**Erreur safe update :**

1. Go to Edit --> Preferences
2. Click "SQL Queries" tab and uncheck "Safe Updates" check box
3. Query --> Reconnect to Server // logout and then login
4. Now execute your sql query

**Les types de données**

|  |  |
| --- | --- |
| TINYINT  SMALLINT  MEDIUMINT  INT  BIGINT | -128 127  -32768 32767  -8388608 8388608  -2147483648 2147483648  -9223372036854775808 9223372036854775808 |
| UNSIGNED TINYINT  UNSIGNED SMALLINT  UNSIGNED MEDIUMINT  UNSIGNED INT  UNSIGNED BIGINT | 0 255  0 65535  0 16777215  0 4294967295  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  TEXT  MEDIUMTEXT  LONGTEXT | 2^8 octets  2^16 octets  2^24 octets  2^32 octets |
| espece ENUM('chat', 'chien', 'tortue')  espece SET('chat', 'chien', 'tortue') | Seul chat chien ou tortue peuvent être entré, sinon stockera une chaine vide par défault  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 (  colonne1 description\_colonne1,  colonne2 description\_colonne2,  colonne3 description\_colonne3,  [PRIMARY KEY (colonne\_clé\_primaire)]  )  [ENGINE= INNODB]; | Cree table Nom\_table si elle n'existe pas  Nom colonne1 type (id SMALLINT UNSIGNED NOT NULL AUTO\_INCREMENT,)  Nom colonne2 type (espece VARCHAR(40) NOT NULL,)  Nom colonne3 type (sexe CHAR(1),)  Clé primaire (PRIMARY KEY (id)) |
| DROP TABLE Nom\_table; | Supprime la table Nom\_table |

**MODIFICATION D'UNE TABLE**

|  |  |
| --- | --- |
| ALTER TABLE nom\_table  ADD COLUMN nom\_colonne description\_colonne; | Dans la table nom\_table  Ajoute colonne nom\_colonne |
| ALTER TABLE Commande  ADD CONSTRAINT fk\_client\_numero FOREIGN KEY (client) REFERENCES Client(numero); | Dans la table Commande  Ajoute nom\_foreign\_key la foreign key client qui fait reference a la table client(numero) |
| ALTER TABLE nom\_table  DROP COLUMN nom\_colonne; | Dans la table nom\_table  supprime colonne nom\_colonne |
| ALTER TABLE nom\_table  CHANGE ancien\_nom nouveau\_nom description\_colonne; | Dans la table nom\_table  Change nom de colonne par un autre (CHANGE nom prenom VARCHAR(10) NOT NULL;) |
| ALTER TABLE nom\_table  MODIFY nom\_colonne nouvelle\_description; | Dans la table nom\_table  Modifie la description (MODIFY nom VARCHAR(30) NOT NULL DEFAULT 'Blabla') |

**Insertion de données**

|  |  |
| --- | --- |
| INSERT INTO Animal (espece, sexe, date\_naissance)    VALUES ('tortue', 'F', '2009-08-03 05:12:00'); | Insert dans Animal et colonne espece, sexe, date\_naissance  Les valeurs respectivement 'tortue', 'F', '2009-08-03 05:12:00' |
| 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); | Insertion multiple |
| ~~INSERT INTO Animal~~  ~~SET nom='Bobo', espece='chien', sexe='M', 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  FROM Animal | Selectionne nom\_colonne  Dans la table nom\_table |
| UNION | Relie 2 select (à mettre à la fin du premier et avant la seconde) |
| SELECT DISTINCT espece  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%  WHERE espece LIKE \_  WHERE espece LIKE BINARY '%Lu%';  WHERE espece NOT LIKE %a% | Condition espèce contient c… (n'importe quel chaine après le c  Condition espèce contient 1seule caractère (P\_rl\_ pourrais etre egale a Parle – Perle – Perla)  La majuscule est obligatoire  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 :**  =  <  <=  >  >=  <> ou !=  <=> | **Comparaisons where :**  Égal  Inférieur  Inférieur ou égal  Supérieur  Supérieur ou égal  Différent  égal (valable pour NULL aussi) |
| **Plusieurs valeurs dans where**  AND  OR  XOR  NOT | **Plusieurs valeurs dans where**  ET  OU  OU exclusif  NON |
| **APRES LE WHERE SEULEMENT**  ORDER BY nom\_colonne  ORDER BY nom\_colonne DESC  ORDER BY nom\_colonne, nom\_colonne2; | **APRES LE WHERE SEULEMENT**  Trie par nom\_colonne croissant  Trie par nom\_colonne décroissant  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  WHERE critères; | Supprime dans nom\_table  Les éléments dont la condition est… |
| UPDATE Animal  SET sexe='F', nom='Pataude'  WHERE id=21; | Modifie dans la classe Animal  Sexe = F et nom = Pataude  A condition que l'ID soit 21 |

