# Memo MySQL

## Se connecter depuis cmd:

|  |  |
| --- | --- |
| **Renseigner le chemin path** | set PATH=%PATH%;C:\"Program Files"\MySQL\"MySQL Server 5.6"\bin |
| **Connexion mysql** | mysql -h localhost -u root -p |
| **Déconnexion mysql** | quit ou exit |
| **Définir l'encodage (OBLIGATOIRE à chaque connexion !)** | SET NAMES 'utf8'; **(latin1 gère mieux les accent...)** |

## Base

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Commentaire : -- (Metre un espace minimum apres !)** | Fin d'instruction : ; | | Retour a la ligne : \n | | Echapper un ' : mètre un autre ' devant : '' |
| **Ecrire du texte (pas de "") : 'texte'** | Echappement caractère : \ | | | Tabulation : \t | |
| **Créer un utilisateur avec tout privilège sur une base ici la base elevage** | | GRANT ALL PRIVILEGES ON elevage.\* TO 'pseudo'@'localhost' IDENTIFIED BY 'mot\_de\_passe'; | | | |

## Type de données

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Numérique**  **ENTIER**  **DECIMAUX**  **aproximatif** | | TINYINT : mini -128 / maxi : 127 / **1octet** | | | | | SMALLINT : mini -32728 / maxi : 32767 / **2octet** | | | | | | | | MEDIUMINT : mini -8388608 / maxi : 8388607 / **3octet** | |
| INT : mini -2147486448 / maxi : 214748647 / **4octet** | | | | | | | | | | BIGINT : mini -9223372036854775808 / maxi : 9223372036854775807 / **8octet** | | | | |
| UNSIGNED : décale le mini a 0. (pas de négatif) | | | | | | INT(4) ZEROFILL : Affichera toujours 4 chiffre minimum avec des zéro devant si plus petit | | | | | | | | |
| DECIMAL(5,3) : stocke des nombres à 5 chiffres avec 3 chiffres âpres la virgule | | | | | | | | | | | | NUMERIC(5,3) : similaire a DECIMAL(x,x) | | |
| FLOAT : avec ou sans paramètre(5,3) | | | | | DOUBLE : sans paramètre | | | | | | FLOAT et DOUBLE sont moins précis que DECIMAL et NUMERIC | | | |
| **Alphanumériques** | | | VARCHAR(x) : 255 caractère max, la taille s'adapte | | | | | | | CHAR(x) : 255 max mais prendra toujours la même taille (remplace par des espace le manque) | | | | | | |
| TINYTEXT : 255max | | | | | | | | TEXT : 655 35 max | | | | | |
| MEDIUMTEXT : 167 772 15 max | | | | | | | | LONGTEXT : 4 294 967 295 max | | | | | |
| **Binaire (ex : photos etc.)** | | | | BINARY(x) et VARBINARY(x) : x caractères max | | | | | | | TINYBLOG BLOB MEDIUMBLOB LONGBLOB avec les même limite que pour les TEXT | | | | | |
| **Cas particulier (ne pas utiliser sauf obligation)** | | | | | ENUM et SET : Permet de limiter les entré saisie par certain choix prédéfinie | | | | | | | | | | | |
| **Date** | DATE : sous format YYYY-MM-DD | | | | | DATETIME : sous format YYYY-MM-DD HH:MM:SS | | | | | | | | | | TIME : sous format HH:MM:SS |
| YEAR : sous format YYYY | | | | | | | | TIMESTAMP : stock sous forme d'un entier (date max : 19/01/2038) | | | | | | | |

## Creation de base de données

|  |  |  |
| --- | --- | --- |
| **Créer une nouvelle base de données** | CREATE DATABASE mabase CHARACTER SET 'utf8'; | |
| **Supprimer une base de données** | DROP DATABASE mabase; | |
| **Supprimer une bdd si elle existe** | DROP DATABASE IF EXISTS mabase; | |
| **Voir un message d'erreur avec un IF** | SHOW WARNINGS; | \W pour voir directement les warnings |
| **Sélectionner une base de données** | USE mabase; | |

## Creation et modification de tables

|  |  |  |  |
| --- | --- | --- | --- |
| **Création d'une table**  *NB : Par default NULL est autoriser*  *NB : On peut choisir une valeur par default en ajoutant dans la description de la colonne : DEFAULT 'valeur'* | 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; | | // créer la table Animal  // créer la colonne id avec Auto incrémentation  // créer la colonne espece  // créer la colonne sexe (NULL par defaut et autoriser)  // créer la colonne date\_naissance  // créer la colonne nom  // créer la colonne commentaires  // indique que la colonne id est la clé primaire  //indique le moteur utiliser (alternative : MyISAM : moins d'option) |
| **Liste les tables de la dbb** | SHOW TABLES; | | |
| **Listes les colonnes de la table** | DESCRIBE matable ; | | |
| **Supprimer une table** | DROP TABLE matable ; | | |
| **Ajouter une colonne** | ALTER TABLE matable  ADD COLUMN date\_insertion DATE NOT NULL; | // indique dans quel table on fait une modification  // ajoute une colonne date\_insertion | |
| **Supprimer une colonne** | ALTER TABLE matable  DROP COLUMN date\_insertion; | // COLUMN est facultatif (par default c'est une COLUMN) | |
| **Modification de colonne**  **Nom + description** | ALTER TABLE matable  CHANGE nom prenom VARCHAR(10) NOT NULL; | // renomme la colonne nom en prénom // Bien écrire toute la description sinon modifier! // pour pas changer nom écrire deux fois de suite | |
| **Modifier une colonne**  **Description uniquement** | ALTER TABLE matable  MODIFY id BIGINT NOT NULL AUTO\_INCREMENT; | //idem que CHANGE mais ne permet de modifier que la description de la colonne et pas le nom | |

## Insertion des données

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Insertion sans préciser les colonnes**  *NB : il faut donc connaitre l'ordre* | | INSERT INTO Animal  VALUES (NULL, 'chien', 'M', '2010-04-05 13:43:00', 'Rox', NULL); | | | // insère dans animal  // insère la ligne avec les valeurs (le premier NULL est l'id) | |
| **Insertion en précisant les colonnes** | | INSERT INTO Animal (espece, sexe, date\_naissance)  VALUES ('tortue', 'F', '2009-08-03 05:12:00'); | | | | |
| **Insertion multiple** | | 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); | | // 1er ligne  // 2nd ligne  // 3e ligne | | |
| **Exécuter une requête depuis un fichier** | | SOURCE Users\moi\dossier\monFichier.sql (donner le chemins complet) | | | | |
| **Et depuis un fichier en .csv** | LOAD DATA LOCAL INFILE 'personne.csv'  INTO TABLE Personne  FIELDS TERMINATED BY ';'  LINES TERMINATED BY '\n'  IGNORE 1 LINES  (nom,prenom,date\_naissance); | | // charge le fichier de la machine LOCAL  // dans la table Personne  // les colonne finisse par ;  // ou '\r\n' avec certain programme  // n'insere pas la premiere ligne  // nom des colonnes | | | Exemple de fichier personne.csv :  nom;prenom;date\_naissance  Charles;Myeur;1994-12-30  Bruno;Debor;1978-05-12  Mireille;Franelli;1990-08-23 |

## Selection des données

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Voir les entrée d'une table**  **SELECT...FROM...** | | SELECT colonne1, colonne2, ...  FROM nom\_table; | | | SELECT espece, nom, sexe  FROM Animal; | | | | | | SELECT \*  FROM Animal; | |
| **Voir les entré d'une table ou**  **SELECT...FROM...WHERE...** | | SELECT \*  FROM Animal  WHERE espece='chien'; | | | | SELECT \*  FROM Animal  WHERE espece <> 'chat'; | | | = égal || < inférieur || > supérieur  <= inférieur ou égal || >= supérieur ou égal  <> ou != différent || <=> égal (valable pour NULL aussi) | | | |
| AND et && = ET | OR ou || = OU | XOR = OU exclusif | NOT ou ! = NON | | | | | | | | | | | | |
| **Exemple combinées** | // chat et femelle en même temps  SELECT \*  FROM Animal  WHERE espece='chat'  && sexe='F'; | | | // Tortue et peroquet  SELECT \*  FROM Animal  WHERE espece='tortue'  || espece='perroquet'; | | | // toute femelle sauf chien  SELECT \*  FROM Animal  WHERE sexe='F'  AND ! espece='chien'; | | | | | // soit male soit perroquet (pas les deux)  SELECT \*  FROM Animal  WHERE sexe='M'  XOR espece='perroquet'; |
| **CAS particulier de NULL** | SELECT \*  FROM Animal  WHERE nom <=> NULL; | | | SELECT \*  FROM Animal  WHERE nom IS NULL; | | | | SELECT \*  FROM Animal  WHERE commentaires IS NOT NULL; | | | | |
| **Trie des donnees**  **...ORDER BY...**  **DESC = decroissant**  **ASC = croissant** | SELECT \*  FROM Animal  WHERE espece='chien'  ORDER BY date\_naissance; | | | SELECT \*  FROM Animal  WHERE espece='chien'  AND nom IS NOT NULL  ORDER BY nom DESC; | | | | //trie par espece et dans espece tri par date\_naissance  SELECT \*  FROM Animal  ORDER BY espece, date\_naissance; | | | | |
| **Eliminer les doublons** | SELECT DISTINCT espece  FROM Animal; | | | Si il y as 500 chiens dans la table, si on ne met pas DISTINCT on aura 500 lignes en retour a cette requête. Le fait de mettre DISTINCT affichera les espèce sans les répéter | | | | | | | | |
| **Limiter les résultat** | SELECT \*  FROM Animal  ORDER BY id  LIMIT 6 OFFSET 3; | | | | SELECT \*  FROM Animal  ORDER BY id  LIMIT 10; | | | | | // LIMIT = nombre de ligne affecter  // OFFSET = numéro de la ligne ça commence (facultatif, par défaut c'est 0) | | |
| **Recherche approximative**  **...WHERE...LIKE...**  **...WHERE...NOT LIKE...** | SELECT \*  FROM Animal  WHERE commentaires LIKE '%machin%'; | | | | SELECT \*  FROM Animal  WHERE commentaires LIKE 'gr\_s'; | | | | | LIKE BINARY = sensible à la casse  '%' = n'importe quel chaine de caractères  '\_' = un seul caractère | | |
| **Recherche dans intervalle**  **WHERE...BETWEEN...AND...**  **Ou encore ...NOT BETWEEN** | SELECT \*  FROM Animal  WHERE date\_naissance BETWEEN '2008-01-05' AND '2009-03-23'; | | | | | | | | | | | |
| **Recherché parmis une liste**  **...WHERE...IN...** | SELECT \*  FROM Animal  WHERE nom IN ('Moka', 'Bilba', 'Tortilla', 'Balou', 'Dana', 'Redbul', 'Gingko'); | | | | | | | // recherchera toute les lignes dont le nom est contenu dans la liste IN() | | | | |
| **Suprimer des entrées**  **DELETE FROM...WHERE...** | DELETE FROM Animal  WHERE nom = 'Zoulou'; | | // supprime dans la table animal  // les ligne dont le nom est 'Zoulou' || !! ATENTION **TOUJOURS UN WHERE SINON SUPR LA TABLE**!! | | | | | | | | | |
| **Modification des entrées**  **UPDATE...SET...WHERE...** | UPDATE Animal  SET sexe='F', nom='Pataude'  WHERE id=21; | | // dans la table animal  // change le sexe en 'F' et le nom en 'Pataude'  // dans la ligne dont l'id est 21 | | | | | | | | | |

## Sauvegarde

|  |  |  |
| --- | --- | --- |
| **Sauvegarde** | mysqldump -u user -p --opt nom\_de\_la\_base > sauvegarde.sql | // il faut être déconnecter de mysql avant tout |
| **Restaurer** | mysql nom\_base < chemin\_fichier\_de\_sauvegarde.sql | // il faut recréer la base avant d'insérer le contenu sauvegarder |

