**Commandes en SQL**

Raccourcis d'accès à l'invite de commande :

Windows + r => "cmd" + Entrer

Windows + x => c

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | | Utilisation | | | | | | |
| Invite de commande |  |  | |  | | | | | | |
| set PATH= | enregistre un nouveau chemin d'accès aux programmes |  | | set PATH=%PATH%; C:\"Program Files"\MySQL\"MySQL Server 8.0"\bin | | | | |  | |
| cd | accède au dossier indiqué  remonte dans l'arborescence  d'un dossier  de deux dossiers | cd MonDossier\MonSousDossier  cd ..  cd ..\.. | | cd Program Files\MySQL\MySQL Server 8.0\bin | | | | | Si la définition du Path n'a pas fonctionné, se rends manuellement dans le dossier bin avant de se connecter au client ! | |
| mysql | permet de se connecter au client et d'avoir accès à MySQL | mysql --host=localhost --user=root --password=monMotDePasse  mysql -h localhost -u root -pmonMotDePasse  mysql -h localhost -u root -p  mysql -h localhost -u root -p --default-character-set=utf8 | | | | |  | | Rq: 3e syntaxe à privilégier afin de ne pas faire apparaitre le MdP dans l'invite de commande (il sera demandé à l'étape suivante) | |
| SET NAMES | appelée après la connexion, elle définit l'encodage utilisé pour les échanges | mysql> SET NAMES 'utf8'; | |  | | | | | ATTENTION : à exécuter à CHAQUE connexion ! Sauf si précisé dans les paramètres de connexion. | |
| quit  exit | déconnexion du client | quit | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| SQL |  |  | |  | | | | |  | |
| ;  \g | termine une instruction | mysql> monInstruction; | |  | | | | |  | |
| -- | commentaire  ATENTION : uniquement si suivit d'au moins UN espace ! | -- ceci est un commentaire  --ceci est une instruction | |  | | | | |  | |
| # | commentaire non générique de SQL et reconnu uniquement par MySQL | #mon commentaire | |  | | | | |  | |
| ' ' | chaine de caractères | 'Ma chaine de caractères'  'Salut l''ami' | |  | | | | | Rq: pour échapper une apostrophe on peut utiliser le \ avec MySQL, mais il est préférable de la doubler (= norme SQL) | |
| \  \n  \t  \\ | échappement des caractères spéciaux :  retour à la ligne  tabulation  antislash |  | |  | | | | | ATTENTION : valable uniquement avec MySQL ! Pour les autres SGBD, il faut se renseigner (doublement du caractère ou autre caractère d'échappement spécifique) | |
| \_ |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| SELECT | sélectionne les données et les affiches | SELECT 'Hello World !';  SELECT (5+3)\*2, 5+3\*2; | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| + - | addition soustraction |  | | *résultat :*   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 1+1 | 4-2 | 3\*6 | 5/2 | 5 DIV 2 | 5%2 | 5 MOD 2 | | 2 | 2 | 18 | 2.500 | 2 | 1 | 1 | | | | | | |  |
| \* / | multiplication division | SELECT 1+1, 4-2, 3\*6, 5/2, 5 DIV 2, 5 % 2, 5 MOD 2; | |  |
| % DIV | modulo division entière |  | |  |
|  |  |  | |  | | | | |  | |
| Boolean |  |  | |  | | | | |  | |
| BIT | bit (0 ou 1)  sert notamment à stocker les boolean (car le type boolean true/false n'existe pas en SQL) | BIT | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Données numériques |  |  | |  | | | | |  | |
| TINYINT  SMALLINT  MEDIUMINT  INT  BIGINT | nombres entiers relatifs (équiv. Java)  1 octet (byte)  2 octets (short)  3 octets  4 octets (int)  8 octets (long) |  | |  | | | | |  | |
| DECIMAL( , ) NUMERIC( , ) | nombres décimaux  param1: nb de chiffres significatifs  param2: nb chiffres après la virgule | DECIMAL(5,3) => 1,2 12345 12,345 | |  | | | | | Rq: en cas de dépassement de chiffres après la virgule, MySQL arrondit la valeur.  Rq: stockage de la valeur exacte des nb sous forme de chaine ! Idéal pour la comparaison ! | |
| FLOAT  ***REAL***  DOUBLE | nombres décimaux  4 octets (float)  4 octets  8 octets (double)  param1: nb de chiffres significatifs  param2: nb chiffres après la virgule | FLOAT  FLOAT(5,3) | |  | | | | | ATTENTION : avec MySQL, REAL est aussi défini sur 8 octets => pb lors d'un éventuel changement de SGBD, à proscrire donc !  Rq: stockage sous forme de valeur approchée (ex: 1,23 => 1,2300000000001). Bien pour quantité, mais pas pour comparaison stricte ! | |
| UNSIGNED | attribut interdisant les nb négatifs |  | |  | | | | |  | |
| ( )  ( ) ZEROFILL | définit une taille minimale d'affichage du nb | INT(3) => 5 23 1586  INT(3) ZEROFILL => 005 023 1586 | |  | | | | | ATTENTION : l'utilisation de l'espace comme remplissage automatique peut poser des problèmes lors de l'utilisation des données ! | |
|  |  |  | |  | | | | |  | |
| Dates |  |  | |  | | | | |  | |
| DATE | stocke une date, cad jour, mois et année |  | | *Syntaxes pour définir une date :*   |  |  | | --- | --- | | *en nombre* | *en chaine* | | AAMMJJ  AAAAMMJJ | 'AAMMJJ'  'AAAAMMJJ'  'AAAA-MM-JJ'  'AA/MM/JJ'  'AAAA;MM;JJ"  … | | | | | | Rq: pour entrer une date en chaine de caractère, peu importe le séparateur utilisé.  Rq: c'est sous la forme en rouge que la date sera stockée par MySQL. | |
| DATETIME | stocke une date et une heure |  | | *Syntaxes pour définir un* DATETIME *:*   |  |  | | --- | --- | | *en nombre* | *en chaine* | | AAMMJJHHMMSS  AAAAMMJJHHMMSS | 'AAMMJJ HHMMSS'  'AAAAMMJJ HHMMSS'  'AAAA-MM-JJ HH:MM:SS'  'AA/MM/JJ HH+MM+SS'  … | | | | | | ATTENTION : peu importe le séparateur, mais date et heure doivent être séparés par un espace si défini par une chaine de caractère ! | |
| TIME | stocke une heure précise ou alors un intervalle de temps pouvant être de plusieurs jours ou même négatif |  | | *Syntaxes pour définir un* TIME *:*   |  |  | | --- | --- | | *en nombre* | *en chaine* | | HHMMSS | 'HHMMSS'  'HHH:MM:SS'  'J HH:MM:SS'  … | | | | | | ATTENTION : si l'intervalle comprend un nombre de jour, il doit être précisé en premier et séparé du reste par un espace ! | |
| YEAR | stocke une année seulement sur un octet => valeurs possibles de 1901 à 2155 |  | |  | | | | |  | |
| TIMESTAMP | le timestamp d'une date = nb de secondes écoulées depuis le 01/01/1970 0h00min00  stocké sur 4 octets => date lim 19/01/2038 |  | |  | | | | | ATTENTION: en réalité, en SQL ce n'est pas le vrai timestamp qui est stocké, mais la date sous forme numérique AAAMMJJHHMMSS  Il est impossible de stocker un VRAI timestamp dans la colonne TIMESTAMP ! | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Chaine de caractères |  |  | |  | | | | |  | |
| CHAR( ) | texte court de moins de 255 caractères  toutes les chaines de même longueur (= param) et complétées par des espaces au besoin | CHAR(5) | |  | | | | |  | |
| VARCHAR( ) | texte court de moins de 255 caractères  chaine de longueur variable (max = param) et stockant également la longueur de la chaine en question |  | |  | | | | |  | |