**Jointures**

|  |  |
| --- | --- |
| **APRES FROM SEULEMENT**  INNER JOIN Animal  ON Espece.id = Animal.espece\_id | **APRES FROM SEULEMENT**  Joins avec la table Animal  Ou l'id d'espece = l'id d'Animal |
| LEFT JOIN Animal  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)  FROM (  SELECT Animal.id, Animal.nom  FROM Animal  INNER JOIN Espece  ON Espece.id = Animal.espece\_id  WHERE sexe = 'F'  AND Espece.nom\_courant IN ('Tortue d''Hermann', 'Perroquet amazone')  ) AS tortues\_perroquets\_F; | Sélectionne la date de naissance la plus basse  Dans… (sous-requête dans from)  que l'on appellera Tortues\_perroquer\_F |
| SELECT id, nom, espece\_id  FROM Race  WHERE espece\_id < (  SELECT id  FROM Espece  WHERE nom\_courant = 'Tortue d''Hermann'); | Sélectionne id, nom, espèce  Dans la classe Race  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  Négatif : renvoie -1 zéro : renvoie 0 positif : renvoie +1 |
| ABS(nombre) | Renvoie la valeur absolu du nombre |
| MOD(nombre, division) | Retourne le reste de la division |
| BIT\_LENGTH(chaine) :  CHAR\_LENGTH(chaine) :  LENGTH(chaine) : | Retourne le nombre de bits de la chaîne.  Retourne le nombre de caractères de la chaîne.  retourne le nombre d'octets de la chaîne. |
| STRCMP(chaine1, chaine2) | Compare les deux chaînes passées en paramètres  Retourne :  0 si les chaînes sont les mêmes,  -1 si la première chaîne est classée avant dans l'ordre alphabétique  1 dans le cas contraire. |
| LPAD("chaineAModifier", taille, 'supplément caractère') 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.  Gauche  Droite |
| TRIM(LEADING FROM 'chaine')  TRIM(TRAILING FROM 'chaine')  TRIM(BOTH 'e' FROM 'chaine') | Supprime les caracteres inutile (par default les espaces)  Supprime les espaces à droite  Supprime les espaces à gauche  Supprime les e à gauche ET à droite |
| SUBSTRING('texte',3)  SUBSTRING('texte', 2, 3) | Récupère une partie de la chaine  A partir de la 3eme lettre incluse (xte)  De la 2eme lettre, de longueur 3 (ext) |
| INSTR(chaine, rech) | Retourne la position de la recherche dans chaine ("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)  Avec COUNT : écrit le nombre d'une espèce par ligne |
| select espece  from animal  group by espece; |  |
| select espece, count(espece)  from animal  group by espece; |  |
| GROUP BY nom\_courant, sexe WITH ROLLUP; | Sous-groupe de groupe exemple pour :  GROUP BY nom\_courant, sexe WITH ROLLUP   |  |  |  | | --- | --- | --- | | Chat | F | 9 | | Chat | M | 9 | | Chat | NULL | 18 | |
| 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**  **de '1001-01-01' à '9999-12-31'** |
| TIME | La valeur doit être de la forme **HH:MM:SS**  de '**-838:59:59**' à '**838:59:59**' |
| DATETIME | Doit être de la forme **AAAA-MM-JJ HH:MM:SS**  de '**1001-01-01 00:00:00**' à '**9999-12-31 23:59:59**' |
| TIMESTAMP | Doit être un **int** de forme **AAAAMMJJHHMMSS**  du **1er janvier 1970 00h00min00s** au **19 janvier 2038 03h14min07s.** |
| YEAR | Doit être un **int** de forme **AAAA**  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  %d Jour du mois (nombre à deux chiffres, de 00 à 31)  %e Jour du mois (nombre à un ou deux chiffres, de 0 à 31)  %w jour de la semaine (dimanche = 0,…, samedi = 6)  %W Nom du jour de la semaine  %a Nom du jour de la semaine en abrégé  %m Mois (nombre de deux chiffres, de 00 à 12)  %c Mois (nombre d'un ou deux chiffres, de 0 à 12)  %M Nom du mois  %b Nom du mois en abrégé  %y Année, sur deux chiffres  %Y Année, sur quatre chiffres  %r Heure complète, format 12h (hh:mm:ss AM/PM)  %T Heure complète, format 24h (hh:mm:ss)  %h Heure sur deux chiffres et sur 12 heures (de 00 à 12)  %H Heure sur deux chiffres et sur 24 heures (de 00 à 23)  %l Heure sur un ou deux chiffres, sur 12 heures (de 0 à 12)  %k Heure sur un ou deux chiffres, sur 24 heures (de 0 à 23)  %i Minutes (de 00 à 59)  %s Secondes (de 00 à 59)  %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)  **Si il y a une coupure avant un commit, les dernières insertion seront annulé** |
| COMMIT;  ROLLBACK; | Pour valider les requêtes  Pour annuler les requêtes |