## Index (INDEX, UNIQUE, KEY peuvent être des index par la gauche, contrairement a FULLTEXT)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Créer une table avec index**  **Dans description colonne**  **KEY = index simple**  **UNIQUE = index unique** | CREATE TABLE nom\_table (  colonne1 INT KEY,  colonne2 VARCHAR(40) UNIQUE,  ); | | | | | //  //-- Crée un index simple sur colonne1  //-- Crée un index unique sur colonne2 / UNIQUE permet de s'assurer que jamais vous n’insérerez deux fois la même valeur (ou combinaison de valeurs) dans la table | | | |
| **Créer une table avec index A la suite des description**  **(permet de créer des index sur n colonnes)**  **INDEX = tout type d'index**  ***NB :*** *utilisez INDEX partout, sauf pour définir un index dans la description même de la colonne(utiliser alors KEY dans ce cas uniquement)* | 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),  INDEX ind\_date\_naissance (date\_naissance),  UNIQUE INDEX ind\_uni\_nom\_espece (nom, espece)  INDEX ind\_nom (nom(10))  )  ENGINE=INNODB; | | | | | | //crée la table animal  //  //  // NB : la création d'index accélère les recherche mais ralentie les requête d'insertion et de mise à jour car l'index est recalculer à chaque fois que les données sont modifier. Donc si un grand volume de données est à insérer après la création de la table, mettre les index seulement après insertion  //  -- index sur la date de naissance  -- Index sur le nom et l'espece  -- index sur le nom (le chiffre entre parenthèses étant le nombre de caractères pris en compte)  // ajouter ind\_ devant l'index pour pouvoir se repérer facilement | | |
| **Ajout index après création de la table** | ALTER TABLE nom\_table  ADD INDEX ind\_nom (nom); | | | | // sur la table nom\_table  // créer un index ind\_nom basé sur nom // on peut remplacer INDEX par UNIQUE ou FULLTEXT | | | | |
| **Alternative de syntaxe à ALTER TABLE** | CREATE INDEX ind\_nom  ON nom\_table (nom); | | | | //créer un index appeler ind\_nom  // sur la table nom\_table et prend en compte la colonne nom | | | | |
| **Complément sur UNIQUE**  **avec CONSTRAINT**  **= OPTIONELLE**  ***NB :*** *lorsque vous créez un index UNIQUE, vous pouvez explicitement créer une contrainte. C'est fait automatiquement bien sûr si vous ne le faites pas.* | CREATE TABLE nom\_table (  colonne1 INT NOT NULL,  colonne2 VARCHAR(40),  colonne3 TEXT,  CONSTRAINT [symbole\_contrainte] UNIQUE [INDEX] ind\_uni\_col2 (colonne2)  );  ALTER TABLE nom\_table  ADD CONSTRAINT [symbole\_contrainte] UNIQUE ind\_uni\_col2 (colonne2); | | | | | | | | // syntaxe en créant la table en même temps que l'index  // syntaxe en créant l'index après la table |
| **Supprimer un index** | | ALTER TABLE nom\_table  DROP INDEX nom\_index; | | | | | | *NB : on ne peut modifier un index, il faudra donc le supprimer puis le recréer avec les nouvelle valeur.* | |
| **Créer Index FULLTEXT apres création table**  ***NB****: uniquement sur CHAR, VARCHAR ou TEXT* | | CREATE FULLTEXT INDEX ind\_full\_titre  ON Livre (titre); | | | | | | // créer un index fulltext nommé ind\_full\_titre  // sur la colonne titre de la table livre | |
| **Exemple d'utilisation de FULLTEXT dans recherche**  **...WHERE MATCH ...AGAINST...** | SELECT \*  FROM Livre  WHERE MATCH(auteur)  AGAINST ('Terry'); | | //Sélectionne toute les colonne  //De la table livre  //Ou l'on cherche dans auteur  //Le mot terry | | | | | *NB : la recherche dois faire 4 lettre mini,*  *NB: est considéré comme un mot toute suite de caractères composée de lettres, de chiffres, tirets bas \_ et apostrophes '.*  *Ex : porte-clé sera considéré comme deux mot : porte et clé.* | |
| **Autre exemples FULLTEXT**  **Avec une colonne indiquant la pertinence** | SELECT \*, MATCH(titre, auteur) AGAINST ('Jules Verne Lune')  FROM Livre; | | | Affichera une colonne supplémentaire avec la pertinence des recherche effectué | | | | | |

|  |  |  |
| --- | --- | --- |
| **Autre exemples FULLTEXT**  **IN BOOLEAN MODE**  **Autre exemple**  **Autre exemple** | SELECT \*  FROM Livre  WHERE MATCH(titre)  AGAINST ('+bonheur -ogres' IN BOOLEAN MODE);  AGAINST ('"Lune à la Terre"' IN BOOLEAN MODE);  AGAINST ('petit\*' IN BOOLEAN MODE); | //Selectionne toute les colonne  //De la table livre  //Ou, dans la colonne titre  //Il y as le mot bonheur (+) et PAS le mot ogres (-)  // il y as exactement la suite de mot Lune à la Terre (entre " ") dans cette ordre  // recherche tout les mot commençant par petit (\*) ex : petit et petite |
| **Recherche avec extension de requête**  **WITH QUERY EXPANSION** | SELECT \*  FROM Livre  WHERE MATCH(titre, auteur)  AGAINST ('Daniel' WITH QUERY EXPANSION); | Effectuera une première requête avec le mot daniel  puis, avec les résultat retourner ré-effectue une requête avec tout les mot de toute les colonne du premier résultat |

## Clés primaires et étrangères

|  |  |  |  |
| --- | --- | --- | --- |
| **Création clé primaire**  **A la création de la table**  **PRIMARY KEY**  *NB: une table n'a qu'une seul clé primaire* | CREATE TABLE Animal (  id SMALLINT AUTO\_INCREMENT,  (...)  PRIMARY KEY (id)  )  ENGINE=InnoDB; | *NB: une clé primaire est comme un index NOT NULL et UNIQUE*  // créer une colonne auto incrémenter id  // indique que la colonne id est une clé primaire  De préference sur un type INT (unsigned) car une recherche va plus vite sur un chiffre qu'un texte | |
| **Création clé primaire**  **Apres la création de la table** | ALTER TABLE nom\_table  ADD [CONSTRAINT [symbole\_contrainte]] PRIMARY KEY (colonne\_pk1 [, colonne\_pk2, ...]); | // la ou les colonne ne doivent pas contenir NULL  // on peut créer des clé primaire composite (de plusieurs colonnes) | |
| **Suppression de la clé primaire** | ALTER TABLE nom\_table  DROP PRIMARY KEY | | |
| **Création de clé étrangère**  **FOREIGN KEY**  **REFERENCES** | 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  FOREIGN KEY (client)  REFERENCES Client(numero)  )  ENGINE=InnoDB; | | //NB: la clé étrangère et celle a lequel celle ci se réfère doit être de même type. ex si la clé de référence est INT UNSIGNED la clé étrangère doit aussi etre INT UNSIGNED  //  //  // on donne un nom à la clé étrangère  // colonne sur laquelle on crée la clé étrangère  // colonne sur laquelle se réfère la clé étrangère : la\_table(la\_colonne)  //  // |
| **Création de clé étrangère après la création de la table** | ALTER TABLE Commande  ADD CONSTRAINT fk\_client\_numero FOREIGN KEY (client) REFERENCES Client(numero); | | |
| **Suppression de la clé étrangère** | ALTER TABLE nom\_table  DROP FOREIGN KEY symbole\_contrainte | | |

## Jointures Interne et Externe

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Jointure interne**  **...INNER JOIN...ON...**  *=> Renvois la description de l'animal dont le nom est cartouche* | | SELECT Espece.description  FROM Espece  INNER JOIN Animal  ON Espece.id = Animal.espece\_id  WHERE Animal.nom = 'Cartouche'; | // sélectionne la colonne **description** de la table **espece**  // on travail sur la table **espece**  // qu'ont joins à la table **animal**  // la jointure se fait sur les colonnes **id** de la table **espece** et **espece\_id** de la table **animal**  // sélectionne les ligne qui possède **cartouche** dans la colonne **nom** de la table **animal** | | | |
| **Autre exemple**  *=> sélection du nom des animaux commençant par "Ch" ainsi que l'id et la description* | | SELECT Espece.id, Espece.description, Animal.nom  FROM Espece  INNER JOIN Animal  ON Espece.id = Animal.espece\_id  WHERE Animal.nom LIKE 'Ch%'; | | //Sélectionne **id** de **espece**, **description** de **espece** et **nom** de **animal**  // on travail sur la table **espece**  // qu'ont joins à la table **animal**  // joins les champs (clé primaire = clé etrangere)  // sélectionne les **nom** de **animal** commençant par **Ch%** (**%**=n'importe quoi apres ch) | | |
| **Autre exemple avec alias**  *=> même requête que la précédente mais avec des alias* | | SELECT e.id, e.description, a.nom  FROM Espece AS e  INNER JOIN Animal AS a  ON e.id = a.espece\_id  WHERE a.nom LIKE 'Ch%'; | | | // l'alias de la table espece est e  // l'alias de la table animal est a | |
| **Autre exemple avec alias**  **Sur le résultat renvoyer également**  *=> même requête que la précédente mais avec des alias qui désigne a la fois la colonne et la table* | | SELECT Espece.id AS id\_espece,  Espece.description AS description\_espece,  Animal.nom AS nom\_bestiole  FROM Espece  INNER JOIN Animal  ON Espece.id = Animal.espece\_id  WHERE Animal.nom LIKE 'Ch%'; | | | // la colonne **id** de la table **espece** est renommer en **id\_espece**  *NB: le résultat renvoyé, dans le nom des colonne sera remplacer par l'alias.*  *Ex: en définissant l'alias comme ci-contre la colonne* ***description*** *s'appellera* ***description\_espece*** | |
| **Jointure Externe**  **LEFT JOIN...ON...**  NB: une jointure externe sélectionne également les ligne qui n'ont pas de correspondance | | SELECT Animal.nom AS nom\_animal, Race.nom AS race  FROM Animal  LEFT JOIN Race  ON Animal.race\_id = Race.id  WHERE Animal.espece\_id = 2  AND Animal.nom LIKE 'C%'  ORDER BY Race.nom, Animal.nom; | | | //  // **animal** est la table de gauche  // **race** est la table de droite / on joint avec la table de gauche donc **animal**  // on indique la correspondance des clé  // on récupère **espece\_id** de la table animal dont l'**id** est **2**  //ET dont le nom commence par **C**  // ordonné par race puis par nom | |
| **Jointure Externe**  **RIGHT JOIN...ON...** | | SELECT Animal.nom AS nom\_animal, Race.nom AS race  FROM Animal  RIGHT JOIN Race  ON Animal.race\_id = Race.id  WHERE Race.espece\_id = 2  ORDER BY Race.nom, Animal.nom; | | | *NB: 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.* | |
| **Alternative**  **USING**  **NATURAL JOIN** | SELECT \*  FROM table1  INNER JOIN table2 USING (colonneJ);  -- équivalent à  SELECT \*  FROM table1  INNER JOIN table2 ON Table1.colonneJ = table2.colonneJ; | | | | SELECT \*  FROM table1  NATURAL JOIN table2;  -- équivalent à  SELECT \*  FROM table1  INNER JOIN table2  ON table1.B = table2.B; | SELECT \*  FROM table1  NATURAL JOIN table3;  -- EST ÉQUIVALENT À  SELECT \*  FROM table1  INNER JOIN table3  ON table1.A = table3.A AND table1.C = table3.C; |

## Sous-requête (en général possible avec une jointure)

|  |  |  |  |
| --- | --- | --- | --- |
| **Exemple de sous requête**  **Dans FROM** | SELECT MIN(date\_naissance)  FROM (  SELECT Animal.id, Animal.sexe, Animal.date\_naissance, Animal.nom, Animal.espece\_id  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 le plus petit date\_naissance  // Début de la sous requête toujours entre ()  // dont le nom courant est ('ça' ou 'ça')  //Donne un alias a la ss requête **OBLIGATOIRE** |
| **Sous requête renvoyant une valeur**  **Dans WHERE** | SELECT id, sexe, nom, commentaires, espece\_id, race\_id  FROM Animal  WHERE race\_id =  (SELECT id FROM Race WHERE nom = 'Berger Allemand'); | | SELECT id, nom, espece\_id  FROM Race  WHERE espece\_id = (  SELECT MIN(id) FROM Espece ); |
| **Sous requête renvoyant une ligne**  *NB: seul les opérateur = et != ou <> sont possible*  *NB: La requête ne doit renvoyer qu'une seul ligne* | SELECT id, sexe, nom, espece\_id, race\_id  FROM Animal  WHERE (id, race\_id) = (  SELECT id, espece\_id  FROM Race  WHERE id = 7); | | // ce qu'il y as entre (x,y ) après le WHERE doit être ce qu'il y as, et dans l'ordre du SELECT x,y de la sous requête |
| **Sous requête renvoyant une seul colonne de résultat avec IN**  *NB : NOT IN fonctionne aussi* | SELECT id, nom, espece\_id  FROM Animal  WHERE espece\_id IN (  SELECT id  FROM Espece  WHERE nom\_courant IN ('Tortue d''Hermann', 'Perroquet amazone') ); | | Ainsi, la requête principale sélectionnera les lignes qui ont un **espece\_id** parmi ceux renvoyés par la sous-requête |
| **Sous requête renvoyant une seul colonne de résultat avec ANY**  *NB : syntaxe alternative : SOME* | SELECT \*  FROM Animal  WHERE espece\_id < ANY (  SELECT id  FROM Espece  WHERE nom\_courant IN ('Tortue d''Hermann', 'Perroquet amazone') ); | | La requête suivante signifie donc "Sélectionne les lignes de la table Animal, dont l'**espece\_id** est inférieur à **au moins une** des valeurs sélectionnées dans la sous-requête". |
| **Sous requête renvoyant une seul colonne de résultat avec ALL** | SELECT \*  FROM Animal  WHERE espece\_id < ALL (  SELECT id  FROM Espece  WHERE nom\_courant IN ('Tortue d''Hermann', 'Perroquet amazone') ); | | Par contre, si vous utilisez ALL plutôt que ANY, cela signifiera "Sélectionne les lignes de la table Animal, dont l'**espece\_id** est inférieur à **toutes** les valeurs sélectionnées dans la sous-requête". |
| **Sous requête avec**  **EXISTS** | SELECT id, nom, espece\_id  FROM Race  WHERE EXISTS (SELECT \* FROM Animal WHERE nom = 'Balou'); | Une condition avec EXISTS sera vraie (et donc la requête renverra quelque chose) si la sous-requête correspondante renvoie au moins une ligne et le contraire avec NOT EXISTS | |
| **Sous requête avec**  **NOT EXISTS** | SELECT \* FROM Race  WHERE NOT EXISTS  (SELECT \* FROM Animal WHERE Animal.race\_id = Race.id); | je veux sélectionner toutes les races dont on ne possède aucun animal | |