| TINYTEXT  TEXT  MEDIUMTEXT  LONGTEXT | texte long comprenant au maximum :  28 octets 256 caractères UTF-8  216 octets 65 536  224 octets 16 777 216  232 octets 4 294 967 296 |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| *ENUM(' ', ' ')* | Enumération = uniquement certaines valeurs de type chaine autorisées  (max 65 535 valeurs possibles)  ATTENTION : spécifique à MySQL | espece ENUM('chien', 'chat', 'cheval') | | *Valeurs possibles pour la colonne* **espece** *:*   |  |  | | --- | --- | | *index* | *valeur* | |  | NULL | | 0 | '' | | 1 | 'chien' | | 2 | 'chat' | | 3 | 'cheval' | | | | | | Rq: si on essaie d'introduire une valeur non autorisée, la colonne contiendra soit une chaine vide, soit NULL (si colonne autorisée lors de sa création à ne pas avoir de valeur) | |
| *SET(' ', ' ')* | Set = un ou plusieurs éléments parmi les valeurs autorisées  (max 64 valeurs possibles)  ATTENTION : spécifique à MySQL | espece SET('chien', 'chat', 'cheval') | | *Valeurs possibles pour la colonne* **espece** *:*   |  |  |  | | --- | --- | --- | | *Binaire* | *Décimal* | *valeur* | |  |  | **NULL** | | **000** | **0** | **''** | | **001** | **1** | **'chien'** | | **010** | **2** | **'chat'** | | **100** | **4** | **'cheval'** | | 011 | 3 | 'chien,chat' | | 101 | 5 | 'chien,cheval' | | 110 | 6 | 'chat,cheval' | | 111 | 7 | 'chien,chat,cheval' | | | | | | ATTENTION : les chaines de valeurs possibles dans un SET ne peuvent contenir de virgule car celle-ci sert à séparer ces valeurs en cas de valeurs multiple dans une même case !  Rq: pas de doublons ! ~~'chien,chien'~~ | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Chaines Binaires | suite d'octets sans interprétation (texte, images, objets, …) |  | |  | | | | |  | |
| BINARY( ) | binaire court de moins de 255 caractères  toutes les chaines de même longueur (= param) et complétées au besoin |  | |  | | | | |  | |
| VARBINARY( ) | binaire court de moins de 255 caractères  chaine de longueur variable (max = param) et stockant également la longueur de la chaine en question |  | |  | | | | |  | |
| TINYBLOB  BLOB  MEDIUMBLOB  LONGBLOB | binaire long comprenant au max :  28 octets 256 o  216 octets 65 ko  224 octets 16 Mo  232 octets 4 Go |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Utilisateur |  |  | |  | | | | |  | |
| CREATE USER | création d'un nouvel utilisateur | CREATE USER 'userName'@'localhost' IDENTIFIED BY 'monMotDePasse'; | |  | | | | |  | |
| @ | définit à partir d'où l'utilisateur pourra se connecter |  | | | | |  | |
| IDENTIFIED BY | définit le mot de passe de l'utilisateur |  | | | | |  | |
| IDENTIFIED WHITH  BY | précise le type d'authentification  (ATTENTION : type d'authentification modifiée comme ceci depuis SQL 8) | CREATE USER 'userName'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'monMotDePasse'; | |  | | | | |  | |
| ALTER USER | modification d'un utilisateur | ALTER USER 'username'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'userpassword'; | |  | | | | |  | |
| GRANT | attribuer des droits à l'utilisateur sur la modification du contenu de certains éléments | GRANT ALL PRIVILEGES ON maBDD.monTableau TO 'userName'@'localhost'; | | *Pour accorder tous les droits :*  GRANT ALL PRIVILEGES ON maBDD.\* TO 'userName'@'localhost'; | | | | |  | |
| ON | précise les BDD et les tables sur lesquels les droits sont donnés |  | |
| TO | définit l'utilisateur et son hote à qui ces droits sont accordés |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Base de donnée |  |  | |  | | | | |  | |
| SHOW DATABASES | liste les BDD existantes dans le moteur | SHOW DATABASES | |  | | | | |  | |
| CREATE DATABASE | créée une nouvelle BDD | CREATE DATABASE maBaseDD CHARACTER SET 'utf8'; | |  | | | | |  | |
| CHARACTER SET | défini l'encodage utilisé par défaut de toutes les tables que contiendra la BDD (il sera alors inutile de le préciser à chaque création de table) |  | | | | |  | |
| IF NOT EXISTS | évite le message d'erreur si une BDD existe déjà sous ce nom | CREATE DATABASE IF NOT EXISTS maBaseDD CHARACTER SET 'utf8'; | |  | | | | |  | |
| DROP DATABASE | supprime la BDD et TOUTES les tables et les infos qu'elle contient ! | DROP DATABASE maBaseDD;  DROP DATABASE IF EXISTS maBaseDD; | |  | | | | |  | |
| USE | sélectionne la BDD sur laquelle on va travailler | USE maBaseDD; | | *Connexion au client ET à la BDD à utiliser en une étape :*  mysql -h localhost -u nomUtilisateur -p --default-character-set=utf8 maBaseDD | | | | |  | |
| SHOW TABLES | liste les tables de la BDD | SHOW TABLES; | |  | | | | |  | |
| DESCRIBE | liste les colonnes de la table en question | DESCRIBE maTable; | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Table |  |  | |  | | | | |  | |
| CREATE TABLE | création d'une table dans la BDD | CREATE TABLE IF NOT EXISTS maTable **(**  maColonne1 INT UNSIGNED NOT NULL AUTO\_INCREMENT,  maColonne2 VARCHAR(30) NOT NULL DEFAULT 'MaValeurDefaut',  maColonne3 DATE NOT NULL DEFAULT '2000-01-01',  maColonne4 TEXT,  PRIMARY KEY(maColonne1)  **);** | | | | | |  | Rq: par défaut NULL est autorisée.  Rq: NULL est la valeur par défaut s'il est autorisé et aucune valeur par défaut précisée.  ATTENTION : la valeur par défaut doit être une constante ! | |
| PRIMARY KEY( ) | définit une colonne comme clé primaire, cad comme identifiant unique pour chaque ligne  => refus d'insérer une nouvelle ligne si elle contient une clé déjà existante |  |  | |
| AUTO\_INCREMENT | option permettant d'utiliser la dernière valeur entrée dans la colonne et de l'incrémenter dans la nouvelle ligne insérée  (souvent utilisée pour la clé primaire) |  |  | |
| ENGINE | précise le moteur de table qui doit être utilisé pour gérer la table en question (par défaut = MyISAM)  ATTENTION : spécifique à MySQL | CREATE TABLE IF NOT EXISTS maTable **(**  **…**  **)** ENGINE=INNODB**;** | |  | | | | | Rq: MyISAM permet un traitement rapide des tables, mais ne gère pas les clés étrangères ni les transactions.  InnoDB est plus lent, mais gère tout. | |
| DESCRIBE | liste les colonnes (nom, type, caractéristiques, options) de la table | DESCRIBE maTable; | |  | | | | |  | |
| SELECT \* FROM | obtenir toutes les données de la table | SELECT \* FROM maTable; | |  | | | | |  | |
| DROP TABLE | supprime la table en question et toutes ses données ! | DROP TABLE IF EXISTS maTable; | |  | | | | |  | |
| ALTER TABLE  ADD  DROP | modifier la table  ajouter  supprimer  une colonne | ALTER TABLE maTable ADD maColonne5 INT NOT NULL;  ALTER TABLE maTable DROP maColonne4; | | ALTER TABLE maTable ADD maColonne5 INT **NOT NULL** DEFAULT 0; | | | | | Rq: DEFAULT permet de remplir les lignes préexistantes avec une valeur par défaut si la nouvelle colonne n'est pas nullable | |
