE TRIGGER after_insert_animal AFTER INSERT<br>ON Animal FOR EACH ROW<br>**corps_trigger**; |
| OLD.id<br>NEW.id | **OLD** : représente les valeurs des colonnes de la ligne traitée avant qu'elle ne soit modifiée par l'événement déclencheur. Ces valeurs peuvent être lues, mais pas |

| | |
|---|---|
| | modifiées.<br>**NEW** : représente les valeurs des colonnes de la ligne traitée après qu'elle a été modifiée par l'événement déclencheur. Ces valeurs peuvent être lues et modifiées. |
| DROP TRIGGER nom_trigger; | Supprime un trigger |

**Les vues :**

| | |
|---|---|
| CREATE [OR REPLACE] VIEW V_Animal_details<br>AS SELECT Animal.id, Animal.sexe, Animal.date_naissance, Animal.nom, Animal.commentaires,<br>Animal.espece_id, Animal.race_id, Animal.mere_id, Animal.pere_id, Animal.disponible,<br>Espece.nom_courant AS espece_nom, Race.nom AS race_nom<br>FROM Animal<br>INNER JOIN Espece ON Animal.espece_id = Espece.id<br>LEFT JOIN Race ON Animal.race_id = Race.id; | Cree une vue V_Animal_detail<br>Requête de la vue<br><br>OR REPLACE non obligatoire (ne pas mettre les crochets), remplace la vue si elle existe déjà. |
| SELECT * FROM V_Animal_details;<br><br>SELECT id FROM V_Animal_details; | Sélectionnes toutes les données dans la vue V_animal_detail<br>Sélectionne juste la conne id |
| ATTENTION, le select d'une vue est figé. Si la table est modifier, la vue n'aura pas les modifications<br>L'ORDER BY n'est pas prioritaire, s'il y en a un sdans le select, c'est celui-ci qui sera pris en compte | |
| DROP VIEW V_Race; | Supprime la vue V_Race |

**Creation / modification / suppression utilisateurs :**

| | |
|---|---|
| CREATE USER 'login'@'hote' [IDENTIFIED BY 'mot_de_passe']; | Cree un utilisateur<br><br>Login : son login<br>hote : l'ip ou adresse ou il peut se connecter (mettre % pour "n'importe où"<br><br>Identified by : crée mot de passe pour l'utilisateur (ne pas mettre crochet, non obligatoire) |
| RENAME USER 'max'@'localhost' TO 'maxime'@'localhost'; | Modifie nom et accès hôte de l'utilisateur |
| SET PASSWORD FOR 'thibault'@'194.28.12.%' = PASSWORD('basket8'); | Modifie le mot de passe de l'utilisateur |
| DROP USER 'login'@'hote'; | Supprime un utilisateur |

**Privilèges utilisateurs :**

| **Listes des privilèges** | **Listes des privilèges** |
|---|---|
| CREATE TABLE | Création de tables |
| CREATE TEMPORARY TABLE | Création de tables temporaires |
| CREATE VIEW | Création de vues **(il faut également avoir le privilège SELECT sur les colonnes sélectionnées par la vue)** |
| ALTER | Modification de tables **(avec ALTER TABLE)** |
| DROP | Suppression de tables, vues et bases de données |
| CREATE ROUTINE | Création de procédures stockées **(et de fonctions stockées)** |
| ALTER ROUTINE | Modification et suppression de procédures stockées **(et fonctions stockées)** |
| EXECUTE | Exécution de procédures stockées **(et fonctions stockées)** |
| INDEX | Création et suppression d'index |
| TRIGGER | Création et suppression de triggers |
| LOCK TABLES | Verrouillage de tables **(sur lesquelles on a le privilège SELECT)** |
| CREATE USER | Gestion d'utilisateur **(commandes CREATE USER, DROP USER, RENAME USER et SET PASSWORD)** |
| **Niveau de privilège** | **Niveau de privilège** |
| *.* | Privilège global : **s'applique à toutes les bases de données**, à tous les objets. Un privilège de ce niveau sera stocké dans la table mysql.user. |
| * | **Si aucune base de données n'a été préalablement sélectionnée** (avec USE nom_bdd), c'est l'équivalent de *.* . Sinon, le privilège s'appliquera à **tous les objets de la base** de données qu'on utilise (et sera stocké dans la table mysql.db). |
| nom_bdd.* | Privilège de base de données : **s'applique à tous les objets de la base nom_bdd**(stocké dans mysql.db). |
| nom_bdd.nom_table | **Privilège de table** (stocké dans mysql.tables_priv). |
| nom_table | Privilège de table : **s'applique à la table nom_table de la base de données dans laquelle on se trouve**, **sélectionnée au préalable avec USE nom_bdd** (stocké dans mysql.tables_priv). |
| nom_bdd.nom_routine | **S'applique à la procédure (ou fonction) stockée nom_bdd.nom_routine**(privilège stocké dans mysql.procs_priv). |