## Jointure et sous requête : modification des données

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Insertion avec**  **INSERT INTO... SELECT...FROM**  **...WHERE** | | INSERT INTO Animal  (nom, sexe, date\_naissance, race\_id, espece\_id)  SELECT 'Yoda', 'M', '2010-11-09', id AS race\_id, espece\_id  FROM Race  WHERE nom = 'Maine coon'; | | | // insert dans la table animal  // Les champs (x,y,z,...) !!!! attention a l'ordre entre colonnes et après le SELECT  // après le SELECT placer les champs que l'on veut insérer entre guillemet simple ''  // Les champs **race\_id** et **espece\_id** nous sont inconnue, ont fait donc une requête dans le FROM de la table ou sont les attribut et dans WHERE ce qu'on cherche | | | |
| **Mise a jour de table avec sous requête dans**  **UPDATE...** | | | UPDATE Animal SET commentaires = 'Coco veut un gâteau !'  WHERE espece\_id =  (SELECT id FROM Espece WHERE nom\_courant LIKE 'Perroquet%'); | | | | | // met a jour la colonne commentaire de la table **animal**  // ou l'**id\_espece** est egal à  // prend l'**id** de dont le nom courant commence par **Peroquet** |
| **Limitation**  **UPDATE...**  *NB: les commandes suivante renverront une erreur !!* | | | | UPDATE Animal SET race\_id =  (SELECT race\_id FROM Animal  WHERE nom = 'Cawette' AND espece\_id = 2)  WHERE nom = 'Callune'; | | | // La sous-requête utilise la table Animal, or vous cherchez à modifier le contenu de celle-ci.  ***NB: on ne peut pas modifier un élément d'une table que l'on utilise dans une sous-requête.*** | |
| **Jointure avec**  **UPDATE...** | UPDATE Animal  INNER JOIN Espece  ON Animal.espece\_id = Espece.id  SET Animal.commentaires = Espece.description  WHERE Animal.commentaires IS NULL  AND Espece.nom\_courant IN ('Perroquet amazone', 'Tortue d''Hermann'); | | | | | // on met a jour la table **animal**  // qu'ont joint a la table **espece**  // corespondance des clé primaire => etrangere  // les **commentaires** d'**animal** sont remplacer par la **description** de **espece**  // dans tout les champs **commentaires** qui sont NULL  // ET dont le **nom\_courant** de **espece** est dans (**x,y**) | | |
| **Suppression avec sous-requête DELETE FROM...**  **WHERE (SELECT...)** | | | DELETE FROM Animal  WHERE nom = 'Carabistouille' AND espece\_id =  (SELECT id FROM Espece WHERE nom\_courant = 'Chat'); | | | // supprime dans la table **animal**  // la ligne dont le **nom** est **Carabistouille** ET dont l'**id** est  // l'**id** de la table **espece** avec le **nom\_courant** qui est **Chat** | | |
| **Suppression avec jointure DELETE FROM...**  **WHERE (SELECT...)** | | | DELETE Animal  FROM Animal  INNER JOIN Espece ON Animal.espece\_id = Espece.id  WHERE Animal.nom = 'Carabistouille'  AND Espece.nom\_courant = 'Chat'; | | | Vous remarquez une petite différence avec la syntaxe "classique" de DELETE (sans jointure) : je précise le nom de la table dans laquelle les lignes doivent être supprimées juste après le DELETE.  En effet, comme on utilise plusieurs tables, cette précision est obligatoire. Ici, on ne supprimera que les lignes d'**Animal** correspondantes. | | |

## Union de plusieurs requêtes

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Syntax UNION** | | SELECT ...  UNION  SELECT ... | | 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. Le nombre de requêtes qu'il est possible d'unir est illimité. | | |
| **Règles UNION** | | Il est absolument indispensable que toutes les requêtes unies renvoient le même nombre de colonnes.  Il faut que les colonnes sélectionné soit de même type sur les deux requêtes  Le nom des colonnes est celui donnée à la première requête  L'ordre des colonne doit être le même dans toute les requêtes  S'il y as des doublons de renvoyé ceux-ci sont supprimer | | | | |
| **UNION ALL...** | | SELECT \* FROM Espece  UNION ALL  SELECT \* FROM Espece; | Chaque résultat n’apparaît qu'une seule fois avec UNION. Pour la simple et bonne raison que lorsque vous faites UNION, les doublons sont effacés. En fait, UNION est équivalent à UNION DISTINCT. Si vous voulez conserver les doublons, vous devez utiliser UNION ALL. | | | |
| **LIMIT dans la 1er requêtes avec UNION** | | SELECT id, nom, 'Race' AS table\_origine FROM Race LIMIT 3  UNION  SELECT id, nom\_latin, 'Espèce' AS table\_origine FROM Espece; | | | // affiche les 3 premier résultat de la table **race** avec les champs **id,nom** et **table\_origine** qui aura la valeur '**race**' | |
| **LIMIT dans la dernière requêtes avec UNION** | | SELECT id, nom, 'Race' AS table\_origine FROM Race  UNION  SELECT id, nom\_latin, 'Espèce' AS table\_origine FROM Espece LIMIT 2; | | | // **ATTENTION** ici le LIMIT est appliquer à l'ensemble des résultat | |
| **LIMIT dans une requête autre que la première avec UNION** | | SELECT id, nom, 'Race' AS table\_origine FROM Race  UNION  (SELECT id, nom\_latin, 'Espèce' AS table\_origine FROM Espece LIMIT 2); | | | Afin de préciser que la LIMIT ne s'applique qu'a la requête en cours il faut la mettre entre parenthèses ( ) | |
| **ORDER BY dans requête UNION** | | SELECT id, nom, 'Race' AS table\_origine FROM Race  UNION  SELECT id, nom\_latin, 'Espèce' AS table\_origine FROM Espece  ORDER BY nom DESC; | | | Il n'est possible de faire un ORDER BY uniquement sur l'ensemble de la requête, autrement dit, on ne pourra pas faire ORDER BY sur les requête intermédiaire (sauf exeption) | |
| *NB : il faut bien mettre ici ORDER BY nom, et surtout pas ORDER BY Race.nom ou ORDER BY Espece.nom\_latin. En effet, l'ORDER BY agit sur l'ensemble de la requête, donc en quelque sorte, sur une table intermédiaire composée des résultats des deux requêtes unies. Cette table n'est pas nommée, et ne possède que trois colonnes :* id, nom *et* table\_origine *(définies par la première clause SELECT rencontrée).* | | | | | | |
| **Exception pour ORDER BY sur requête intermédiaire** | (SELECT id, nom, 'Race' AS table\_origine FROM Race ORDER BY nom DESC LIMIT 6)  UNION  (SELECT id, nom\_latin, 'Espèce' AS table\_origine FROM Espece LIMIT 3); | | | | | Dans une requête intermédiaire, il est possible d'utiliser un ORDER BY mais uniquement combiné à un LIMIT. Cela permettra de restreindre les résultats voulus (les X premiers dans l'ordre défini par l'ORDER BY par exemple). |

## Options des clés étrangères

|  |  |  |
| --- | --- | --- |
| **Syntaxe**  **Suppression clé étrangère**  **ON DELETE** | ALTER TABLE nom\_table  ADD [CONSTRAINT fk\_col\_ref]  FOREIGN KEY (colonne)  REFERENCES table\_ref(col\_ref)  ON DELETE {RESTRICT | NO ACTION | SET NULL | CASCADE}; | //L'option ON DELETE permet de changer la manière dont la clé étrangère gère la suppression d'une référence.  //Si l'option n'est pas définie et que l'on tente de suprimmer une clé étrangère un message d'erreur nous empêchera de la supprimer si la moindre ligne y fait référence |
| **RESTRICT** | RESTRICT est le comportement par défaut. Si l'on essaye de supprimer une valeur référencée par une clé étrangère, l'action est avortée et on obtient une erreur.NO ACTION a exactement le même effet (en MySQL uniquement, dans d'autre SGBD RESTRICT est plus strict). | |
| **SET NULL**  **Exemple** | ALTER TABLE Animal  ADD CONSTRAINT fk\_race\_id FOREIGN KEY (race\_id) REFERENCES Race(id) ON DELETE SET NULL; | Si on choisit SET NULL, alors tout simplement, NULL est substitué aux valeurs dont la référence est supprimée. |
| **CASCADE** | Ce dernier comportement est le plus risqué . En effet, cela supprime purement et simplement toutes les lignes qui référençaient la valeur supprimée ! | |
| **Syntaxe**  **Modification clé étrangère**  **ON UPDATE**  **Exemple** | ALTER TABLE Animal  ADD CONSTRAINT fk\_race\_id FOREIGN KEY (race\_id) REFERENCES Race(id)  ON DELETE SET NULL  ON UPDATE CASCADE; | Les quatre comportements possibles sont exactement les mêmes que pour la suppression.  CASCADE signifie que l'événement est répété sur les tables qui référencent la valeur.  **IMPORTANT** les modification sur les clé auto-incrémenter sont à proscrire |

## Violation de contrainte d'unicité

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Insertion avec**  **INSERT IGNORE INTO....** | INSERT IGNORE INTO Espece (nom\_courant, nom\_latin, description)  VALUES ('Chien en peluche', 'Canis canis', 'Tout doux'); | | | Si on a une contrainte d'unicité qui devrais renvoyer un message d'erreur a cette requête, celui-ci sera ignorer (pas afficher seulement) | |
| **Insertion avec**  **UPDATE IGNORE....** | UPDATE IGNORE Espece SET nom\_latin = 'Canis canis'  WHERE nom\_courant = 'Chat'; | | | Idem que la précédente mais avec un UPDATE : les message d'erreur sont ignorer et la requête n'est pas exécuter (car contrainte d'unicité!) | |
| **LOAD DATA INFILE... IGNORE** | LOAD DATA [LOCAL] INFILE 'nom\_fichier' IGNORE  INTO TABLE nom\_table  (...) | La même option est disponible avec la commande LOAD DATA INFILE, ce qui est plutôt pratique si vous voulez éviter de devoir traficoter votre fichier suite à une insertion partielle due à une ligne qui ne respecte pas une contrainte d'unicité. | | | |
| **Remplacer une ligne**  **REPLACE INTO...** | REPLACE INTO Animal (sexe, nom, date\_naissance, espece\_id)  VALUES ('F', 'Spoutnik', '2010-08-06 15:05:00', 3); | | Dans le cas d'une insertion qui enfreint une contrainte d'unicité, REPLACE ne va ni renvoyer une erreur, ni ignorer l'insertion.  REPLACE va supprimer l'ancienne ligne et créer la nouvelle. | | |
| **IMPORTANT** toutes les lignes qui empêcherais l'insertion à cause de contraintes d'unicité seront supprimées. | | | | | |
| **LOAD DATA INFILE... REPLACE** | LOAD DATA [LOCAL] INFILE 'nom\_fichier' REPLACE  INTO TABLE nom\_table (...) | | | | *NB: IGNORE et REPLACE ne peuvent être utilisé en même temps, autrement dit il faut choisir !* |
| **Modifier une ligne**  **ON DUPLICATE KEY UPDATE** | INSERT INTO Animal (sexe, date\_naissance, espece\_id, nom, mere\_id)  VALUES ('M', '2010-05-27 11:38:00', 3, 'Spoutnik', 52)  ON DUPLICATE KEY UPDATE mere\_id = 52; | | | | 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). |
| **IMPORTANT** Dans le cas où plusieurs lignes seraient impliquées, seule une de ces lignes est modifiée (et impossible de prédire laquelle). Il faut donc **à tout prix** éviter d'utiliser cette clause quand plusieurs contraintes d'unicité pourraient être violées par l'insertion. | | | | | |

## Fonction général (pas d'espace entre le nom de la fonction et () )

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | SELECT VERSION(); | Affiche la version actuel de MySQL | |
| **utilisateur** | SELECT CURRENT\_USER(), USER(); | CURRENT\_USER() renvois l'user utilisé a authentification, USER() renvois l'user spécifié a authentification; | |
| **Dernier id** | LAST\_INSERT\_ID() | Renvois le dernier id crée par auto-incrémentation | |
| **Nombre de ligne que la dernière requête à renvoyé** | SELECT SQL\_CALC\_FOUND\_ROWS id, nom, espece\_id,  FROM Race  LIMIT 3;  SELECT FOUND\_ROWS() AS avec\_option; | | Ici nous effectuons une requête avec une clause LIMIT qui renverra 3 résultat.  Avec SQL\_CALC\_FOUND\_ROW() après le SELECT et FOUND\_ROWS() dans une autre commande le resultat sera celui de tout les résultat qui aurais été renvoyé si le LIMIT n'était pas présent. |
| **Convertir un type de données** | SELECT CAST('870303' AS DATE); | Convertie la chaine '870303' en DATE  Type possible : BINARY, CHAR, DATE, DATETIME, UNSIGNED, SIGNED, DECIMAL // SIGNED et UNSIGNED = INT | |