| CHANGE  MODIFY | modifier nom et/ou type de donnée  modifier le type de données | ALTER TABLE maTable CHANGE maColonne5 nouveauNomColonne5 BIGINT NOT NULL;  ALTER TABLE maTable MODIFY maColonne2 VARCHAR(30) DEFAULT 'MaNouvelleValeurDefaut'; | | *=> nom et type de donnée modifiée*  *=> valeur par défaut modifiée et NULL accepté* | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Ligne / Données |  |  | |  | | | | |  | |
| INSERT INTO | ajouter une ligne de données à la table | INSERT INTO maTable VALUES(valeur1, valeur2, valeur3, valeur4);  INSERT INTO maTable VALUES(NULL, valeur2, valeur3, NULL); | | *=> 1er NUL remplacé par nb via l'auto-incrémentation. 2e NULL gardé comme vide.* | | | | |  | |
| ( , , ) | en précisant le nom des colonnes | INSERT INTO maTable (maColonne2, maColonne3) VALUES (valeur2, valeur3);  INSERT INTO maTable (maColonne2, maColonne3) VALUES (valeur2A, valeur3A),  (valeur2B, valeur3B),  (valeur2C, valeur3C); | | *=> équivalente à la commande ci-dessus !*  *=> insertion de plusieurs lignes en même temps* | | | | |  | |
| *SET* | ATTENTION : spécifique à MySQL | INSERT INTO maTable SET maColonne1=valeur1, maColonne2=valeur2, maColonne3=valeur3; | |  | | | | |  | |
| DELETE | suppression des données (des lignes) sélectionnées de la même manière qu'avec SELECT | DELETE FROM maTable WHERE maColonneInt = 8; | |  | | | | |  | |
|  | vider toute la table | DELETE FROM maTable; | |  | | | | |  | |
| UPDATE | modifier les données  SET indique les nouvelles valeurs  WHERE sélectionne la/les lignes concernées | UPDATE maTable  SET maColonneInt = 5, maColonneVarChar = 'nouvelle valeur'  WHERE maColonneId = 18;  UPDATE maTable  SET maColonneVarChar = 'nouvelle valeur'; | | *=> attribution des nouvelles valeurs 5 et 'nouvelle valeur' aux colonnes concernées de la ligne identifiée par l'id 18*  *=> ATTENTION : modifie les valeurs de la colonne VarChar pour toutes les lignes de la table !!!* | | | | |  | |
|  | gérer les contraintes d'unicité (primary key et index UNIQUE) |  | |  | | | | |  | |
| IGNORE | option permettant de na pas lever d'erreur en cas de violation de contrainte d'unicité | **INSERT** IGNORE INTO maTable (maColonne2, maColonne3) VALUES (valeur2, valeur3);  **UPDATE** IGNORE maTable  SET maColonneVarChar = 'nouvelle valeur'; | |  | | | | | ATTENTION : en cas de violation d'unicité, aucune erreur ne s'affiche, mais la ligne n'est pas affectée ! | |
| REPLACE | à utiliser à la place de INSERT pour automatiquement supprimer les lignes existantes qui provoquent une violation de la contrainte d'unicité à l'insertion, puis insère la nouvelle ligne | REPLACE INTO maTable (maColonne2, maColonne3) VALUES (valeur2, valeur3); | |  | | | | |  | |
| ON DUPLICATE KEY UPDATE | option à ajouter à INSERT afin de seulement mettre à jour les données de la ligne contenant la valeur qui doit être unique | **INSERT** INTO maTable (maColonne2, maColonne3, maColonne4) VALUES (valeur2, valeur3, valeur4)  ON DUPLICATE KEY UPDATE maCononne2 = valeur2, maColonne4 = valeur4; | | *ici, si la colonne 3 possède un index UNIQUE et que la valeur3 existe déjà dans la table, alors les colonnes 2 et 4 seront mise à jour sur cette ligne. Sinon, une nouvelle ligne sera créée.* | | | | | ATTENTION : si plusieurs contraintes d'unicités sont violées par la même instruction sur plusieurs lignes, seule une seule de ses lignes sera mise à jour sans que l'on puisse prévoir laquelle ! | |
|  |  |  | |  | | | | |  | |
| LOAD DATA | ajouter des données à partir d'un fichier formaté (format .csv par exemple) | LOAD DATA LOCAL INFILE 'C:\adresseFichier\monFichier.csv'  INTO TABLE maTable  FIELDS  TERMINATED BY '\t'  ENCLOSED BY ''  ESCAPED BY '\\'  LINES  STARTING BY ''  TERMINATED BY '\n' '\r\n' avec Windows  IGNORE 1 LINES  (maColonne2, maColonne3); | | *ATTENTION sous Windows :*  LOAD DATA LOCAL INFILE 'C:/adresseFichier/monFichier.csv' | | | | |  | |
| LOCAL | précise que le fichier est côté client. Il faut alors préciser son chemin d'accès complet.  Si non précisé => fichier côté serveur. Dans ce cas, obligatoirement dans le répertoire de la BDD créé par MySQL | *Vérifier la valeur de* ***local\_infile*** *:*  SHOW VARIABLES where Variable\_name = 'local\_infile';  ***Autoriser le local\_infile :***  *D'abord sous l'utilisateur root :*  SET @@global.local\_infile = 1;  *Puis lors de la connexion à l'utilisateur de la BDD :*  mysql -h localhost … =utf8 --local-infile=1 | | | | | ATTENTION : avec LOCAL, pour avoir accès aux fichiers, il faut que la variable local\_infile de MySQL soit à 1 (= autorisée). Or par défaut elle vaut 0. | |
| FIELDS  LINES | définit le format de fichier utilisé  (facultatif, valeurs par défaut = valeurs indiquées ici)  Si précisées, définir au moins une des sous-clauses possibles chacun |  | | | | | ATTENTION : avec les fichiers créés sous Windows => LINES TERMINATED BY '\r\n' | |
| IGNORE X LINES | ignore les X premières lignes (not. la 1re ligne qui précise souvent le titre des colonnes) |  | | | | |  | |
|  | pour gérer les cas de violation de contrainte d'unicité : | *avec l'option* IGNORE *pour simplement ignorer les messages d'erreurs :*  LOAD DATA LOCAL INFILE 'C:\adresseFichier\monFichier.csv' IGNORE  INTO TABLE maTable  … | | *avec l'option* REPLACE *pour supprimer les anciennes valeurs et les remplacer par les nouvelles :*  LOAD DATA LOCAL INFILE 'C:\adresseFichier\monFichier.csv' REPLACE  INTO TABLE maTable  … | | | | |  | |
|  |  |  | |  | | | | |  | |
| Fichier |  |  | |  | | | | |  | |
| SOURCE  \. | charge un fichier et exécute les commandes qu'il contient | SOURCE C:\Users\nomUtilisateur\monDossie\monFichier.sql;  \. Users\nomUtilisateur\monDossie\monFichier.sql; | *ATTENTION sous Windows :*  SOURCE C:/Users/nomUtilisateur/monDossie/monFichier.sql;  \. C:/Users/nomUtilisateur/monDossie/monFichier.sql; | | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Sauvegarde |  |  | |  | | | | |  | |
| mysqldump > | client permettant la sauvegarde des BDD à partir de l'invite de commande | mysqldump -h localhost -u utilisateur -p --opt MaBaseDD > C:\adresse\nomFichierSauvegarde.sql | |  | | | | | Rq: la sauvegarde créé un fichier texte qui contient toutes les commandes SQL pour recréer la BDD | |