| START TRANSACTION; | Permet de commencer une transaction sans avoir a mettre autocommit a 0  **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  **READ** : verrou de lecture. Les autres sessions pourront toujours lire les données des tables verrouillées, mais ne pourront plus les modifier.  **WRITE** : verrou d'écriture. Les autres sessions ne pourront plus ni lire ni modifier les données des tables verrouillées.  **AS** : si on n'en met pas, on ne pourra accédé seulement sans alias  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  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  **GLOBAL :** définit le niveau d'isolation pour toutes les sessions MySQL qui seront créées dans le futur. Les sessions existantes ne sont pas affectées.  **SESSION :** définit le niveau d'isolation pour la session courante.  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.  **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. Si l'on pense à bien utiliser les verrous là où c'est nécessaire, c'est un niveau d'isolation tout à fait suffisant.  **READ COMMITTED :**  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.  **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;  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  FROM 'SELECT \* FROM Adoption WHERE client\_id = ? AND animal\_id = ?'; | Nom de la requête prépare : select\_adoption  La requête qui sera faite quand on appellera select\_adoption |
| SET @colonne = 'nom';  SET @req\_animal = CONCAT('SELECT ', @colonne, ' FROM Animal WHERE id = ?');  PREPARE select\_col\_animal  FROM @req\_animal; | Syntaxe quand on utilise une variable,  penser à concaténé dans une autre variable toute le chaine.  Préparation de la requête  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 :**  PREPARE select\_adoption  FROM 'SELECT \* FROM Adoption WHERE client\_id = ? AND animal\_id = ?';  SET @id = 3;  SET @client = 2;  EXECUTE select\_adoption USING @client, @id; | **En une fois :**  Prépare une requête avec le nom select\_adoption  Les '?' sont les parties qui seront remplacé  Cree la variable id  Cree la variable client  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 |  CREATE PROCEDURE afficher\_races(PARAMETRES)  BEGIN  SELECT id, nom, espece\_id, prix  FROM Race;  END|  DELIMITER ; | Change le délimiteur ; par |  Créé une procédure du nom afficher\_race()  Début de la procédure  On peut faire autant de requête que l'on veut  Fin de la procédure  Rechange le délimiteur pour les requêtes normal  PARAMETRES : variable type (exemple : p\_espece\_id INT) |
| CALL afficher\_races(PARAMETRES A ENVOYER); | Appelle la procédure afficher\_races()  PARAMETRES : variable qu'on envoie (exemple : @espece\_id) |
| DELIMITER |  CREATE PROCEDURE compter\_races\_selon\_espece (IN p\_espece\_id INT, OUT p\_nb\_races INT)  BEGIN  SELECT COUNT(\*) INTO p\_nb\_races  FROM Race  WHERE espece\_id = p\_espece\_id;  END |  DELIMITER ; | IN : valeur entrant (utilisable que par la procédure)  OUT : valeur de sortie (la procédure modifie seulement)  INOUT : les deux au-dessus en même temps  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  instructions  ELSEIF autre\_condition THEN  instructions  ELSE  instructions  END IF; | SI  SINON SI  SINON  FIN SI |
| CASE v\_sexe  WHEN 'F' THEN -- Première possibilité  SELECT 'Je suis une femelle !' AS sexe;  WHEN 'M' THEN -- Deuxième possibilité  SELECT 'Je suis un mâle !' AS sexe;  ELSE  SELECT 'Je suis en plein questionnement existentiel...' AS sexe;  END CASE; | Variable v\_sexe  SI v\_SEXE = f  SINON SI v\_sexe = M  SINON  FIN SI |
| WHILE condition DO  END WHILE; | Boucle TANT QUE |
| super\_while: WHILE condition DO  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  WHEN sexe = 'M' THEN 'Je suis un mâle !'  WHEN sexe = 'F' THEN 'Je suis une femelle !'  ELSE 'Je suis en plein questionnement existentiel...'  END AS message  FROM Animal  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  FROM Animal  WHERE espece\_id = 5; | Condition if dans select : condition, valeur si vrai, valeur si faut |