## Fonction Scalaires (agit sur chaque ligne indépendamment)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Arrondir nombre** | | | SELECT CEIL(3.2);  SELECT FLOOR(3.2);  SELECT ROUND(3.22, 1);  SELECT TRUNCATE(3.22, 1); | Arrondit au nombre entier supérieur (4 ici)  Arrondit au nombre entier inférieur (3 ici)  ROUND(n, d) Arrondit au nombre a d décimales le plus proche (3.2 ici)  Arrondit en enlevant les décimal en trop (3.2 ici) | | | |
| **Exposants** | | | SELECT POW(2, 5) | N exposant e : 2^5 (32 ici) | | | |
| **Racines** | | | SELECT SQRT(4); | Donne la racine carrée (ici 2) | | | |
| **Nombre aléatoire** | | | SELECT RAND(); | Donne un nombre aléatoire entre 0 et 1 (par exemple 0.08978611469155748) | | | |
| **Signe du nombre** | | | SELECT SIGN(-43); | Renvois -1 si le nombre est négatif, 0 si égal à 0 et 1 si positif | | | |
| **Valeur absolu** | | | SELECT ABS(-43); | Renvois la valeur absolu du nombre (sans le signe)(ici : 43) | | | |
| **Modulo** | | | SELECT MOD(56, 10); | Renvois le modulo (le reste de la division) (ici 6) | | | |
| **Longueur chaine** | | | SELECT BIT\_LENGTH('élevage');  SELECT CHAR\_LENGTH('élevage');  SELECT LENGTH('élevage'); | Renvois le nombre de bits de la chaine  Renvois le nombre de caractères de la chaine  Renvois le nombre d'octets de la chaine | | | |
| **Comparer chaine** | | | SELECT STRCMP('chaine', 'texte'); | renvois 0 si les chaînes sont les mêmes, -1 si la première chaîne est classée avant dans l'ordre alphabétique et 1 dans le cas contraire. | | | |
| **Répéter chaine** | | | SELECT REPEAT('Ok ', 3); | Renvois la chaine répéter x fois REPEAT('chaine', x); | | | |
| **Compléter / réduire chaine** | | | SELECT LPAD('texte', 20, \*);  SELECT RPAD(texte, long, caract); | Les longueurs de chaine doivent faire x caractère et sont complété par le caractère (ici \*) a gauche  Idem mais à droite | | | |
| **Retirer caractères inutile** | | | SELECT TRIM(LEADING 't' FROM ' Tralala ') ;  SELECT TRIM(TRAILING 'a' FROM ' Tralala ') ;  SELECT TRIM(BOTH 'e' FROM 'eeeBouHeee'); | | | | Retire les caractères a l'avant de la chaine (ici retire les t devant la chaine 'tralala' => 'ralala'  Retire les caractères a l'arrière de la chaine (ici retire les a devant la chaine 'tralala' => 'tralal'  Retire les caractères a l'arrière et l'avant de la chaine |
| **Récupérer une sous-chaine** | | | SELECT SUBSTRING('texte' ,2, 3); | Renvois une partie de la chaine avec 1er paramètre : chaine / 2em : début de la position / 3em : pendant combien de caractères (facultatif) | | | |
| **Recherche de la position de la 1er occurrence** | | | SELECT INSTR('tralala', 'la');  SELECT POSITION('la' IN 'tralala');  SELECT LOCATE('la', 'tralala') ; | INSTR(chaine, recherche) //Les 3 fonctions retourne 0 si le mot n'est pas trouver  POSITION(recherche IN chaine) //la première position est donc 1 et non 0 comme souvent  LOCATE(recherche, chaine) // peut prendre un param supplémentaire a la fin : chercher à partir de x carac | | | |
| **Changer la casse des chaines** | | | SELECT LOWER('AhAh');  SELECT UPPER('AhAh'); | Renvois la chaine en minuscule  Renvois la chaine en majuscule | | | |
| **Récupérer un nombre de caract** | | | SELECT LEFT('123456789', 5), SELECT RIGHT('123456789', 5); | Ici Renvois les 5 premier nombre de la chaine  Ici Renvois les 5 dernier nombre de la chaine | | | |
| **Inverser chaine** | | | SELECT REVERSE('abcde'); | Renvois ici : edcba | | | |
| **remplacer** | INSERT(chaine, pos, long, nouvCaract)  REPLACE(chaine, ancCaract, nouvCaract) | | | le paramètre chaine est la chaîne de caractères dont on veut remplacer une partie, pos est la position du premier caractère à remplacer, long le nombre de caractères à remplacer, et nouvCaract est la chaîne de caractères qui viendra remplacer la portion de chaine choisie.  tous les caractères (ou sous-chaînes) ancCaract seront remplacés par nouvCaract. | | | |
| **concaténation** | | CONCAT(chaine1, chaine2,…)  CONCAT\_WS(separateur, chaine1, chaine2) | | | renvoie simplement une chaîne de caractères, résultat de la concaténation de toutes les chaînes passées en paramètres.  même chose que CONCAT(), sauf que la première chaîne passée sera utilisée comme séparateur, donc placée entre chacune des autres chaînes passées en paramètres. | | |
| **Code ascii** | | ASCII(chaine)  CHAR(ascii1, ascii2,…) | | | Prend comme parametre une chaine et renvois le code ascii correspondant  Prend comme parametre la ou les valeurs et revois les lettre correspondant au code ascii | | |
| **recherche** | | FIELD(rech, chaine1, chaine2, chaine3,…) | | | Recherche parmi les chaines... renvois 0 si pas trouver, 1 si trouver dans le 1er mot etc... | | |
| **Imbrication** | | SELECT REPLACE(REPLACE(nom, 'a', '@'), 'e', '3') | | | | Donc, vous effectuez un premier remplacement, en changeant les "a" par les "@" : REPLACE(nom, 'a', '@'). Ensuite, **sur la chaîne résultant de ce premier remplacement**, vous effectuez le second | |

## Fonction d'agrégation (regroupe les différentes lignes : renvois un résultat unique pour toute les lignes)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de ligne** | SELECT COUNT(\*)  SELECT COUNT(colonne)  SELECT COUNT(DISTINCT colonne) | | Renvois le nombre de ligne sélectionnées par la requête (attention NULL compte également)  seules les lignes dont la valeur de colonne n'est pas NULL seront prises en compte.  Idem que la précédente mais supprime les doublons | |
| **Maximum**  **Minimum** | SELECT MIN(prix)  SELECT MAX(prix) | | Renvois le nombre le plus petit //Les deux fonctionne avec les dates, chaine caractères  Renvois le nombre le plus grand // | |
| **Somme**  **Moyenne** | SELECT SUM(prix)  SELECT AVG(prix) | | Renvois la somme de toute les valeurs  Renvois la moyenne des sommes | |
| **Concaténation** | SELECT SUM(prix), GROUP\_CONCAT(nom\_courant)  FROM Espece; | | | GROUP\_CONCAT(colonne) renvois une colonne de résultat suplémentaire indiquant toute les lignes concerné |
| **Syntaxe**  **Concaténation** | 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 : chaîne de caractères à utiliser pour séparer les différentes valeurs. Par défaut, c'est une virgule. | | |

## Regroupement

|  |  |  |
| --- | --- | --- |
| **GROUP BY** | SELECT COUNT(\*) AS nb\_animaux  FROM Animal  GROUP BY espece\_id; | Compte toute les ligne  De la table **animal**  Groupé par **espece\_id** // se place après le WHERE s'il y en as un |
| **Autre Exemple**  **GROUP BY**  **Et jointure** | SELECT nom\_courant, COUNT(\*) AS nb\_animaux  FROM Animal  INNER JOIN Espece ON Animal.espece\_id = Espece.id  GROUP BY nom\_courant; | Dans le SELECT ou encore ORDER BY doit se trouver uniquement des nom de colonne utiliser dans le GROUP BY |
| **Multicritères GROUP BY** | SELECT nom\_courant, sexe, COUNT(\*) AS nb\_animaux  FROM Animal  INNER JOIN Espece ON Espece.id = Animal.espece\_id  GROUP BY nom\_courant, sexe; | Sélectionne **nom\_courant**, **sexe**, compte toute les ligne  De la table **animal**  Jointure de la table **espece** ou **espece.id** = **animal\_espece.id**  Groupé par **nom\_courant** puis par sexe (l'ordre est important) |
| **Super-agrégats**  **WITH ROLLUP** | SELECT nom\_courant, sexe, COUNT(\*) as nb\_animaux  FROM Animal  INNER JOIN Espece ON Espece.id = Animal.espece\_id  WHERE sexe IS NOT NULL  GROUP BY nom\_courant, sexe WITH ROLLUP; | L'option WITH ROLLUP placer après le GROUP BY permet d'indiquer le nombre total de ligne renvoyé pour chaque colonne.  Par exemple ici si le résultat renvois 9 chat male et 9 chat femelle, une ligne supplémentaire indiquera le nombre total de chat soit 18 et ainsi de suite.  Et enfin une dernière ligne indiquant tout le nombre de ligne de l'ensemble |
| **COALESCE avec WITH ROLLUP** | 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; | Permet de modifier le nom renvoyé par le GROUP BY (qui par défaut, est NULL)  Dans l'exemple ci contre dans le groupement par **nom\_courant**, sera afficher le nom courant et a la fin le **'total'**. Attention de bien être sur que la requête ne renvois pas des NULL, sinon NULL sera remplacer par total partout  COALESCE(colonne1, 'Nom\_renvoyé') |
| **Condition avec 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; | On ne peut pas faire de condition sur les fonction avec une clause WHERE il faudra donc utilisé HAVING après le GROUP BY  N'utilisez HAVING que pour les condition sur les fonction, car HAVING n'est pas optimisé, tandis que WHERE l'est. |

## Obtenir la date/l'heure actuelle

|  |  |  |
| --- | --- | --- |
| **Date actuelle** | SELECT CURDATE(), CURRENT\_DATE(), CURRENT\_DATE; | Renvois la date actuelle, 3 façon de l'écrire pour le même résultat |
| **Heure actuelle** | SELECT CURTIME(), CURRENT\_TIME(), CURRENT\_TIME; | Renvois l'heure actuelle, 3 façon de l'écrire pour le même résultat |
| **Date et heure actuelle** | SELECT NOW();  Alternative : SYSDATE()LOCALTIME, CURRENT\_TIMESTAMP(), LOCALTIMESTAMP | Renvois l'heure et la date actuelle, peut aussi s'utiliser pour ne renvoyer que la date ou seulement l'heure. |

## Formater une donnée temporel

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Extraire DATE** | SELECT DATE(NOW()); | | Renvois une colonne sous formatDATE TIME en format DATE | |
| **Extraire un jour** | SELECT DAY(NOW());  SELECT WEEKDAY(NOW());  SELECT DAYNAME(NOW());  SELECT DAYOFYEAR(NOW());  SET lc\_time\_names = 'fr\_FR'; | | Renvois le jour du mois (sous forme de nombre entre 1 et 31)  Renvois l'index du jour de la semaine (sous forme de nombre entre 0 et 6; lundi=0 etc.)  Renvois le nom du jour de la semaine (string)  Renvois le numéro du jour dans l'année (entre 1 et 366)  Paramètre pour DAYNAME (renverra le nom du jour en français) | |
| **Extraire une semaine** | SELECT WEEK(NOW())  SELECT WEEKOFYEAR(NOW())  SELECT YEARWEEK(NOW()) | | Renvois le numéro de la semaine (entier entre 0 et 53 : 52 semaine complète + 1 semaine de 1 ou 2jour)  Renvois le numéro de la semaine (entier entre 1 et 53 ) (**préférer celui la...)**  Renvois également l'année (sous format YYYYSS ) | |
| **Extraire un mois** | SELECT MONTH(NOW())  SELECT MONTHNAME(NOW()) | | Renvois le numéro du mois (entre 1 et 12)  Renvois le nom du mois (string) | |
| **Extraire l'année** | SELECT YEAR(NOW()) | | Renvois l'année (sous format YYYY) | |
| **Extraire l'heure** | SELECT TIME(NOW())  SELECT HOUR(NOW())  SELECT MINUTE(NOW())  SELECT SECOND(NOW()) | | Renvois l'heure complète d'un DATETIME  Renvois l'heure  Renvois les minutes  Renvois les secondes | |
| **Extraire date et l'heure sous certain format** | SELECT DATE\_FORMAT(NOW(), format)  Exemple :  SELECT DATE\_FORMAT(NOW(), 'le %W %e %M %Y'); | | Pour le format voici la syntaxe : %d = **jour** (de 00 à 31) | %e = **jour** (de 0 à 31) | %D = **jour** (suffixe : 1rst, 2nd ...) | %w = numéro **jour** de la semaine (dimanche =0, lundi=1 etc.) | %W = nom du **jour** | %m = mois (de 00 à 12) | %c = **mois** (de 0 à 12) | %M = nom du **mois** | %y = **année** (2chiffre) | %Y = **année** (4chifre)  %r = **heure** (format 12h) | %T = **heure** (format 24h) | %H = **heure** (2 chiffre) | %k = **heure** (2chifre) | %i = **minute** (de 1 à 2 chiffre) | %s ou %S = **seconde** (2 chifres) et d'autres.... | |
| **Format standard** | SELECT DATE\_FORMAT(NOW(), GET\_FORMAT(DATE, 'EUR') )  SELECT DATE\_FORMAT(NOW(), GET\_FORMAT(TIME, 'EUR') )  SELECT DATE\_FORMAT(NOW(), GET\_FORMAT(DATETIME, 'EUR') ) | | | renvois une DATE sous le format européens  renvois un TIME sous le format européens  renvois un DATETIME sous le format européens |
| **Converti à chaine caractères** | | SELECT STR\_TO\_DATE('03/04/2011 à 09h17', '%d/%m/%Y à %Hh%i') AS StrDate;  SELECT STR\_TO\_DATE('11.21.2011', GET\_FORMAT(DATE, 'USA')) AS date\_usa; | | |

## Calcule sur données temporel

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Différence entre 2 date/heure** | SELECT DATEDIFF(date1, date2)  SELECT TIMEDIFF(temps1, temps2)  SELECT TIMESTAMPDIFF(DAY,temps1, temps2) | | Renvois la différence en nombre de jour entre 2 date  Renvois la différence en format TIME entre 2 DATETIME ou TIME  Renvois la différence en format voulu | HOUR YEAR QUARTER (trimestre) | |
| **Ajout intervalle** | SELECT ADDDATE(NOW(), INTERVAL 3 MONTH)  SELECT ADDDATE(NOW(), 12)  SELECT ADDTIME(NOW(), '05:15:00') | | Ajoute 3 mois a la date | MONTH YEAR DAY WEEK MINUTE SECOND etc...  Ajoute 12 jours a la date  Ajoute 5heures et 15 minutes a l'heure | |
| **Soustraction intervalle** | SELECT SUBDATE('2011-05-21 12:15:56', INTERVAL '3 02:10:32' DAY\_SECOND)  SELECT SUBDATE('2011-05-21', 12)  SELECT DATE\_SUB('2011-05-21', INTERVAL 3 MONTH)  SELECT SUBTIME('2011-05-21 12:15:56', '18:35:15') | | | Soustrait 3 jour et 2h1032s a la date renseigner param 1  Soustrait 12 jours  Soustrait 3 mois  Soustrait 18h35m et 15s au param 1 |
| **Créer une date MAKEDATE** | | SELECT MAKEDATE(2012, 60) | Renvois sous format DATE le 60em jour de l'année 2012 | |
| **Créer une heure MAKETIME** | | SELECT MAKETIME(3,45,34) | Renvois sous format TIME 3h45m34s | |
| **Convertir seconde en TIME**  **Convertir TIME en seconde** | | SELECT SEC\_TO\_TIME(102569), SELECT TIME\_TO\_SEC('01:00:30'); | Renvois sous format TIME la convertion des seconde données en parametre  Et vice-versa | |
| **Dernier jour du mois** | | SELECT LAST\_DAY('2012-02-03') | Renvois le dernier jour du mois de février 2012 (soit le 29) | |