| mysql < | ATTENTION : pour récupérer la sauvegarde, il faut que la BDD existe dans MySQL (si elle a été supprimée, il faut au préalablement se connecter à MySQL et la recréer avant de la remplir grâce à la sauvegarde) | ***En ligne de commande :***  mysql MaBaseDB < C:\adresse\nomFichierSauvegarde.sql  ***Via MySQL :***  USE MaBaseDB;  SOURCE C:\adresse\nomFichierSauvegarde.sql; | | *ATTENTION sous Windows :*  USE MaBaseDB;  SOURCE C:/adresse/nomFichierSauvegarde.sql; | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Sélection des données |  |  | |  | | | | |  | |
| SELECT | permet d'afficher des données, résultats de traitement, … | SELECT 3+2; | |  | | | | |  | |
| DISTINCT | option permettant d'éliminer les valeurs identiques | SELECT DISTINCT maColonneVarChar FROM maTable; | | *Exemple :*  SELECT DISTINCT espece FROM Animal;  = ne retourne pas les 500 lignes, mais seulement les 5 ou 6 espèces différentes présentent dans l'élevage. | | | | |  | |
| IFNULL( , ) | option à appliquer sur les colonnes.  Si la valeur de la colonne est nulle, renvoie la valeur passée en param 2 |  | | SELECT NOM, PRENOM, DATE\_NAISSANCE, IFNULL(COMMISSION, 0) AS commission FROM employes; | | | | | Rq: tjrs utiliser avec un alias, sinon le nom de la colonne est : "IFNULL(COMMISSION, 0)"  ATTENTION :   * MySQL => IFNULL( , ) * SqlServer => ISNULL( , ) * Oracle => NVL( , ) | |
| FROM | accompagné de FROM il permet de sélectionner dans les tables les données à afficher | SELECT maColonne1, maColonne2 FROM maTable;  SELECT \* FROM maTable; | |  | | | | |  | |
| WHERE | permet de définir des conditions de sélection sur les données | SELECT \* FROM maTable WHERE maColonneVarChar = 'Valeur particulière';  SELECT \* FROM maTable WHERE maColonneInt >= 3;  SELECT \* FROM maTable WHERE maColonneDate < '2008-01-01';   |  |  | | --- | --- | | Opérateurs | | | = | égal | | <=> | égal valable pour NULL aussi | | != ou <> | différent | | < <= | inférieur ou égal | | > >= | supérieur ou égal | | AND ou && | et | | OR | ou | | XOR | ou exclusif | | NOT ou ! | non | | | *Exemples :*  SELECT \* FROM Animal  WHERE sexe='F' AND NOT espece='chien';  = obtenir tous les animaux femelle sauf les chiennes  WHERE sexe='M' XOR espece='chat';  = obtenir tous les animaux mâles et les chattes (=tous les animaux qui sont mâles et chats, mais pas les deux en même temps)  WHERE surnom <=> NULL;  WHERE surnom IS NULL;  = obtenir tous les animaux qui n'ont pas encore été nommés  WHERE surnom IS NOT NULL;  = obtenir tous les animaux qui porte un surnom | | | | |  | |
| GROUP BY | regroupe les lignes si la valeur du champ est identique  regroupe d'abord par Nom, puis par NomJF | *Exemple avec une table* "personne"  ID NOM NOMJF PRENOM AGE  SELECT NOM, NOMJF, avg(AGE), count(PRENOM)  FROM persone  GROUP BY NOM, NOMJF; | | SELECT a.espece, **COUNT(\*)** **AS nb\_animaux**  FROM animal AS a  GROUP BY **nb\_animaux**; | | | | | ATTENTION : les colonnes apparaissant dans le SELECT doivent obligatoirement   * soit être dans le GROUP BY * soit avoir une fonction qui regroupe leurs valeurs   ATTENTION : GROUP BY ne doit contenir aucun calcul, aucune fonction ! Si nécessité de trier par un résultat, le calcul se fait dans le SELECT et on utilise l'alias (voir ex. ci-contre)  Rq: l'option WITH ROLLUP permet d'obtenir les résultats des groupes intermédiaires et globaux. *Voir documentation SQL ou site OpenClassRooms* | |
| HAVING | permet d'ajouter une condition de sélection sur le résultat d'une fonction d'agrégation (ce qui n'est pas possible dans la clause WHERE) | SELECT a.espece, COUNT(\*) AS nb\_animaux  FROM animal AS a  GROUP BY a.espece  HAVING nb\_animaux > 15; | | SELECT a.espece, COUNT(\*) AS nb\_animaux  FROM animal AS a  **WHERE nb\_animaux > 15** *=> ERREUR !*  GROUP BY a.espece; | | | | |  | |
| ORDER BY | permet de trier les résultats selon le critère choisi  tri par ordre croissant  tri par ordre décroissant  tri d'abord par Date, puis par Int | SELECT \*  FROM maTable  WHERE maColonneVarChar = 'Valeur'  ORDER BY maColonneDate; *(*ASC *par défaut)*  ORDER BY maColonneDate DESC;  ORDER BY maColonneDate, maColonneInt; | |  | | | | | ATTENTION : toujours en dernier dans une requête ! | |
| LIMIT | restreint le résultat à un certain nombre de ligne  retourne les lignes 1 à 10  retourne les lignes 5 à 14  ATTENTION: syntaxe spé. MySQL | SELECT … ORDER BY maColonneInt LIMIT 10;  SELECT … ORDER BY maColonneInt LIMIT 10 OFFSET **4**;  *SELECT … ORDER BY maColonneInt LIMIT* ***4****, 10;* | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| WHERE LIKE | LIKE permet des recherches plus précises sur le contenu des chaines de caractères  '%' = n'importe quelle chaine (même de longueur 0)  '\_' représente un seul caractère  BINARY rend la recherche sensible à la casse | SELECT \* FROM maTable  WHERE maColonneVarChar LIKE '%Litre%';  WHERE maColonneVarChar NOT LIKE '%Litre%';  WHERE maColonneVarChar LIKE BINARY '%Litre%'; | | *=> chaine contenant le mot '****l****itre' au singulier ou au pluriel, en fin de phrase ou non.*  *=> chaine ne contenant pas le mot '****l****itre'*  *=> chaine contenant le mot '****L****itre' avec une majuscule !*  *Exemple de recherche de chaine :*  'b%' = commençant pas b  'b\_' = **égale à 2 caractères** et commençant par b  '%b\_%' = contenant un mot de 2 ca. commençant par b  '%ch%ne' = contenant ch et finissant pas ne | | | | |  | |
| WHERE BETWEEN | BETWEEN permet de définir facilement et plus lisiblement un intervalle numérique ou temporel  mais aussi avec l'ordre alphabétique  insensible à la casse  sensible à la casse | SELECT \* FROM Animal  WHERE maColonneInt BETWEEN 50 AND 200;  WHERE maColonneDate BETWEEN '2009-01-01' AND '2010-12-31';  WHERE maColonneVarChar BETWEEN 'a' AND 'd';  WHERE maColonneVarChar BETWEEN BINARY 'A' AND BINARY 'D'; | | *équivalent à*  WHERE maColonneInt >= 50 AND maColonneINT <= 200; | | | | |  | |
| WHERE IN | IN permet d'effectuer une recherche à partir d'une liste de valeurs | SELECT \* FROM maTable  WHERE maColonneVarChar IN ('Valeur1', 'Valeur2', 'Valeur3');  WHERE maColonneInt IN (2, 4, 8, 16);  WHERE maColonneDate NOT IN ('2009-01-01', '2009-02-01'); | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Index | = liste ordonnée d'une partie de la table permettant une recherche accélérée des résultats | il faut créer un/des **index sur les colonnes qui nous serviront le plus** à la recherche/récupération de données  Mais index = mémoire + ralentissement lors de l'ajout des données (MàJ de l'index)  => utilisation intelligente des **"index par la gauche"** (cad dans l'index (colA, colB, colC) se cachent déjà les index (colA) et (colA, colB). On a donc 3 en 1 !) | | | | | | | | |