**Curseur :**

|  |  |
| --- | --- |
| DECLARE curseur\_client CURSOR  FOR SELECT \*  FROM Client; | Déclare un curseur curseur\_client  Cree la requête select |
| OPEN nom\_curseur;  -- Parcours du curseur et instructions diverses  CLOSE nom\_curseur; | Ouverture/fermeture curseur |
| FETCH curs\_clients INTO v\_nom, v\_prenom;  SELECT CONCAT(v\_prenom, ' ', v\_nom) AS 'Premier client';  FETCH curs\_clients INTO v\_nom, v\_prenom;  SELECT CONCAT(v\_prenom, ' ', v\_nom) AS 'Second client'; | On récupère la première ligne  On assigne les valeurs récupérées à nos variables locales  On récupère la seconde ligne  on assigne les valeurs récupérées à nos variables locales |

**Triggers :**

|  |  |
| --- | --- |
| CREATE TRIGGER nom\_trigger moment\_trigger evenement\_trigger  ON nom\_table FOR EACH ROW  ~~corps\_trigger;~~  BEGIN  -- Instructions  END | | Déclaration d'un trigger  **nom\_trigger** : nom du trigger.  **moment\_trigger :** quand le trigger est déclenché.  **evenement\_trigger** : comment le trigger est déclenché.  **nom\_table** : à quelle table le trigger est attaché.  **FOR EACH ROW** : pour chaque ligne  **corps\_trigger** : contenu du trigger.  **Exemple** :  CREATE TRIGGER after\_insert\_animal AFTER INSERT  ON Animal FOR EACH ROW  **corps\_trigger**; |
| OLD.id  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.  **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  AS SELECT Animal.id, Animal.sexe, Animal.date\_naissance, Animal.nom, Animal.commentaires,  Animal.espece\_id, Animal.race\_id, Animal.mere\_id, Animal.pere\_id, Animal.disponible,  Espece.nom\_courant AS espece\_nom, Race.nom AS race\_nom  FROM Animal  INNER JOIN Espece ON Animal.espece\_id = Espece.id  LEFT JOIN Race ON Animal.race\_id = Race.id; | Cree une vue V\_Animal\_detail  Requête de la vue  OR REPLACE non obligatoire (ne pas mettre les crochets), remplace la vue si elle existe déjà. |
| SELECT \* FROM V\_Animal\_details;  SELECT id FROM V\_Animal\_details; | Sélectionnes toutes les données dans la vue V\_animal\_detail  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  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  Login : son login hote : l'ip ou adresse ou il peut se connecter (mettre % pour "n'importe où"  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**  CREATE TABLE  CREATE TEMPORARY TABLE  CREATE VIEW  ALTER  DROP  CREATE ROUTINE  ALTER ROUTINE  EXECUTE  INDEX  TRIGGER  LOCK TABLES  CREATE USER | **Listes des privilèges**  Création de tables  Création de tables temporaires  Création de vues **(il faut également avoir le privilège SELECT sur les colonnes sélectionnées par la vue)**  Modification de tables **(avec ALTER TABLE)**  Suppression de tables, vues et bases de données  