## Transaction (table InnoDB le supporte contrairement à MyISAM)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Quitter l'autocommit** | | SET autocommit=0; | | Par défaut toute les requête effectué sont autocommitée (validé) et ne sont pas réversible. // **ne fonctionne que pour**  Il faut donc redéfinir l'autocommit à 0 pour pouvoir annuler une ou des requête non validé // **la session en cours** | |
| **COMMIT (validé)** | | COMMIT; | | Valide toute les requêtes précédemment réalisé (qui ne sont pas déjà COMMIT) | |
| **ROLLBACK (annulé)** | | ROLLBACK; | | Annule toutes les requêtes qui ne sont pas COMMIT | |
| **Démarrer Transaction** | | START TRANSACTION; | | Démarre une transaction (permet de laisser lautocommit par défaut : à 1) COMMIT ou ROLLBACK met fin transaction | |
| **Jalon de transaction**  *NB: ce sont des sortes de check-point* | | SAVEPOINT nom\_jalon;  ROLLBACK [WORK] TO [SAVEPOINT] nom\_jalon;  RELEASE SAVEPOINT nom\_jalon; | | | Crée un jalon avec comme nom "**nom\_jalon**"  Annule les requêtes exécutées depuis le jalon "**nom\_jalon**", WORK et SAVEPOINT =optionel  Retire le jalon "**nom\_jalon**" (sans annuler, ni valider les requêtes faites depuis) |
| **Validation implicite**  **valide une transaction** | | CREATE | DROP | CREATE | ALTER | RENAME | LOAD DATA | creation, modification, supression d'utilisateur | START TRANSACTION |**Tout ce qui influe sur la structure de la BDD et non des données.** | | | |
| **Concept ACID** | **A**tomicité  **C**ohérence  **I**solation  **D**urabilité | | Une transaction doit être atomique, c'est-à-dire qu'elle doit former une **entité complète et indivisible**.  Les données doivent rester **cohérentes dans tous les cas** : peut importe comment se termine la transaction  Chaque transaction doit être isolée, donc **ne pas interagir avec une autre transaction**.  Une fois la transaction terminée, les données résultant de cette transaction doivent être **stockées de manière durable**. | | |

## Verrous (restreint l'accès à un élément aux autres session) de table pour MyISAM et INNODB / de ligne : INNOBD seul

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Verrouiller une table** | LOCK TABLES nom\_table [AS alias\_table] [READ | WRITE] [, ...]; | | | Avec READ les autre session pourront consulter les donnée, avec WRITE les autres sessions ne pourrons ni lire ni modifier les données de la table | |
| **Attention** si on bloque plusieurs tables il faut les mettre sur la même requêtes, car LOCK TABLE commence par enlever tous les verrous de table de la session  si une session peut lire les données d'une table, elle peut également poser un verrou de lecture.  Si une session peut modifier les données d'une table, elle peut également poser un verrou d'écriture. | | | | | |
| **Déverrouiller une table** | | UNLOCK TABLES | | Déverrouilles toute les tables (pas possible de spécifier lesquels) | |
| **Interaction avec transaction** | | | START TRANSACTION ôte le verrous de table | LOCK TABLES et UNLOCK TABLES provoque un COMMIT (validation implicite) | | |
| **Verrous implicite de ligne** | | | Les requêtes de **modification et suppression** des données **posent automatiquement un verrou exclusif sur les lignes concernées**, à savoir les lignes sélectionnées par la clause WHERE, ou toutes les lignes s'il n'y a pas de clause WHERE.  Les requêtes d'insertion quant à elles posent un **verrou exclusif sur la ligne insérée**. | | |
| **Verrou exclusifs SELECT** | | SELECT \* FROM Animal  WHERE espece\_id = 5 FOR UPDATE; | | | Les autres session ne peuvent ni lire ni modifier la ligne tant que la ligne n'est pas COMMIT |
| **Verrou partagé SELECT** | | SELECT \* FROM Animal  WHERE espece\_id = 5 LOCK IN SHARE MODE; | | | les autres session peuvent lire mais pas modifier la ligne tant que la ligne n'est pas COMMIT |
| * On pose un verrou partagé lorsqu'on fait une requête dans le but de lire des données. * On pose un verrou exclusif lorsqu'on fait une requête dans le but (immédiat ou non) de modifier des données. * Un verrou partagé sur les lignes x va permettre aux autres sessions d'obtenir également un verrou partagé sur les lignes x, mais pas d'obtenir un verrou exclusif. * Un verrou exclusif sur les lignes x va empêcher les autres sessions d'obtenir un verrou sur les lignes x, qu'il soit partagé ou exclusif. | | | | | |
| **ATTENTION** : lors de modification et suppression de donnée avec une clause WHERE si la ou les colonnes dans la clause ne sont pas indexées **TOUTE** les lignes de la colonne acquiert un verrou exclusif. Veiller donc à mètre des INDEX sur les colonnes souvent utilisé pour les transaction. | | | | | |
| **Syntaxe isolation** | SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL { READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE } | | | | |
| Le mot-clé 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  READ UNCOMMITTED niveau par defaut : les requetes de selection non-verouillante travail sur un 'screenshot' de la base de donnée  READ COMMITTED les requete de selection non verouillante travail toujours avec une base de données à jour au moment de la requete  REPEATABLE READ idem que READ COMMITED sauf qu'elle pourra lire les données pas encore COMMIT (donc non sur que celle-ci soit finalement inserer/modifier)  SERIALIZABLE idem que REPEATABLE READ sauf que autocommit est desactivé, tout les SELECT simples sont implicitement convertis en SELECT...LOCK IN SHARE MODE | | | | | |

## Requêtes préparées ( les variables et requête préparés n'existe que pour et pendant la session ou elles sont crée)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Créer Variable utilisateur**  **Sélectionne**  **Sélectionne et modifie** | | SET @age = 24;  SELECT @age, @poids, @salut;  SELECT @age := 32; | | | *Les variables utilisateur doivent toujours être précédé d'un @ (insensible à la casse). les quatre types sont : entier, réel(decimal) string et binaire. les @ sont composé de lettres, chiffre et \_ $ .*  -- On peut sélectionné et modifier une @ en assignant la nouvelle valeur après := | |
| **Requêtes préparé**  **Avec valeur inconnu** | PREPARE nom\_requete  'SELECT \* FROM Client WHERE email = ?' | | | | -- choisir un nom pour la requête préparé (si déjà existante la dernière la remplace)  Créer une requête préparé ou **?** représente un paramètre qu'il faudra appeler (**entre guillemet)** | |
| **Requête préparé dans @** | | SET @req = 'SELECT \* FROM Race'; | | | On peut enregistrer la requêtes dans une variable utilisateur | |
| **Exécuter une requête préparé** | | | EXECUTE nom\_requete [USING @parametre1, @parametre2, ...]; | | | On ne peut renseigner autre chose que des variable utilisateur |
| **Supprimer une requête préparé** | | | | DEALLOCATE PREPARE nom\_requete; | | |

## Procédures stockées (série d'instruction SQL stocké même après la fin de la session)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Créer une procédure stockée** | | | CREATE PROCEDURE nom\_procedure ([parametre1 [, parametre2, ...]])  BEGIN  (instruction de la procédure)  END | | | | | | | Créer une procédure (parenthèse obligatoire)  Indique que la procédure commence  Les instruction de la procédure (**utiliser un ; mais avec délimiteur** )  Indique que la procédure est terminé (utiliser le délimiteur ici !) | |
| **Définir un délimiteur** | | | DELIMITER |  DELIMITER // | | Sera considérer comme fin d'instruction | et non plus ;  Sera considérer comme fin d'instruction // et non plus ; | | | | | |
| **Utilisé une procédure stockée** | | | | CALL nom\_procedure([@var1] [, @var2]) | | | | | Appel la procédure. les | est toujours le délimiteur ! | | |
| **IN OUT INOUT et type** | DELIMITER |  CREATE PROCEDURE afficher\_race\_selon\_espece (IN p\_espece\_id INT)  (...)  DELIMITER ; | | | | | | Par défaut le paramètre renseigner est un IN (et donc facultatif) ensuite il faut indiquer le type (INT, CHAR etc...).IN = paramètre entrant, valeur fournis à la procédure | OUT = paramètre sortant, dont la valeur sera utilisable en dehors et définie par la procédure | INOUT = paramètre entrant et possiblement sortant | | | |
| **OUT dans SELECT... INTO ...** | CREATE PROCEDURE compter\_races\_selon\_espece (p\_espece\_id INT, OUT p\_nb\_races INT)  BEGIN  SELECT COUNT(\*) INTO p\_nb\_races  (...) | | | | | | | | | Le INTO permet d'assigner une valeur de résultat à une variable de sortie. Le résultat assigner ne dois comporter qu'une seul ligne en retour et chaque valeur renvoyé dois avoir une variable de sortie |
| **Exemple**  **INOUT** | | SET @prix = 0;  CALL calculer\_prix (13, @prix);  SELECT @prix AS prix\_intermediaire;  CALL calculer\_prix (24, @prix);  SELECT @prix AS total; | | | | -- On initialise @prix à 0  INOUT peut être utiliser, par exemple comme l'exemple ci-contre, en calculant un prix d'achat pour plusieurs articles. Le prix vaut initialement 0, auquel on ajoute successivement les achat.  Les requêtes SELECT permette ici de voir les différentes valeur de @**prix** au fur et à mesure | | | | |
| **Suppression** | | DROP PROCEDURE afficher\_races; | | | | La seul façon de modifier une procédure est de la supprimer puis de la réécrire | | | | |

## Structurer ses instructions (les structures doivent être dans BEGIN...END dans le cas d'une procédure stockée)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bloc d'instruction** | | | | BEGIN  SELECT 'Bloc d''instructions principal';  BEGIN  SELECT 'Bloc d''instructions 2, imbriqué dans le bloc principal';  BEGIN  SELECT 'Bloc d''instructions 3, imbriqué dans le bloc d''instructions 2';  END;  END;  END; | | | | | | | | | Les bloc d'instruction sont délimiter par BEGIN...END  Ceux-ci peuvent être imbriquer les un dans les autres.  Bien indenter son code en fonction des niveau d'instruction.  Séparer par des ; (sauf dans les procédures !) |
| **Variable local** | | DECLARE nom\_variable type\_variable [DEFAULT valeur\_defaut]; | | | | | | | | | La déclaration des variables doivent être faite en tout début du bloc dans lequel la variable est utilisée. insensible à la casse | | |
| **ATTENTION** : si un bloc imbriqué déclare une variable locale ayant le même nom qu'une variable locale déclarée dans un bloc d'un niveau supérieur, il s'agira toujours de deux variables locales différentes, et seule la variable locale déclarée dans le bloc imbriqué sera visible dans ce même bloc | | | | | | | | | | | | | |
| **Structure IF...**  *NB : peut s'utiliser uniquement dans procédure stockée* | | | | IF condition THEN instructions  [ELSEIF autre\_condition THEN instructions  [ELSEIF ...]]  [ELSE instructions]  END IF; | | | | | | | | Si condition ok alors **instructions**  Sinon si condition alors **instructions**  Sinon si condition alors **instructions**...  Sinon **instructions**  Fin du IF | |
| **Structure CASE...**  *NB: peut s'utiliser dans requête ou procédure stockée* | | | | CASE valeur\_a\_comparer  WHEN possibilite1 THEN instructions  [WHEN possibilite2 THEN instructions] ...  [ELSE instructions]  END CASE; | ou END si ce n'est **pas** dans une procédure stockée | | | | | | | | La valeur qu'ont veut comparer  Si **possibilite1** == **valeur\_a\_comparer** alors **instructions**  Si **possibilite2** == **valeur\_a\_comparer** alors **instructions**  Sinon **instructions** (Attention on ne peut pas comparer a NULL mais ont peut tester avec IS NULL) // Fin de CASE | |
| **Structure IF... dans requête** | | | | | IF(condition, valeur\_si\_vrai, valeur\_si\_faux) | | | | | Ont ne peut utiliser que IF() dans une requête qui n'est pas dans une procédure stockées | | | |
| **Boucle WHILE...** | | | | WHILE condition DO  instructions  END WHILE; | | | | -- Attention de ne pas oublier le DO, erreur classique  La boucle WHILE répète une série d'instruction **tant que** la condition données **reste vrai.** penser à incrémenter pour ne pas créer boucle infinie | | | | | |
| **Boucle REPEAT...** | | | | REPEAT  instructions  UNTIL condition | | | | La boucle REPEAT travaille en quelque sorte de manière opposée à WHILE, puisqu'elle exécute des instructions de la boucle **jusqu'à ce que la condition donnée devienne vraie**.  **Attention**, comme la condition d'une boucle REPEAT est vérifiée après le bloc d'instructions de la boucle, **on passe au moins une fois dans la boucle**, même si la condition est tout de suite fausse ! | | | | | |
| **Label sur boucle ou bloc d'instruction** | | | | | | | [nom\_label]: [WHILE] [REPEAT] [BEGIN] | | | | | Obligatoire de donnée un label pour les instructions ITERATE et LEAVE | |
| **LEAVE** | LEAVE nom\_label; | | | | | Déclenche la sortie immédiate de la structure dont le label est donné (label impossible sur IF et IF()) | | | | | | | |
| **ITERATE** | ITERATE nom\_label; | | | | | Cette instruction ne peut être utilisée que dans une boucle. Lorsqu'elle est exécutée, une **nouvelle itération de la boucle commence**. | | | | | | | |
| **Boucle LOOP...** | | | [label:] LOOP  instructions  END LOOP [label] | | | | | | **Attention** : une boucle LOOP doit intégrer dans ses instructions un élément qui va la faire s'arrêter, typiquement une instruction LEAVE. | | | | |