| INDEX  KEY | création d'un index sur la/les colonnes concernées lors de la création de la table | CREATE TABLE maTable **(**  maColonneInt INT UNSIGNED NOT NULL AUTO\_INCREMENT,  maColonneVarChar VARCHAR(100),  maColonneDate DATE NOT NULL DEFAULT '2000-01-01',  maColonneTxt TEXT,  PRIMARY KEY(maColonneInt),  INDEX monIndexC23 (maColonneVarChar(20), maColonneDate),  KEY monIndexC4 (maColonneTxt(20))  **)** ENGINE=INNODB**;** | | | |  | | | ATTENTION : pour les index "simples" et UNIQUE sur les colonnes de type TEXT, BLOB ou VARCHAR( ) longs, il faut préciser le nb de caractère à prendre en compte (20 à 30 max) | |
| ADD INDEX  CREATE INDEX | création d'un index sur une table déjà existante | ALTER TABLE maTable  ADD INDEX monIndexC2(maColonneVarChar(20));  CREATE INDEX monIndexC2  ON maTable (maColonneVarChar(20)); | | | |  | | |  | |
| DROP INDEX | supprimer un index | ALTER TABLE maTable  DROP INDEX monIndexC22; | |  | | | | |  | |
| UNIQUE | ajoute une contrainte à la/aux colonne(s) dont la combinaison ne peux être 2 fois la même | CREATE TABLE maTable **(**  …  PRIMARY KEY(maColonneInt),  UNIQUE INDEX monIndexC23 (maColonneVarChar(20), maColonneDate),  FULLTEXT INDEX monIndexC4 (maColonneTxt)  **)** ENGINE=INNODB**;** | | | | CREATE UNIQUE INDEX monIndexC2  ON maTable(maColonneVarChar(20)); | | |  | |
| FULLTEXT | utile pour des recherches puissantes sur les chaines, valable uniquement sur les colonnes de type **CHAR**, **VARCHAR** et **TEXT** | CREATE FULLTEXT INDEX monIndexC2  ON maTable(maColonneVarChar); | | | ATTENTION : contrairement aux index "simples" et UNIQUE, l'utilisation des "index par la gauche" n'est pas valable avec FULLTEXT, il faut créer autant d'index que de colonne ou combinaison de colonne nécessaire ! | |
| SPATIAL | index spécifique aux données spatiales (point, lignes, polygones, …) |  | | | |  | | |  | |
|  |  |  | |  | | | | |  | |
| MATCH ( ) AGAINST ( ) | faire une recherche "naturelle" sur un FULLTEXT | SELECT \* FROM maTable  WHERE MATCH (maColonneTxt) AGAINST ('valeur recherchée'); | | IN NATURAL LANGUAGE MODE *par défaut* | | | | | ATTENTION : ce type de recherche FULLTEXT découpe la chaine recherchée en mot (séparés par tout sauf ' et \_) et cherche les mots exacts dans la/les colonnes indexées en ignorant les mots < 4 ca., mots communs en anglais (about, after, once, under, …), mots rencontrés dans au moins la moitié des lignes.  Rq: par défaut les résultats sont triés par pertinence | |
|  | faire une recherche avec extension de requête | SELECT \* FROM maTable  WHERE MATCH (maColonneTxt) AGAINST ('valeur recherchée' WITH QUERY EXPANSION); | | *Exemple :*  SELECT \* FROM Livres WHERE MATH (titre, auteur) AGAINST ('Daniel' WITH QUERY EXPANSION);  *=> lignes 1 à 3 = 1re recherche, ligne 4 = 2e rech.*   |  |  | | --- | --- | | **Daniel** Pennac | Comme un roman | | **Daniel** Pennac | Au **bonheur** des ogres | | **Daniel** Penac | La Fée Carabine | | Jacqueline Harpman | Le **bonheur** dans le crime | | | | | | Rq: correspond à une recherche naturelle dont les résultats de la 1re recherche (les mots des textes correspondant à la 1re réponse) sont utilisées pour effectuer une 2e recherche naturelle et élargir la réponse avec des résultats associés. | |
|  | faire une recherche avec booléens | SELECT \* FROM maTable  WHERE MATCH (maColonneTxt) AGAINST ('valeur recherchée' IN BOOLEAN MODE); | | *Conditions sur la chaine de caractère à rechercher :*   |  |  | | --- | --- | | +abcd | doit contenir le mot abcd | | -abcd | ne doit pas contenir le mot abcd | | "abcd efg" | doit contenir exactement le groupe de mot "abcd efg" dans cet ordre | | abc\* | doit contenir un mot commençant par abc |   *Exemple :*  AGAINST ('+petit\* -"mère-grand" IN BOOLEAN MODE);  => textes contenant "petit", "petits", "petite" ou "petites" ET ne contenant pas le groupe de mot "mère-grand" | | | | | Rq: peut se faire sur colonnes non indexées, mais plus lent. Ignore les mots < 4 lettres. | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Clés |  |  | |  | | | | |  | |
| PRIMARY KEY( ) | défini une clé primaire (à partir des colonnes passées en paramètre) qui servira d'identifiant pour chaque ligne (=> combinaison unique et pas de valeur NULL) | CREATE TABLE IF NOT EXISTS maTable1 **(**  maColonne1A INT UNSIGNED AUTO\_INCREMENT,  maColonne1B VARCHAR(30) DEFAULT 'MaValeurDefaut',  maColonne1C DATE NOT NULL DEFAULT '2000-01-01',  maColonne1D TEXT,  PRIMARY KEY(maColonne1)  **) ENGINE=InnoDB;** | | *Définition après création de la table :*  ALTER TABLE maTable1  ADD PRIMARY KEY (maColonne1);  *Suppression de la clé primaire :*  ALTER TABLE maTable  DROP PRIMARY KEY; | | | | | Rq: inutile de créer un index pour les clés primaire, il se crée automatiquement.  ATTENTION : il ne peut y avoir qu'une seule clé primaire par table ! Il FAUT une clé primaire pour chaque table ! | |
| FOREIGN KEY( )  REFERENCES | définit une clé étrangère, cad une référence d'une table vers une autre | CREATE TABLE IF NOT EXISTS maTable2 **(**  maColonne2A INT UNSIGNED AUTO\_INCREMENT,  maColonne2B INT UNSIGNED,  maColonne2C TEXT,  PRIMARY KEY(maColonne2A),  CONSTRAINT maCleEtrangere  FOREIGN KEY (maColonne2B)  REFERENCES maTable1 (maColonne1A)  **) ENGINE=InnoDB;** | | *Définition après création de la table :*  ALTER TABLE maTable2  ADD CONSTRAINT maCleEtrangere  FOREIGN KEY (maColonne2B)  REFERENCES maTable1 (maColonne1A);  *Suppression de la clé primaire :*  ALTER TABLE maTable2  DROP FOREIGN KEY maCleEtrangere; | | | | | Rq: la création d'une clé étrangère sur une/plusieurs colonne(s) => création automatique d'un index sur cette/ces colonne(s).  ATTENTION : en revanche, la/les colonnes à qui la clé font référence dans l'autre table doivent posséder un index !  ATTENTION : les colonnes à qui on attribue la clé étrangère doivent être au même nb et du même type (dans le même ordre) que les colonnes de références dans l'autre table  ATTENTION : le moteur MyISAM ne gère pas les clés étrangères, contrairement à InnoDB. | |
| ON DELETE | option facultative précisant le comportement lors de la suppression d'une donnée utilisée comme référence dans une/d'autres table(s)  annule la requête et affiche msg erreur  idem  remplace la référence par NULL  supprime toutes les lignes qui y font référence | …  CONSTRAINT maCleEtrangere  FOREIGN KEY (maColonne2B)  REFERENCES maTable1 (maColonne1A)  ON DELETE RESTRICT  RESTRICT (defaut)  NO ACTION  SET NULL  CASCADE | |  | | | | | Rq: RESTRICTED et NO ACTION ont le même comportement en MySQL, mais un comportement différent dans d'autres SGBD.  ATTENTION : l'utilisation du mode CASCADE est très risquée car peut supprimer beaucoup de donnée involontairement ! A utiliser avec précaution ! | |