Création de procédures stockées **(et de fonctions stockées)**  Modification et suppression de procédures stockées **(et fonctions stockées)**  Exécution de procédures stockées **(et fonctions stockées)**  Création et suppression d'index  Création et suppression de triggers  Verrouillage de tables **(sur lesquelles on a le privilège SELECT)**  Gestion d'utilisateur **(commandes CREATE USER, DROP USER, RENAME USER et SET PASSWORD)** |
| **Niveau de privilège**  \*.\*  \*  nom\_bdd.\*  nom\_bdd.nom\_table  nom\_table  nom\_bdd.nom\_routine | **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).  Privilège de base de données : **s'applique à tous les objets de la base nom\_bdd**(stocké dans mysql.db).  **Privilège de table** (stocké dans mysql.tables\_priv).  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).  **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)], ...]  ON [type\_objet] niveau\_privilege  TO utilisateur [IDENTIFIED BY mot\_de\_passe]; | Cree un privilege  privilege : le privilège à accorder à l'utilisateur (SELECT, CREATE VIEW, EXECUTE,…) ;  (liste\_colonnes) : facultatif - liste des colonnes auxquelles le privilège s'applique ;  niveau\_privilege : niveau auquel le privilège s'applique (\*.\*, nom\_bdd.nom\_table,…) ;  type\_objet : en cas de noms ambigus, il est possible de préciser à quoi se rapporte le niveau : TABLE ou PROCEDURE.  GRANT SELECT,  UPDATE (nom, sexe, commentaires),  DELETE,  INSERT  ON elevage.Animal  TO 'john'@'localhost' IDENTIFIED BY 'exemple2012'; |
| GRANT SELECT,  UPDATE (nom, sexe, commentaires),  DELETE,  INSERT  ON elevage.Animal  TO 'john'@'localhost' IDENTIFIED BY 'exemple2012'; | Cree le privilège de SELECT, UPDATE(les colonnes nom,sexe,commentaires seulement), DELETE, INSERT  Dans la table elevage.Animal  Pour 'john'@'localhost' et change son mot de passe par 'exemple2012' |
| GRANT ALL  ON elevage.Client  TO 'john'@'localhost'; | Donne TOUT les droits sur la table client a l'utilisateur 'john'@'localhost' |
| GRANT USAGE  ON \*.\*  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  ON elevage.\*  TO 'joseph'@'localhost' IDENTIFIED BY 'ploc4'; | GRANT OPTION : permet de donner l'autorisation de donner des privilège (n'est pas mis avec all). L'utilisateur peut donner seulement des privilèges qu'il possède déjà |
| REVOKE DELETE  ON elevage.Animal  FROM 'john'@'localhost'; | Supprime le privilège de DELETE de la table elevage.animal pour 'john'@'localhost' |
| WITH  MAX\_QUERIES\_PER\_HOUR 50  MAX\_CONNECTIONS\_PER\_HOUR 5;  MAX\_UPDATES\_PER\_HOUR 50 | (Après from) remettre a 0 pour supprimer une limitation  Nombre de requête max par heure  Nombre de connexion max par heure  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. |