## Gestionnaires d'erreurs, curseurs et utilisation avancée

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Erreur** | ERROR 1062 (23000): Duplicate entry '21' for key 'ind\_uni\_animal\_id' | | | | | | | **1062** : le **numéro d'erreur MySQL** (un nombre entier) ;  **23000** : l'**identifiant de l'état SQL** (une chaîne de 5 caractères) ;  **Duplicate entry '21' ...**' : un message donnant le détail de l'erreur. | |
| **Création**  **gestionnaire d'erreur** | | | DECLARE { EXIT | CONTINUE } HANDLER FOR { numero\_erreur | { SQLSTATE identifiant\_erreur } | condition }  instruction ou bloc d'instructions (BEGIN...END) | | | | | | |
| * Un gestionnaire d'erreur définit **une seul** instruction, ou un bloc d'instructions (BEGIN ... END;), qui va être **exécuté en cas d'erreur** correspondant au gestionnaire. * Tous les gestionnaires d'erreur doivent être déclarés au même endroit : **après la déclaration des variables locales, mais avant les instructions de la procédure**. * Un gestionnaire peut, soit provoquer l'**arrêt de la procédure** (EXIT), soit **faire reprendre la procédure** après avoir géré l'erreur (CONTINUE). * On peut identifier le type d'erreur que le gestionnaire va reconnaître de trois manières différentes : **un numéro d'erreur, un identifiant, ou une CONDITION**. * Un gestionnaire étant défini grâce au mot-clé DECLARE, comme les variables locales, il a exactement **la même portée** que celles-ci. | | | | | | | | | |
| **Exemple** | DECLARE EXIT HANDLER FOR SQLSTATE '23000'  BEGIN  SELECT 'Une erreur est survenue...';  SELECT 'Arrêt prématuré de la procédure';  END; | | | | | | DECLARE CONTINUE HANDLER FOR SQLSTATE '23000'  SELECT 'Une erreur est survenue...'; | | |
| **Exemple**  **Condition** | DECLARE violation\_contrainte  CONDITION FOR SQLSTATE '23000';  DECLARE EXIT HANDLER FOR violation\_contrainte  BEGIN  SELECT 'Une erreur est survenue...';  SELECT 'Arrêt prématuré de la procédure';  END; | | | | | | -- On nomme l'erreur dont l'identifiant est 23000 "violation\_contrainte"  -- Le gestionnaire sert donc à intercepter les erreurs de type "violation\_contrainte"  Ont peut déclarer plusieurs gestionnaire dans un même bloc d'instruction  lorsque plusieurs gestionnaires d'erreur peuvent correspondre à l'erreur déclenchée (ou à l'avertissement), c'est le plus précis qui est utilisé. | | |
| **Exemple**  **Condition**  **prédéfinie** | DECLARE EXIT HANDLER FOR SQLEXCEPTION  BEGIN  SELECT 'Une erreur est survenue...';  SELECT 'Arrêt prématuré de la procédure';  END; | | | | | | * SQLWARNING : tous les identifiants SQL commençant par '01', c'est-à-dire les avertissements et les notes ; * NOT FOUND : tous les identifiants SQL commençant par '02' * SQLEXCEPTION : tous les identifiants SQL ne commençant ni par '00', ni par '01', ni par '02', donc les erreurs. | | |
| **Déclaration de curseur**  **CURSOR FOR** | | DECLARE curseur\_client CURSOR  FOR SELECT \*  FROM Client; | | | | | On créer un curseur nommé **curseur\_client** et qui comportera les lignes renvoyé par le SELECT  on déclare les curseurs **après les variables locales et les conditions**, mais **avant les gestionnaires d'erreur**. | | |
| **Parcourir le curseur** | | OPEN nom\_curseur;  FETCH nom\_curseur INTO @var1[,@var2][,...];  CLOSE nom\_curseur; | | | | | Ouverture du parcours des lignes renvoyé par le SELECT  Place un curseur sur la première ligne, récupère les résultat et déplace le curseur à la prochaine  Fin du parcours. **Attention** : utiliser une boucle car après la dernière ligne il y a une erreur | | |
| **Exemple boucle curseur** | | DECLARE fin TINYINT DEFAULT 0;  DECLARE curs\_clients CURSOR  (instruction SELECT);  DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin = 1;  OPEN curs\_clients;  loop\_curseur: LOOP  FETCH curs\_clients INTO v\_nom, v\_prenom;  IF fin = 1 THEN  LEAVE loop\_curseur;  END IF;  END LOOP;  CLOSE curs\_clients; | | | | | | | -- Variable locale utilisée pour stopper la boucle  -- Gestionnaire d'erreur pour la condition NOT FOUND  = si une erreur NOT FOUND est rencontré (ce qui arrive à la fin  Du parcours de FETCH) alors **fin** prendra la valeur **1**  -- Structure IF pour quitter la boucle à la fin des résultats  = lorsque **fin** prendra pour valeur **1** (donc quand la boucle à été entièrement parcouru, la boucle prendra fin sans envoyer d'erreur) |
| *NB sur les Booléens en MySQL : les booléens ont pour valeur 1 pour vrai et 0 pour faux. un booléens est de type TINYINT(1) et as comme synonyme BOOL et BOOLEAN. TRUE est synonyme de 1 et FALSE est synonyme de 0.* | | | | | | | | | |
| **utiliser des variables utilisateur dans un bloc d'instructions** | | | | | | Il est possible des les utilisé mais faire attention car si on modifie dans un bloc, elle le sera aussi en global | | | |
| **exécuter une procédure dans un bloc d'instructions** | | | | | | On peut appeler une autre procédure dans un bloc d'instruction | | | |
| **Transaction et gestions d'erreurs** | | | | Un usage classique et utile des gestionnaires d'erreur est l'annulation des transactions en cas d'erreur. | | | | | |
| **créer et exécuter une requête préparée dans un bloc d'instructions** | | | | | Ceci permet de créer des requêtes dynamiques, puisqu'on prépare une requête à partir d'une chaîne de caractères. attention tout de même au problème de sécurité | | | | |

## Triggers (ou déclencheurs)(exécute une instruction ou bloc, dès modification de la table auquel il sont attachés)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Créer un trigger** | CREATE TRIGGER  nom\_trigger  moment\_trigger  evenement\_trigger  ON nom\_table FOR EACH ROW  (corps\_trigger); | | * CREATE TRIGGER **nom\_trigger** : les triggers ont donc un nom. * **moment\_trigger evenement\_trigger** : servent à définir quand et comment le trigger est déclenché. * ON **nom\_table** : c'est là qu'on définit à quelle table le trigger est attaché. * FOR EACH ROW : pour chaque ligne insérée/supprimée/modifiée selon ce qui a déclenché le trigger. * **corps\_trigger** : c'est le contenu du trigger. Comme pour les procédures stockées, il peut s'agir soit d'une seule instruction, soit d'un bloc d'instructions. | | |
| Un trigger est déclenché après INSERT, UPDATE et/ou DELETE (evenement\_trigger) et peut s'exécuter BEFORE ou AFTER (moment\_trigger) la requête.  Dans le corps d'un trigger on peut utiliser OLD et NEW (ex : OLD.client\_nom et NEW.client\_nom dans le cas d'un UPDATE) qui auront les valeur avant et après modification  Dans le cas d'une insertion ou modification, si un trigger peut modifier la valeur de NEW.colonne il doit être exécuter BEFORE (sinon la ligne aura déjà été inséré/modifié)  Un trigger peut modifier et/ou insérer des données dans n'importe quelle table sauf les tables utilisées dans la requête qui l'a déclenché (sauf celle ou il est attaché)  Erreur pendant un trigger BEFORE/AFTER : la requête n'est pas exécuter et en cas de table transactionnel (comme innodb et pas myisam) ROLLBACK  **Commande interdite et limitation trigger** : TRANSACTION, COMMIT, ROLLBACK | requête préparées | les procédures appeler par un trigger ne peuvent pas envoyer d'information au client MySQL (par exemple elle ne peuvent exécuter un SELECT qui produit un affichage client mais peuvent faire un SELECT...INTO @var et peuvent renvoyé des info avec les OUT et INOUT)  **Une suppression ou modification de données déclenchée par une clé étrangère ne provoquera pas l'exécution du trigger correspondant** | | | | | |
| **Supprimer un trigger** | | DROP TRIGGER nom\_trigger; | | Il n'est pas possible de modifier un trigger. si on supprime une table, le/les triggers le seront également | |
| **Exemple de trigger** | DELIMITER |  CREATE TRIGGER before\_update\_animal BEFORE UPDATE  ON Animal FOR EACH ROW  BEGIN  IF NEW.sexe IS NOT NULL  AND NEW.sexe != 'M'  AND NEW.sexe != 'F'  THEN  SET NEW.sexe = NULL;  END IF;  END |  DELIMITER ; | | | | Délimiteur pour le bloc d'instruction  Créer un trigger exécuter avant l'insertion d'une ligne  Sur la table animal, pour chaque ligne  Début du bloc d'instruction  Si le sexe n'est pas NULL  ET qu'il n'est pas **M**  Et qu'il n'est pas **F**  Alors  Le sexe est NULL  Fin du IF  Fin du bloc d'instruction  On remet le délimiteur sur ; |
| **Exemple de trigger**  **Avec création de message d'erreur personnalisé** | CREATE TABLE Erreur (  id TINYINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,  erreur VARCHAR(255) UNIQUE);  INSERT INTO Erreur (erreur) VALUES ('Erreur : sexe doit valoir "M", "F" ou NULL.');  DELIMITER |  CREATE TRIGGER before\_insert\_animal BEFORE INSERT  ON Animal FOR EACH ROW  BEGIN  IF NEW.sexe IS NOT NULL  AND NEW.sexe != 'M'  AND NEW.sexe != 'F'  THEN  INSERT INTO Erreur (erreur) VALUES ('Erreur : sexe doit valoir "M", "F" ou NULL.');  END IF;  END |  DELIMITER ; | | | | On créer une nouvelle table Erreur  Avec un id auto-incrémenté  Et un message erreur UNIQUE  On insère le message erreur dans la table Erreur  On créer le trigger  Si le sexe n'est pas NULL  ET qu'il n'est pas **M**  Et qu'il n'est pas **F**  Alors  Déclenche le message d'erreur personnalisé  (car on tente d'insérer le même message qu'au dessus et qu'il y as un index UNIQUE, ce qui affichera le message a l'utilisateur) |

## Vues (stockage d'une requête SELECT(pas les données) de manière durable, apparais dans les tables)

|  |  |  |  |
| --- | --- | --- | --- |
| **Création d'une vue**  **Remplacement**  **Création d'une vue toute option** | CREATE [OR REPLACE] VIEW V\_nom\_vue  AS requete\_select;  ALTER VIEW nom\_vue  AS requete\_select  CREATE [OR REPLACE]  [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  VIEW V\_nom\_vue [(liste\_colonnes)]  AS requete\_select  [WITH [CASCADED | LOCAL] CHECK OPTION] | | OR REPLACE en cas de remplacement de la vue du même nom.  On place notre requête SELECT après le AS  On peut également remplacer une vue avec ALTER VIEW, qui as le même effet que OR REPLACE mais contrairement a celle-ci, ALTER VIEW génère une erreur si la vue n'est pas existante  MySQL seulement : définie l'algorithme utilisé  [(liste\_colonnes)] est optionnel et contiens les alias (attention a l'ordre)  les modification ne peuvent être effectuer si WHERE ne les sélectionne pas |
| **Utilisation de vue** | SELECT \* FROM V\_nom\_vue; | | Par convention, mettre un V\_ devant le nom de la vue |
| **Restriction** : pas de sous requête dans la clause FROM | pas de variable utilisateur, système, locales | tout le reste est dispo (jointure etc.)  **Attention** : si une vue fait un SELECT \* et que la table sur laquelle elle travail ajoute une colonne, il faudra remplacer la vue car elle stock le nom des colonnes et pas \*  **Remarque** : ne pas mettre de clause LIMIT, DISTINCT, ALL dans une vue, car si l'appel a la vue comporte ces même clause, on ne sais pas laquelle sera pris en compte | | | |
| **Suppression Vue** | DROP VIEW V\_nom\_vue; | | |
| **Condition pour requête UPDATE** | Il est possible de modifier des données à partir d'une vue définie avec une jointure, à condition de ne modifier qu'**une seule table**.  Les commande suivante sont interdite dans la Vue : DISTINCT, LIMIT, fonction d'agrégation ( SUM() etc.), GROUP BY, HAVING, UNION, sous requête  On ne peut pas modifier les données à partir d'une vue qui est elle-même définie à partir d'une vue qui ne permet pas la modification des données  Ce n'est pas non plus possible à partir d'une vue dont la clause WHERE contient une sous-requête faisant référence à une tables de la clause FROM | | |
| **Condition pour requête INSERT** | Reprend toute les condition de la requête UPDATE  Toutes les colonnes sans valeur par défaut et ne pouvant pas être NULL de la table dans laquelle on veut inséré doivent être référencées par la vue  Il ne doit y avoir que des jointures interne | La même colonne ne peut pas être référencée deux fois dans la vue | | |
| **Condition pour requête DELETE** | | Il est possible de supprimer des données si : il est possible de modifier des données a partir de cette vue et que la vue soit mono-table | |