| ON UPDATE | option facultative précisant le comportement lors de la modification d'une donnée utilisée comme référence dans une/d'autres table(s) (uniquement si la modification porte sur la colonne servant de référence)  annule la requête et affiche msg erreur  idem  remplace la référence par NULL  **MàJ des lignes avec la nouvelle valeur** | …  CONSTRAINT maCleEtrangere  FOREIGN KEY (maColonne2B)  REFERENCES maTable1 (maColonne1A)  ON UPDATE RESTRICT  RESTRICT (defaut)  NO ACTION  SET NULL  **CASCADE** | |  | | | | |  | |
| DROP FOREIGN KEY | supprime une clé étrangère | ALTER TABLE maTable  DROP FOREIGN KEY maCleEtrangere; | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Jointures | rassemblement des informations de deux tables (liées par une ou plusieurs colonnes) en une table temporaire sur laquelle on peut appliquer des requêtes |  | | *voir image* SQL-Jointure pour plus de précision entre **INNER JOIN**, **OUTER JOIN** et **FULL JOIN** | | | | |  | |
| INNER JOIN | jointure interne  = ne tiens pas compte des infos si NULL de part ou d'autre de la jointure  jointure interne sur plusieurs colonnes  jointure sur plusieurs tables | SELECT \*  FROM maTable1  INNER JOIN maTable2  ON maTable1.maColonne1A = maTable2.maColonne2B  WHERE …  ORDER BY …  LIMIT …  ;  SELECT \* FROM maTable1  **INNER JOIN** maTable2  ON maTable1.maColonne1A = maTable2.maColonne2B  AND maTable1.maColonne1D = maTable2.maColonne2J  …  SELECT \* FROM maTable1  INNER JOIN maTable2  ON maTable1.maColonne1A = maTable2.maColonne2B  INNER JOIN maTable3  ON maTable1.maColonne1D = maTable3.maColonne3A  … | | | *Obtenir info dans la table2 à partir de la sélection sur la table 1 en une seule requête :*  SELECT maTable2.maColonne2C  FROM maTable2  INNER JOIN maTable1  ON maTable1.maColonne1A = maTable2.maColonne2B  WHERE maTable1.id = 3;  *Exemple :*  SELECT Animal.nom AS nom\_animal, Race.nom AS race  FROM Animal  INNER JOIN Race  ON Animal.race\_id = Race.id  WHERE Animal.espece\_id = 2  ORDER BY Race.nom, Animal.nom; | | | | Rq: jointure interne => les chats (espece\_id = 2) ne présentant pas de race (race\_id = NULL car batard) ne s'afficherons pas. | |
|  |  | *Exemple de jointure et jointure interne avec des alias :*  SELECT Espece.nom\_courant AS espece, Animal.nom AS nom\_animal, Race.nom AS race\_animal,  Pere.nom AS papa, Race\_pere.nom AS race\_papa,  Mere.nom AS maman, Race\_mere.nom AS race\_maman  FROM Animal  INNER JOIN Espece -> jointure avec Espece  ON Animal.espece\_id = Espece.id  INNER JOIN Race -> jointure avec Race  ON Animal.race\_id = Race.id  **LEFT** JOIN Animal AS Pere -> auto-jointure avec Animal sur colonne pere\_id  ON Animal.pere\_id = Pere.id  **LEFT** JOIN **Race** AS Race\_pere -> sous-jointure de Pere (table Animal) avec Race  ON Pere.race\_id = Race\_pere.id  INNER JOIN Animal AS Mere -> auto-jointure avec Animal sur colonne mere\_id  ON Animal.mere\_id = Mere.id  INNER JOIN Race AS Race\_mere -> sous-jointure de Mere (table Animal) avec Race  ON Mere.race\_id = Race\_mere.id; | | | | | | | 3 tables :   * Animal incluant les colonnes espece\_id, race\_id, pere\_id et mere\_id * Espece * Race   INNER JOIN => sélectionne uniquement les animaux dont on connait l'espèce, la race, la mère et la race de la mère.  LEFT JOIN => sélectionne les animaux même si on ne connait pas le père et ceux dont on connait le père même si on ne connait pas sa race.   * Affichage du tableau : espèce, nom, race, nom du père, race du père, nom de la mère, race de la mère | |
| LEFT JOIN  LEFT OUTER JOIN | jointure externe par la gauche  = prenant en compte les éléments de gauche n'ayant pas de correspondance dans la table de droite (la table jointe) | SELECT \*  FROM maTable1 *-> table de gauche*  LEFT JOIN maTable2 *-> table de droite (jointe)*  ON maTable1.maColonne1A = maTable2.maColonne2B  WHERE …  ORDER BY …  LIMIT …  ; | | *Exemple :*  SELECT Animal.nom AS nom\_animal, Race.nom AS race  FROM Animal  LEFT OUTER JOIN Race  ON Animal.race\_id = Race.id  WHERE Animal.espece\_id = 2  ORDER BY Race.nom, Animal.nom; | | | | | Rq: jointure externe => les chats (espece\_id = 2) ne présentant pas de race s'afficherons dans la liste avec race = NULL. | |
| RIGHT JOIN  RIGHT OUTER JOIN | jointure externe par la droite  = prenant en compte les éléments de droite (table jointe) n'ayant pas de correspondance dans la table de gauche | SELECT \*  FROM maTable1 *-> table de gauche*  RIGHT JOIN maTable2 *-> table de droite (jointe)*  ON maTable1.maColonne1A = maTable2.maColonne2B  WHERE …  ORDER BY …  LIMIT …  ; | | *Exemple :*  SELECT Animal.nom AS nom\_animal, Race.nom AS race  FROM Animal  RIGHT OUTER JOIN Race  ON Animal.race\_id = Race.id  WHERE Race.espece\_id = 2  ORDER BY Race.nom, Animal.nom; | | | | | Rq: jointure externe => toutes les espèces de chats s'afficheront, y compris celles qui sont dans la liste Race mais qui n'ont pas encore été attribuées à des animaux de la table Animal | |
| USING | si la jointure (interne ou externe) se fait sur une/des colonne(s) ayant le même nom dans les 2 tables, alors on peut simplifier la syntaxe avec USING | SELECT \*  FROM maTable1  INNER|LEFT|RIGHT JOIN maTable2  USING (maColonneX)  WHERE …  ; | | *Sur plusieurs colonnes :*  SELECT \*  FROM maTable1  INNER|LEFT|RIGHT JOIN maTable2  USING (maColonneX, maColonneY)  WHERE …  ; | | | | |  | |
| NATURAL | si toutes les colonnes (et uniquement celles-ci) servant à la jointure ont le même nom dans les 2 tables, alors on peut réaliser une jointure naturelle sans avoir à préciser les colonnes concernées (elles seront déterminées automatiquement) | SELECT \*  FROM maTable1  NATURAL JOIN maTable2  WHERE …  ; | | *Jointure naturelle par la gauche :*  SELECT \*  FROM maTable1  NATURAL LEFT JOIN maTable2  WHERE …  ; | | | | |  | |
| UPDATE INNER JOIN SET | modification de données à partir des données d'une autre table grâce à une jointure | UPDATE maTable1  INNER JOIN maTable2  ON maTable1.maC1D = maTable2.maC2J  SET maTable1.maC1E = maTable2.maC2L  WHERE … ; | |  | | | | |  | |
| DELETE FROM INNER JOIN | suppression de données sélectionnées à partir des données d'une autre table grâce à une jointure | DELETE **maTable1** *-> précise la table où les données doivent être supprimées*  FROM maTable1  INNER JOIN maTable2  ON maTable1.maC1D = maTable2.maC2J  WHERE maTable1.maC1E = maTable2.maC2L; | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Sous requêtes |  |  | |  | | | | |  | |