Pour pouvoir ajouter un privilège à un utilisateur, il faut posséder le privilège GRANT OPTION (par default, l'utilisateur ROOT est le seul a le posséder)

| | |
|---|---|
| GRANT privilege [(liste_colonnes)] [, privilege [(liste_colonnes)], ...]<br>ON [type_objet] niveau_privilege<br>TO utilisateur [IDENTIFIED BY mot_de_passe]; | Cree un privilege<br><br>privilege : le privilège à accorder à l'utilisateur (SELECT, CREATE VIEW, EXECUTE,...) ;<br>(liste_colonnes) : facultatif - liste des colonnes auxquelles le privilège s'applique ;<br>niveau_privilege : niveau auquel le privilège s'applique (*.*, nom_bdd.nom_table,...) ;<br>type_objet : en cas de noms ambigus, il est possible de préciser à quoi se rapporte le niveau : TABLE ou PROCEDURE.<br><br>GRANT SELECT,<br>UPDATE (nom, sexe, commentaires),<br>DELETE,<br>INSERT<br>ON elevage.Animal<br>TO 'john'@'localhost' IDENTIFIED BY 'exemple2012'; |
| GRANT SELECT,<br>UPDATE (nom, sexe, commentaires),<br>DELETE,<br>INSERT<br>ON elevage.Animal<br>TO 'john'@'localhost' IDENTIFIED BY 'exemple2012'; | Cree le privilège de SELECT, UPDATE(les colonnes nom,sexe,commentaires seulement), DELETE, INSERT<br><br>Dans la table elevage.Animal<br>Pour 'john'@'localhost' et change son mot de passe par 'exemple2012' |
| GRANT ALL<br>ON elevage.Client<br>TO 'john'@'localhost'; | Donne TOUT les droits sur la table client a l'utilisateur 'john'@'localhost' |
| GRANT USAGE<br>ON *.*<br>TO 'john'@'localhost' IDENTIFIED BY 'test2012usage'; | Ne change pas les droit sur toutes les table à l'utilisateur 'john'@'localhost' mais change son mot de passe |
| GRANT SELECT, UPDATE, INSERT, DELETE, GRANT OPTION<br>ON elevage.*<br>TO 'joseph'@'localhost' IDENTIFIED BY 'ploc4'; | GRANT OPTION : permet de donner l'autorisation de donner des privilège (n'est pas mis avec all).<br>L'utilisateur peut donner seulement des privilèges qu'il possède déjà |
| REVOKE DELETE<br>ON elevage.Animal<br>FROM 'john'@'localhost'; | Supprime le privilège de DELETE de la table elevage.animal pour 'john'@'localhost' |
| WITH<br><br><br>MAX_QUERIES_PER_HOUR 50<br>MAX_CONNECTIONS_PER_HOUR 5;<br>MAX_UPDATES_PER_HOUR 50 | (Après from) remettre a 0 pour supprimer une limitation<br><br>Nombre de requête max par heure<br>Nombre de connexion max par heure<br>Nombre de UPDATE par heure |

**INFORMATIONS SUR LA BASE DE DONNÉES ET LES REQUÊTES :**

| | |
|---|---|
| SHOW CREATE TABLE Espece | Montre la requete ayant servi à cree la table **espece** |
| SHOW CHARACTER SET | Montre les sets de caractères (encodages) disponibles. |
| SHOW [FULL] COLUMNS FROM nom_table [FROM nom_bdd] | Liste les colonnes de la table précisée, ainsi que diverses informations (type, contraintes, …). Il est possible de préciser également le nom de la base de données. En ajoutant le mot-clé FULL, les informations affichées pour chaque colonne sont plus nombreuses. |
| SHOW DATABASES | Montre les bases de données sur lesquelles on possède des privilèges (ou toutes si l'on possède le privilège global SHOW DATABASES). |
| SHOW GRANTS [FOR utilisateur] | Liste les privilèges de l'utilisateur courant, ou de l'utilisateur précisé par la clause FOR optionnelle. |
| SHOW INDEX FROM nom_table [FROM nom_bdd] | Liste les index de la table désignée. Il est possible de préciser également le nom de la base de données. |
| SHOW PRIVILEGES | Liste les privilèges acceptés par le serveur MySQL (dépend de la version de MySQL). |
| SHOW PROCEDURE STATUS | Liste les procédures stockées. |
| SHOW [FULL] TABLES [FROM nom_bdd] | Liste les tables de la base de données courante, ou de la base de données désignée par la clause FROM. Si FULL est utilisé, une colonne apparaîtra en plus, précisant s'il s'agit d'une vraie table ou d'une vue. |
| SHOW TRIGGERS [FROM nom_bdd] | Liste les triggers de la base de données courante, ou de la base de données précisée grâce à la clause FROM. |
| SHOW [GLOBAL \| SESSION] VARIABLES | Liste les variables système de MySQL. Si GLOBAL est précisé, les valeurs des variables seront celles utilisées lors d'une nouvelle connexion au serveur. SiSESSION est utilisé (ou si l'on ne précise ni GLOBAL ni SESSION), les valeurs seront celles de la session courante. Plus d'informations sur les variables système seront données dans le prochain chapitre. |
| SHOW WARNINGS | Liste les avertissements générés par la dernière requête effectuée. |