## Tables temporaires (n'existent que dans la session qui les a crées et supprimé à la déconnexion)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Création table temporaire** | CREATE TEMPORARY TABLE TMP\_nom\_table (  Colonne1 description  (...)  ); | | Même syntaxe que pour une table normal mais on ajoute TEMPORARY  Un SHOW TABLES ne montrera que les table et vues, une table temporaire n'apparaitra pas  On peut donner le même nom à une table temporaire qu'une table déjà existante, dans ce cas la vrai table sera masqué et toute requête sera fait sur la temporaire | |
| **Modification** | ALTER TABLE TMP\_nom\_table (...) | | Même syntaxe que pour une table normal | |
| **Suppression** | DROP [TEMPORARY] TABLE TMP\_nom\_table; | | TEMPORARY est optionnel mais préférable (pour être sur de supprimer une table temporaire) | |
| **Restriction** : pas de clé étrangère | on ne peut pas faire référence à une table temporaire deux fois de suite dans la même requête (= auto-jointure, sous requête)  **Transaction** : créer une table temporaire ou supprimer ne provoque pas de validation implicite cependant elle ne peuvent être annulées par ROLLBACK | | | | |
| **Créer table temporaire à partir d'une existante** | | CREATE TEMPORARY TABLE TMP\_nom\_table  LIKE table\_origine; | | Tout les attributs de la table d'origine sont conservés, sauf les clés étrangères  Il est possible de créer une table normal de la même manière sans le TEMPORARY |
| **Créer table temporaire à partir de données** | | CREATE TEMPORARY TABLE TMP\_nom\_table  SELECT ... | | Les colonnes de la table temporaire prendront le nom des colonnes sélectionnée. les valeurs par défaut et les contrainte NOT NULL sont conservé mais pas index AI clés |
| Les tables temporaires permettent de **gagner en performance** lorsque, dans une session on doit exécuter plusieurs requêtes sur un même set de données.  On peut utiliser les tables temporaires pour créer des **données de test**.  les tables temporaires peuvent être utilisées pour stocker un set de résultats d'une procédure stockée. | | | | |

## Vue matérialisées (c'est une vue qui stock les données)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **création** | CREATE TABLE VM\_nom\_table  SELECT (...); | | MySQL ne gère pas nativement les vue matérialisé, on doit donc les crée comme une table normal et faire un SELECT après (pas besoins de créer les colonnes : elle le seront avec SELECT) | |
| **Mise à jour**  **manuel** | DELIMITER |  CREATE PROCEDURE maj\_vm\_revenus()  BEGIN  TRUNCATE VM\_Revenus\_annee\_espece;  INSERT INTO VM\_Revenus\_annee\_espece  SELECT (...);  END |  DELIMITER ; | | | Création d'une procedure  TRUNCATE à le même effet que DELETE FROM (sans WHERE) cependant TRUNCATE supprime toute la table puis la recréer (sans les données), ne traite pas les clés étrangère, valide implicitement les transaction (et pas annulable par ROLLBACK) mais est plus rapide que DELETE FROM en particulier pour une importante masse de données |
| **Appel** | CALL maj\_vm\_revenus(); | | Appelle la procédure précédemment créer pour mettre à jour manuellement les données de la table | |
| **Mise à jour automatique** | | Ont peut créer une mise à jour automatique grâce au triggers | | |

* Les **vues** ne permettent pas de gagner en performance. Elle fournissent juste une **interface**, qui peut permettre de gérer les accès des utilisateurs ou de simplifier les requêtes par exemple.
* Les **tables temporaires** servent lorsque, **ponctuellement**, on doit faire un traitement sur un set de données particulier, pour faire des tests ou des statistiques par exemple.
* Les **vues matérialisées** permettent de stocker des **données sélectionnées ou calculées à partir d'autres tables**. Elles permettent de gagner en performance dans le cas où ces données nécessitent de longs calculs, mais ne changent pas trop souvent (**attention à l'équilibre coût du calcul sans la vue matérialisée et coût de la mise à jour de la vue matérialisée**).

## Gestion des utilisateurs

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Créer utilisateur** | | CREATE USER 'login'@'hote' [IDENTIFIED BY 'mot\_de\_passe'];  CREATE USER 'thibault'@'194.28.12.%' IDENTIFIED BY 'baske'; | | | Hote est l'adresse à partir de laquelle on se connecte, ex : localhost  L'utilisateur Thibault peut se connecter a partir d'une adresse 194.28.12.x | | |
| **Supprimer utilisateur** | | DROP USER 'login'@'hote'; | | | | | |
| **Renommer utilisateur** | | RENAME USER 'max'@'localhost' TO 'maxime'@'localhost'; | | | Renomme max en maxime | | |
| **Modifier mot de passe** | | SET PASSWORD FOR 'thibault'@'194.28.12.%' = PASSWORD('basket8'); | | | | | |
| **Privilèges des tables, vues et base de données** | | CREATE TABLE = création de table | CREATE TEMPORARY TABLE = création de tables temporaires | CREATE VIEW = création de vues (il faut avoir le privilège SELECT sur les colonne sélectionnées par la vue) | ALTER = modification des tables (avec ALTER TABLES) | DROP = suppression de tables, vue et base de données | | | | | |
| **Autres privilèges** | | CREATE ROUTINE = création de procédures et fonctions stockées| ALTER ROUTINE = modification de procédures et fonctions stockées | EXECUTE = exécution de procédures et fonction stockées | INDEX = création et suppression d'index | TRIGGER = création et suppression de triggers | LOCK TABLES = verrouillage de tables (avoir privilège SELECT) | CREATE USER = gestion d'utilisateur (privilèges de table, vue, bdd) | | | | | |
| **Niveau d'application des privilèges** | | \*.\* = privilege global : s'applique à toute les bases de données et tout les objet | \* = Si aucune base n'a été préalablement selectionnée c'est pareil que \*.\* Sinon le privilège s'applique à la base de données qu'ont utilise | **nom\_bdd**\* = s'applique à tous les objets de la base **nom\_bdd** | | | | | |
| **Ajout de privilèges** | | GRANT privilege [(liste\_colonnes)] [, privilege [(liste\_colonnes)], ...]  ON [type\_objet] niveau\_privilege  TO utilisateur [IDENTIFIED BY mot\_de\_passe]; | | | | | Il faut le privilège GRANT OPTION pour ajouter des privilèges au utilisateurs. |
| **Exemple** | | GRANT SELECT,  UPDATE (nom, sexe, commentaires),  DELETE,  INSERT  ON elevage.Animal  TO 'john'@'localhost' IDENTIFIED BY 'exempl2';  GRANT SELECT  ON TABLE elevage.Espece  TO 'john'@'localhost' IDENTIFIED BY 'change2012'; | | On ajoute les privilèges de sélection  De mise à jour dur **nom**, **sexe** et **commentaires**  de suppression  et d'insertion  sur la table **Animal** de la base **elevage**  à **john** s'il se connecte depuis **localhost** avec le mot de passe **exempl2**  on ajoute le privilège de sélection  sur la table **Espece** de la base **elevage**  à **john** (a qui on modifie le mot de passe en **change2012** | | | |
| **Révocation de privilèges**  **Syntaxe et exemple** | | REVOKE privilege [, privilege, ...]  ON niveau\_privilege  FROM utilisateur; | REVOKE DELETE  ON elevage.Animal  FROM 'john'@'localhost'; | | | | |
| **Privilèges**  **ALL** | GRANT ALL  ON elevage.Client  TO 'john'@'localhost'; | | ALL accorde tout les privilèges hormis GRANT OPTION.  Ici on accorde tout les droits sur la table **Client** à '**john**'@'**localhost**' | | | | |
| **Privilèges**  **USAGE** | GRANT USAGE  ON \*.\*  TO 'john'@'localhost' IDENTIFIED BY 'test2012usage'; | | A l'inverse de ALL, le privilège USAGE signifie "aucun privilèges".  Utile pour modifier les caractéristiques d'un compte avec la commande GRANT sans modifier les privilèges du compte. ici on modifie le mot de passe sans modifier les privilèges | | | | |
| **Privilèges**  **GRANT OPTION** | GRANT SELECT, UPDATE, INSERT, DELETE  ON elevage.\*  TO 'joseph'@'localhost' IDENTIFIED BY 'ploc4'  WITH GRANT OPTION; | | Un utilisateur possédant le privilège GRANT OPTION peut accorder des privilèges à d'autre utilisateurs (dans la limite de ceux qu'il possède lui-même).  On peut cumuler les privilèges ALL et GRANT OPTION | | | | |
| **Particularité triggers, vue et procédure stockées** | | Les triggers, vues et procédures stockées sont créés dans le but d'être exécuté dans le futur, il y as donc 2 type d'utilisateur : celui qui l'a créer et celui qui l'utilise. Par défaut ce sont les privilèges de **celui qui as définie l'objet** (**définisseur**) qui sont vérifier, ce qui veut dire qu'un utilisateur qui à le privilège EXECUTE peut exécuter une procédure agissant sur les tables pour lesquelles il n'a lui même aucun privilèges. | | | | | |
| **Préciser et modifier le définisseur** | | DELIMITER |  CREATE DEFINER = CURRENT\_USER() PROCEDURE test\_definer2()  BEGIN  SELECT \* FROM Race;  END |  CREATE DEFINER = 'john'@'localhost' PROCEDURE test\_definer3()  BEGIN  SELECT \* FROM Race;  END |  DELIMITER ; | | | | Les commandes de création des vues, triggers et procédures stockées permettent de préciser une clause DEFINER, dans laquelle on précise l'identifiant d'un utilisateur.  Par défaut, c'est l'utilisateur courant qui est utilisé. Cependant il est possible de donner un autre utilisateur comme définisseur de l'objet, à condition d'avoir le privilège global SUPER. Sans ce privilège, on ne peut donner comme DEFINER que soi-même, soit avec CURRENT\_USER ou CURRENT\_USER(), soit avec l'identifiant ('**sdz**'@'**localhost**'par exemple). La clause DEFINER se place après le mot-clé CREATE | |
| **Modification du contexte** | | -- Vues  CREATE [OR REPLACE]  [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  [DEFINER = { utilisateur | CURRENT\_USER }]  [SQL SECURITY { DEFINER | INVOKER }]  VIEW nom\_vue [(liste\_colonnes)]  AS requete\_select  [WITH [CASCADED | LOCAL] CHECK OPTION]  -- Procédures  CREATE  [DEFINER = { utilisateur | CURRENT\_USER }]  PROCEDURE nom\_procedure ([parametres\_procedure])  SQL SECURITY { DEFINER | INVOKER }  corps\_procedure | | | | Il est possible, pour les vues et les procédures stockées, de changer la manière dont les privilèges sont vérifiés à l'exécution. On peut faire en sorte que ce soit les privilèges de l'**utilisateur** qui se sert de la vue ou de la procédure ("l'invocateur") qui soient vérifiés, et non plus les privilèges du définisseur.  Ce n'est pas possible pour les triggers, car ceux-ci ne sont pas exécutés directement par un utilisateur, mais par une action de l'utilisateur (insertion, modification, suppression dans la table sur laquelle le trigger est défini).  Pour changer le contexte de vérification des privilèges d'une vue ou d'une procédure, il faut utiliser la clause SQL SECURITY {DEFINER | INVOKER}. | |
| **Option supplémentaire**  **GRANT** | | GRANT ALL ON elevage.\*  TO 'aline'@'localhost' IDENTIFIED BY 'limited'  WITH MAX\_QUERIES\_PER\_HOUR 50  MAX\_CONNECTIONS\_PER\_HOUR 5;  GRANT USAGE ON \*.\*  TO 'john'@'localhost'  WITH MAX\_UPDATES\_PER\_HOUR 15;  GRANT USAGE ON \*.\*  TO 'john'@'localhost'  WITH MAX\_UPDATES\_PER\_HOUR 0; | | | | On peut limiter trois choses différentes pour un utilisateur :   * le **nombre** de requêtes par heure (MAX\_QUERIES\_PER\_HOUR) : limitation de toutes les commandes exécutées par l'utilisateur ; * le **nombre** de modifications par heure (MAX\_UPDATES\_PER\_HOUR) : limitation des commandes entraînant la modification d'une table ou d'une base de données ; * le **nombre** de connexions au serveur par heure (MAX\_CONNECTIONS\_PER\_HOUR).   Pour cela, on utilise la clause WITH MAX\_QUERIES\_PER\_HOUR **nb** | MAX\_UPDATES\_PER\_HOUR **nb** | MAX\_CONNECTIONS\_PER\_HOUR **nb** de la commande GRANT. On peut limiter une des ressources, ou deux, ou les trois en une fois, chacune avec un nombre différent. | |