| SELECT FROM( ) AS | Table intermédiaire  sélectionne un résultat parmi les résultats déjà sélectionnés par la sous requête  entre les ( ) = requête normale  utilisation d'un Alias pour nommer la table intermédiaire OBLIGATOIRE avec FROM ! | SELECT MIN(maSousTable.**maColonne3**)  FROM (  SELECT maTable1.maColonne1, maTable1.**maColonne3**  FROM maTable  INNER JOIN …  ON …  WHERE …  ) AS maSousTable; | | | *Multiple sous-requêtes :*  SELECT …  FROM (  SELECT …  FROM (  SELECT …  FROM maTable1  …  ) AS maSousTableA  …  ) AS maSousSousTableB; | | | | ATTENTION : aucune table intermédiaire ne doit contenir des colonnes ayant le même nom (par exemple : SELECT maTable1.id, maTable2.id). Dans ce cas il faut renommer l'une des 2 avec un alias. | |
| WHERE <= ( ) | **Valeur intermédiaire**  si la sous-requête ne renvoie qu'une seule valeur (1 ligne) alors on peut l'utiliser directement avec un opérateur de comparaison pour sélectionner les données dans la requête principale | *Intermédiaire renvoie une seule valeur :*  SELECT maColonne1A, maColonne1B, maColonne1C  FROM maTable1  WHERE maColonne1D >= (  SELECT MIN(maColonne2A)  FROM maTable2);  *Intermédiaire renvoie une seule ligne :*  SELECT maColonne1A, maColonne1B, maColonne1C  FROM maTable1  WHERE (**maColonne1D, maColonne1E**) = (  **SELECT maColonne2A, maColonne2C**  FROM maTable2  WHERE id = 7); | | *Exemple :*  SELECT id, nom, espece\_id  FROM Race  WHERE espece\_id = (  SELECT MIN(id) FROM Espece);  *-> id min de Espece = 1 (= chien) => renvoi la table de toutes les races de chien ayant le même espece\_id = 1* | | | | | ATTENTION : si résultat de la sous-requête = 1 seule valeur (1 colonne, 1 ligne), tous les opérateurs marchent.  Mais si résultat renvoie une ligne avec plusieurs colonnes, alors seuls =, != et <> fonctionnent (pas <, <=, >, >=). | |
| WHERE IN ( ) | Colonne intermédiaire  si la sous-requête renvoie une seule colonne, alors on peut utiliser les valeurs de celle-ci dans une sélection via une liste (IN, NOT IN, < ANY/SOME et < ALL) | SELECT maColonne1A, maColonne1B, maColonne1C  FROM maTable1  WHERE maColonne1B IN (  **SELECT maColonne2A**  FROM maTable2  WHERE maColonne2C > 4); | | ANY/SOME *=> à au moins une des valeurs*  ALL *=> à toutes les valeurs*  SELECT …  FROM …  WHERE maColonne1B >= ANY (  SELECT maColonne2A  … ); | | | | | Rq: ANY/SOME et ALL ne peuvent s'utiliser qu'avec des sous-requêtes. | |
| WHERE EXISTS  WHERE NOT EXISTS | renvoie un résultat uniquement si la sous-requête a (ou n'a pas) renvoyé de résultat | SELECT \*  FROM maTable1  WHERE NOT EXIST (  SELECT \*  FROM maTalbe2  WHERE maTable1.maColonne1B= maTable2.maColonne2C); | |  | | | | |  | |
|  | en liant les données des deux requêtes dans le dernier AND de la sous requête, il ne renverra que les lignes qui sont en commun, cad les lignes qui ont été sélectionnées dans la sous requête et qui sont également présentent dans la requête principale | SELECT DISTINCT NO\_COMMANDE  FROM details\_commandes AS d1  INNER JOIN produits  ON d1.REF\_PRODUIT = produits.REF\_PRODUIT  WHERE produits.CODE\_CATEGORIE = 1 AND produits.NO\_FOURNISSEUR = 1  AND EXISTS(  SELECT NO\_COMMANDE  FROM details\_commandes AS d2  INNER JOIN produits  ON d2.REF\_PRODUIT = produits.REF\_PRODUIT  WHERE produits.CODE\_CATEGORIE = 2 AND produits.NO\_FOURNISSEUR = 2  AND d2.NO\_COMMANDE = d1.NO\_COMMANDE ); | | | | | | |  | |
| INSERT SELECT FROM | insertion de données à partir de données d'une autre table | INSERT INTO maTable1 (maC1B, maC1C, maC1D)  SELECT 'ma valeur 1B', **maC2D**, **maC2J**  FROM **maTable2**  WHERE maC2B = 'ma valeur 2B'; | | *Exemple :*  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'; | | | | |  | |
| UPDATE (SELECT FROM ) | modification de données à partir des données d'une autre table grâce à une sous-requête | *Sélection à partir des données d'une autre table :*  UPDATE maTable1 SET maC1B = 'nouvelle valeur'  WHERE maC1D =  (SELECT maC2J  FROM maTable2  WHERE maC2E = 'valeur 2E');  *Modification en attribuant la valeur d'une autre table :*  UPDATE maTable1 SET maC1D =  (SELECT maC2J  FROM maTable2  WHERE maC2E = 'valeur 2E')  WHERE maC1B = 'valeur 1B'; | |  | | | | | ATTENTION : avec une sous-requête, on ne peut pas modifier le contenu d'une table qui nous sert elle-même de table de référence pour la nouvelle valeur, cad que ceci retourne une erreur :  UPDATE maTable1 SET maC1B =  (SELECT maC1F  FROM maTable1  WHERE …)  … ; | |
| DELETE FROM  (SELECT FROM ) | suppression de données sélectionnées à partir des données d'une autre table grâce à une sous-requête | DELETE FROM maTable1  WHERE maC1D =  (SELECT maC2J  FROM maTable2  WHERE maC2E = 'valeur 2E'); | |  | | | | | ATTENTION : idem que ligne précédente : on ne peut supprimer une ligne d'une table qui sert de réf dans une sous-requête. | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Union de requêtes | réunir les résultats de plusieurs requettes |  | |  | | | | |  | |
| UNION | permet de rassembler les résultats de 2 ou plusieurs requêtes en un seul tableau  Rq: les doublons sont automatiquement effacés et n'apparaissent pas dans les résultats | SELECT maC1A, maC1B, maC1C  FROM maTable1  WHERE …  UNION  SELECT maC2A, maC2B, maC2C  FROM maTable2  WHERE …  UNION  … ; | |  | | | | | ATTENTION : le nombre et le type des colonnes (ainsi que l'ordre) doit être le même pour toutes les requêtes unifiées ! Sinon : soit erreur, soit résultats incohérents (nb transformé en chaine de caractères, …)  Rq: les titres des colonnes dans le tableau de résultat seront ceux de la première requête. | |
| UNION ALL | permet de rassembler les résultats de 2 ou plusieurs requêtes en un seul tableau  en conservant les doublons | SELECT maC1A, maC1B, maC1C  FROM maTable1  WHERE …  UNION ALL  SELECT maC2A, maC2B, maC2C  FROM maTable2  WHERE … ; | |  | | | | | ATTENTION : impossible de mélanger UNION et UNION ALL dans une même requête. Dans ce cas, ils seront tous considérés comme UNION ! | |
| ORDER BY | trier les résultats | SELECT … FROM … WHERE …  UNION  SELECT … FROM … WHERE …  ORDER BY **maCB** ;  SELECT … FROM … WHERE … **ORDER BY** maC1D **LIMIT** 5  UNION  (SELECT … FROM … WHERE … **ORDER BY** maC2E **LIMIT** 5)  ORDER BY **maCB** ; | | *Exemple :*  SELECT id, **nom**, 'Race' AS table\_origine FROM Race  UNION  SELECT id, nom\_latin, 'Espèce' AS table\_origine FROM Espece  ORDER BY **nom** DESC; | | | | | Rq: le nom de la colonne utilisé pour le tri est celui défini dans la première requête, qu'importe le nom de la même colonne dans les suivantes.  ATTENTION : on ne peut pas effectuer de tri sur les requêtes intermédiaires avec les unions, seulement sur la table finale.  Sauf si ce tri intermédiaire est associé à un **LIMIT** car il va alors déterminer les données sélectionnées ! | |