## Informations sur la base de données et les requêtes

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SHOW** | SHOW CHARACTER  SHOW [FULL] COLUMNS FROM nom\_table [FROM nom\_bdd]  SHOW DATABASE  SHOW GRANTS [FOR utilisateur]  SHOW INDEX FROM nom\_table [FROM nom\_bdd]  SHOW PRIVILEGES  SHOW PROCEDURE STATUS  SHOW [FULL] TABLES [FROM nom\_bdd]  SHOW TRIGGERS [FROM nom\_bdd]  SHOW [GLOBAL | SESSION] VARIABLES  SHOW WARNINGS | | | | | | Montre les sets de caractères (encodage) disponible  Liste les colonnes de **nom\_table** et divers infos | FULL + FROM **nom\_bdd** pour préciser la bdd  Montre les bases de données sur lesquelles on possède des privilèges  Liste les privilèges de l'utilisateur courant, ou de l'utilisateur précisé par la clause FOR  Liste les index de **nom\_table**, il est possible de préciser le nom de **nom\_bdd**  Liste les privilèges acceptés par le serveur MySQL (dépend de la version)  Liste les procédures stockées  Liste les tables de la base utilisé ou **nom\_bdd** | FULL ajoute une colonne qui précise table/vue  Liste les triggers de la base utilisée ou de **nom\_bdd**  Liste les variables système | par défaut SESSION = session courante | GLOBAL  Liste les avertissements générés par la dernière requête effectuée. | |
| **Clauses additionnelles** | | | SHOW COLUMNS  FROM Adoption  LIKE 'date%';  SHOW CHARACTER SET  WHERE Description LIKE '%arab%'; | | | | La clause LIKE permet de restreindre la liste aux objets dont le nom correspond au motif donné.  WHERE permet d'ajouter diverses conditions. | |
| **DESCRIBE** | | | DESCRIBE nom\_table; | | | | raccourci pour SHOW COLUMNS FROM **nom\_table** | |
| **Requête de création d'objet** | | | SHOW CREATE TABLE Espece \G  SHOW CREATE TRIGGER before\_insert\_adoption \G | | | | Montre la requêtes ayant servi à créer un objet. \G est un délimiteur comme ; il change la manière d'afficher le résultat non sous forme de tableau mais formaté verticalement | |
| **Base de données information\_shema** | | | | | La base de données information\_shema de MySQL contient les informations sur les tables, les colonnes, les contraintes, les vues etc., des bases de données stockées sur le serveur. | | | |
| **EXPLAIN** | | EXPLAIN SELECT Animal.nom, Espece.nom\_courant, Race.nom FROM Animal  INNER JOIN Espece ON Animal.espece\_id = Espece.id  LEFT JOIN Race ON Animal.race\_id = Race.id  WHERE Animal.id = 37; | | | | | | EXPLAIN permet **d'obtenir des informations sur les requêtes** que l'ont exécute sur nos bases de données. Grâce à cette commande il est possible de savoir quelles tables et quels index sont utilisés, et dans quel ordre. **permet l'optimisation, gain de performance**  Pour l'utiliser il suffit d'ajouter EXPLAIN devant la requête dont ont veut obtenir des informations. |
| **Résultat renvoyé par EXPLAIN** | | | | | **id** | **select\_type** | **table** | **type** | **possible\_keys** | **key** | **key\_len** | **ref** | **rows** | **Extra** | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | SIMPLE | Animal | const | PRIMARY,fk\_espece\_id | PRIMARY | 2 | const | 1 |  | | 1 | SIMPLE | Espece | const | PRIMARY | PRIMARY | 2 | const | 1 |  | | 1 | SIMPLE | Race | const | PRIMARY | PRIMARY | 2 | const | 1 |  | | | | | |
| **Décomposition**  **résultat** | | | **id** | | | identifie la requête concernée par l'étape. Ici, il n'y en a qu'une, mais dans le cas d'une requête avec des sous-requêtes, ou avec un UNION, il peut y avoir plusieurs requêtes SELECT différentes. | | |
| **select\_type** | | | le type de la requête SELECT concernée par l'étape | SIMPLE, PRIMARY, UNION, DEPENDANT UNION, UNION RESULT, SUBQUERY, DEPEDENT SUBQUERY, DERIVED, UNCACHEABLE SUBQUERY, UNCACHEABLE UNION | | |
| **table** | | | le nom de la table sur laquelle l'étape est réalisée. | | |
| **type** | | | le type de jointure utilisée par l'étape. Pour un détail des valeurs possibles, je vous renvoie à la documentation officielle. Les voici cependant classées du meilleur (du point de vue de la performance) au moins bon : **system, const, eq\_ref, ref, fulltext, ref\_or\_null, index\_merge, unique\_subquery, index\_subquery, range, index, ALL**. | | |
| **possible\_keys** | | | les index que MySQL a envisagé d'utiliser pour l'étape. | | |
| **key** | | | l'index effectivement utilisé | | |
| **key\_len** | | | la taille de l'index utilisé (peut être inférieure à la longueur de l'index en cas d'utilisation d'un index par la gauche). | | |
| **ref** | | | indique à quoi l'index sera comparé : une constante (const) ou une colonne. | | |
| **rows** | | | le nombre **estimé** de lignes que MySQL devra parcourir pour terminer l'étape (plus ce nombre est petit, mieux c'est). | | |
| **Extra** | | | donne des informations supplémentaires sur l'étape. | | |
| Pour vérifier l'optimisation d'une requête (en général celle dont on sait qu'elle ralentisse l'application) on fera un EXPLAIN sur la requête. Si celle-ci ne contient pas d'index (donc que **key**, **key\_len** et **ref** sont NULL) on cherchera à en ajouter afin de l'optimiser. | | | | | | | | |

## Configuration et options

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Exemple**  **Variables système** | | Autocommit  Character\_set\_client  Foreign\_key\_checks  Ft\_min\_word\_len  Last\_insert\_id  Max\_connections  Storage\_engine  Unique\_checks | | | | Définie si le mode auto-commit est activé ou non  Jeu de caractère utilisé par le client MySQL  Définit si les contraintes de clé étrangère doivent être vérifiées  Taille minimal d'un mot pour qu'il soit inclus dans une rechercher FULLTEXT  Valeur retournée par la fonction LAST\_INSERT\_ID()  Nombre maximal de connexions autorisées au serveur  Moteur de stockage par défaut  Définit si les contraintes d'unicité doivent être vérifiées | |
| **Afficher variables système** | | SHOW VARIABLES;  SHOW VARIABLES LIKE '%auto\_increment%';  SELECT @@autocommit; | | | | Affiches toute les variables système LOCAL  Affiche la variable système commençant par **auto\_increment**  Sélectionne (et affiche) la variable système **autocommit** *(erreur si n'existe pas)* | |
| Les variables systèmes GLOBAL et LOCAL (=session en cours) peuvent avoir des valeurs différentes, de plus certaine variables existante en GLOBAL n'existe pas en LOCAL et vice-versa. Par défaut quand on fait un SHOW VARIABLES sans préciser GLOBAL | LOCAL c'est LOCAL qui est sélectionné sauf si la variables n'existe qu'en GLOBAL | | | | | | | |
| **Modification variables système**  **SET** | | SET niveau nom\_variable = valeur;  SET SESSION max\_tmp\_tables = 5;  SET @@niveau.nom\_variable = valeur; | Pour modifier une variable au niveau GLOBAL il est nécessaire d'avoir le privilège global SUPER.  Certaine variables ne peuvent être modifier avec la commande SET, en revanche les autres (appelé variable dynamique peuvent être modifier) | | | | |
| **Rappel démarrage du client MySQL** | | mysql --host=localhost --user=sdz --password=motdepasse  mysql -h localhost -u sdz -pmotdepasse | | | | | Les deux exemple ci-contre font la même chose : se connecter au serveur avec un utilisateur |
| **Option démarrage du client MySQL** | | --option[=valeur]  mysql -u sdz -p elevage --skip-column-names  --default-character-set=**encodage**  --delimiter=**delim**  --no-beep  --execute=**requetes**  --init-command=**requete**  --safe-updates **ou** --i-m-a-dummy  --vertical | | | Syntaxe pour les options lors de la connexion  Exemple d'option qui n'affiche pas le noms des colonnes dans les résultats  Définit l'encodage par défaut  Modifie le délimiteur  Le client n'émet plus de son en cas d'erreur  Exécute les requêtes données (séparées par ;) puis quitte le client  Exécute la requête donnée des le démarrage du client  N'exécute les requête UPDATE et DELETE que si les lignes à modifier/supprimer ont WHERE  Ecrit les résultats de manière verticale (comme avec \G ) | | |
| **Option démarrage du serveur MySQL** | | --option[=valeur]  mysqld -C greek  --default-storage-engine=**type**  --default-time-zone=**timezone**  --init-file=**nomFichier**  --language=**langue**  --transaction-isolation=**niveau** | | | Syntaxe pour les options lors de la connexion  Exemple d'option serveur qui modifie l'encodage ( -C = --character-set-server=encodage)  Définit le moteur de stockage par défaut  Définit le fuseau horaire à utiliser  Les requêtes définies dans le dossier donnée sont exécutées au démarrage du serveur  Définit la langue à utiliser pour les messages d'erreur  Définit le niveau d'isolation des transactions | | |
| **Stopper serveur** | | mysqladmin -u root -ppassword shutdown | | | | | |
| **Démarrer serveur** | | mysqld | | | Ou mysqld\_safe pour MacOs et Linux | | |
| **Fichier de configuration** | | Lors de leur démarrage, les logiciels MySQL vérifient l'existence de fichiers de configuration à différents endroits. Si plusieurs fichiers de configuration sont trouvés, ils sont tous utilisés. Si une option est spécifiée plusieurs fois (par plusieurs fichiers différents), c'est la dernière valeur qui est prise en compte (l'ordre dans lequel les fichiers de configuration sont lus est donné ci-dessous). Les emplacements vérifiés sont différents selon que l'on utilise Windows ou Unix.  Il est tout à fait possible que ces fichiers de configuration n'existent pas. Dans ce cas, il suffit de le (ou les) créer avec un simple éditeur de texte. | | | | | |
| **WINDOWS** | | **Emplacement** | | **Commentaire** | | | |
| WINDIR\my.ini, WINDIR\my.cnf | | WINDIR est le dossier de Windows. Généralement, il s'agit du dossier C:\Windows. Pour vérifier, il suffit d'exécuter la commande suivante (dans la ligne de commande windows) : echo %WINDIR% | | | |
| C:\my.ini ou C:\my.cnf | | - | | | |
| INSTALLDIR\my.ini ou INSTALLDIR\my.cnf | | INSTALLDIR est le dossier dans lequel MySQL a été installé. | | | |
| **LINUX et MacOS** | | **Emplacement** | | **Commentaire** | | | |
| /etc/my.cnf | | - | | | |
| /etc/mysql/my.cnf | | - | | | |
| ~/.my.cnf | | ~/ est le répertoire de l'utilisateur Unix. Dans un système avec plusieurs utilisateurs, cela permet de définir un fichier de configuration pour chaque utilisateur Unix (le fichier n'étant lu que pour l'utilisateur Unix courant). | | | |
| Que ce soit sous Windows ou sous Unix, il est également possible de donner un fichier de configuration dans les options au démarrage du logiciel. Dans ce cas, le fichier peut se trouver n'importe où, il suffit de fournir le chemin complet. | | | | | | | |
| **Fichier de configuration dossier personnalisé** | | **Option** | | **Commentaire** | | | |
| --defaults-extra-file=chemin\_fichier | | Le fichier de configuration spécifié est utilisé **en plus** des éventuels autres fichiers de configuration. Ce fichier est utilisé en dernier, sauf sous Unix, où le fichier localisé dans le dossier racine de l'utilisateur Unix est toujours utilisé en tout dernier. | | | |
| --defaults-file=chemin\_fichier | | Seul ce fichier de configuration est utilisé, les autres sont ignorés. | | | |
| **Structure du fichier** | | Un fichier de configuration MySQL peut contenir trois types de lignes différentes.   * **option ou option=valeur** : définit l'option à utiliser. C'est exactement la même syntaxe que pour les options à préciser lors du démarrage du logiciel, à ceci près qu'on omet les deux tirets -. * **[logiciel]** ou **[groupe]** : définit le logiciel ou le groupe auquel les options s'appliquent. * **#commentaire** ou **;commentaire** : ligne de commentaire, elle sera ignorée. Notez qu'il est possible de commencer un commentaire au milieu d'une ligne avec #. | | | | | |
| **Options** | Toutes les options disponibles en ligne de commande lors du démarrage d'un logiciel sont utilisables dans un fichier de configuration (pour le logiciel correspondant). On omet simplement les deux caractères - avant l'option, et les raccourcis ne peuvent pas être utilisés.  Si l'option nécessite que l'on précise une valeur, celle-ci peut, mais ne doit pas, être entourée de guillemets. Si la valeur donnée comprend un #, les guillemets seront nécessaires pour que ce caractère ne soit pas considéré comme le début d'un commentaire.  Par ailleurs, les espaces avant et après l'option sont ignorées, et les espaces autour du signe = sont autorisées (ce n'est pas le cas en ligne de commande). | | | | | | |
| **balises [logiciel] et [groupe]** | | Les balises **[logiciel]** et **[groupe]** permettent de spécifier **à quel(s) logiciel(s) s'appliquent les options suivant cette balise** (jusqu'à la balise **[logiciel]** ou **[groupe]** suivante, ou jusqu'à ce que la fin du fichier soit atteinte).  On peut donc, soit donner le nom du logiciel concerné (**[mysqld]**, **[mysql]**, **[mysqldump]**, etc.), soit donner le groupe **[client].** Si le groupe **[client]** est spécifié, **les options suivantes seront prises en compte par tous les logiciels clients**. mysqldump, mysqladmin et mysql sont tous trois des logiciels clients.  Les options données pour ce groupe doivent être valables pour tous les logiciels clients. Si vous essayez de démarrer un logiciel client alors qu'une option non valable pour celui-ci a été donnée dans le fichier de configuration, le logiciel quittera avec une erreur.  Bien entendu, il est possible de spécifier une balise **[client]** pour les options communes à tous les clients, puis une balise **[logiciel]** pour les options spécifiques à un logiciel. | | | | | |
| **Exemple**  ***NB :*** *Tous les logiciels clients, ainsi que le serveur utiliseront le port 3306, mais seul le logiciel* mysql *utilisera l'option* no-beep*.* | | **[client]**  port=3306 # on précise le port à utiliser pour tous les logiciels clients  **[mysqld]**  port=3306 # on précise le port aussi pour le serveur  character-default-set=utf8  **[mysql]**  no-beep | | | | | |