| LIMIT | définit un nb total limite de lignes à renvoyer pour l'ensemble des requêtes unies | SELECT … FROM … WHERE …  UNION  SELECT … FROM … WHERE …  LIMIT 25; | | *Définir une limite pour chaque requête :*  SELECT … FROM … WHERE … LIMIT 5  UNION  (SELECT … FROM … WHERE … LIMIT 5); | | | | | ATTENTION : le dernier LIMIT s'applique à la réunion des requêtes. Pour définir une limite uniquement à la dernière sélection, on doit ajouter des parenthèses ! | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| Fonctions |  |  | |  | | | | |  | |
| VERSION() | renvoi la version du moteur SQL en cours de fonctionnement | SELECT VERSION(); | |  | | | | |  | |
| USER()  CURRENT\_USER() | renvoie des infos sur l'utilisateur avec lequel on est connecté à la BDD | SELECT USER();  SELECT CURRENT\_USER(); | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| LAST\_INSERT\_ID() | renvoie le dernier ID créé par auto-incrémentation dans la BDD par l'utilisateur courant | INSERT INTO maTableA(maC2, maC3)  VALUES(valeur2, valeur3);  INSERT INTO maTableB(maC2, maC3, maC4)  VALUES(valeur2, valeur3, LAST\_INSERT\_ID()); | |  | | | | |  | |
| FOUND\_ROWS()  SQL\_CALC\_FOUND\_ROWS | renvoie le nb de ligne obtenues lors de la dernière requête  associé à cette option dans SELECT : renvoie le nb de ligne qu'aurait rapportée la requête sans le LIMIT | SELECT FOUND\_ROWS(); | | **SELECT** SQL\_CALC\_FOUND\_ROWS maC1, maC2  FROM maTable  LIMIT 3;  SELECT FOUND\_ROWS(); | | | | |  | |
| CAST( AS ) | cast un type en un autre | CAST(maValeur AS MON\_TYPE) | | CAST('190112' AS DATE); => '2019-01-12' | | | | |  | |
| COALESCE( , …) | prend un nb indéfini de paramètre et renvoie la première valeur non-nulle parmi eux | COALESCE(maCol2, maCol7, maCol4, 'valeur par défaut') | | *ex: une table 'animal' avec un prix de base pour chaque espèce (chien, chat, …) et un prix plus élevé s'il est de race (Labrador, Cocker, …)*  SELECT SUM(COALESCE(Race.prix, Espece.prix)) | | | | |  | |
|  |  |  | |  | | | | |  | |
| Fonctions scalaires | s'appliquent sur chaque ligne |  | |  | | | | |  | |
|  | **nombres** |  | |  | | | | |  | |
| CEIL( )  FLOOR( ) | arrondit au nb entier supérieur  arrondit au nb entier inferieur | CEIL(monNombre)  FLOOR(monNombre) | |  | | | | |  | |
| ROUND( )  ROUND( , ) | arrondit à l'entier le plus proche  arrondit au nombre de décimale le plus proche | ROUND(monNombre)  ROUND(monNombre, 2) *=> arrondit à 2 décimales* | |  | | | | |  | |
| TRUNCATE( , ) | tronque au nombre de décimale indiqué en 2e param | TRUNCATE(monNombre, 0) *=> tronque à l'entier*  TRUNCATE(monNombre, 2) *=> tronque à 2 décimales* | |  | | | | |  | |
| POW( , )  POWER( , ) | puissance | POW(monNombre, 2) | |  | | | | |  | |
| SQRT( ) | racine carré | SQRT(monNombre) | | *racine ne :*  POW(monNombre, 1/n); | | | | | Rq: il n'existe pas de fonction pour la racine ne. Il faut utiliser la fonction POW comme ci-contre. | |
| RAND() | nombre aléatoire entre 0 et 1 | RAND() | | *trier les résultats de manière aléatoire :*  SELECT \*  FROM maTable  ORDER BY RAND(); | | | | |  | |
| SIGN( ) | renvoie  -1 si n < 0 | 0 si n = 0 | 1 si n > 0 | SIGN(monNombre) | |  | | | | |  | |
| ABS( ) | valeur absolue | ABS(monNombre) | |  | | | | |  | |
| MOD( , ) | modulo | MOD(monNombre, 10) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  | **caractères** |  | |  | | | | |  | |
| ASCII( ) | renvoie le code ASCII du premier ca. de la chaine | ASCII(maChaine)  ASCII('T') => 84 | |  | | | | |  | |
| CHAR( , …) | renvoie le/les ca. correspondants aux codes ASCII passés en param | CHAR(84, 84+32, '84') => TtT | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  | **Chaines de caractères** |  | |  | | | | |  | |
| BIT\_LENGTH( )  LENGTH( ) | renvoie le nb de bit de la string  renvoie le nb d'octet de la string | BIT\_LENGTH(maChaine)  LENGTH(maChaine) | |  | | | | |  | |
| CHAR\_LENGTH( ) | renvoie le nd de caractères de la string | CHAR\_LENGTH(maChaine) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| LOWER( )  LCASE( ) | passe toute la chaine en minuscule | LOWER(maChaine) | |  | | | | |  | |
| UPPER( )  UCASE( ) | passe toute la chaine en majuscule | UPPER(maCHaine) | |  | | | | |  | |
| REVERSE( ) | inverse les ca. de la chaine | REVERSE(maChaine) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| STRCMP( , ) | compare 2 strings en renvoyant :   * -1 si 1re string avant la 2e * 0 si strings identiques * 1 si 1re string après la 2e   (comme compareTo() en Java) | STRCMP(maString1, maString2) | |  | | | | |  | |
| FIELD( , …) | compare le 1er param avec tous les suivants et renvoie l'index du premier param qui lui est égal, 0 sinon | FIELD('Salut', 'Salut !', 'Bonjour', 'Salut', 'Bonsoir')  *=> retourne 3* | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| CONCAT( , …)  CONCAT\_WS( , …) | concaténation de toutes les chaines passées en paramètres  le 1er param de CONCAT\_WS est utilisé comme séparateur | CONCAT(maChaine1, maChaine2, maChaine3)  CONCAT\_WS(' | ', maChaine1, maChaine2, maChaine3) | |  | | | | |  | |
| REAPEAT( , ) | renvoie le texte concaténé n fois | REPEAT(maChaine, 5) | |  | | | | |  | |
| TRIM( ) | renvoie une string en éliminant les ca. inutiles à l'avant et/ou à l'arrière  à l'avant et à l'arrière  seulement à l'avant  seulement à l'arrière | élimine les **espaces** à l'avant et à l'arrière  élimine les espaces seulement à l'avant  élimine les **'@-'** à l'avant et à l'arrière  élimine les **'@-'** seulement à l'arrière  BOTH (défaut)  LEADING  TRAILING | | TRIM(maChaine)  TRIM(LEADING FROM maChaine)  TRIM(**'@-'** FROM maChaine)  TRIM(TRAILING **'@-'** FROM maChaine) | | | | |  | |
| LEFT( , )  RIGHT( , ) | renvoie seulement les n ca. de la partie gauche | droite de la chaine | LEFT(maChaine, 20) | |  | | | | |  | |
| LPAD( , , )  RPAD( , , ) | renvoie une string avec une longueur définie en tronquant la string de base si trop longue ou en rajoutant un caractère (par la gauche | par la droite) si trop courte   * param 1 = texte * param 2 = longueur voulue * param 3 = string à rajouter | *complétée par la gauche :*  LPAD(maChaine, 50, '@--')  *complétée par la droite :*  LPAD(maChaine, 100, ' '); | |  | | | | |  | |
| SUBSTRING( , )  SUBSTRING( , , ) | renvoie une sous partie de la chaine   * param1 = texte * param2 = position du 1er ca. inclus * *param3 facult = longueur de la chaine* | retourne la fin de la chaine à partir du 5e ca. inclus :  SUBSTRING(maChaine, 5)  retourne la chaine entre les ca. 5 et 15 :  SUBSTRING(maChaine, 5, 10) | |  | | | | | ATTENTION : en SQL le 1er ca. dans une chaine a pour index 1 ! | |
| POSITION( )  INSTR( , )  LOCATE( , ) | recheche une sous-chaine dans une chaine de ca. et retourne la position du premier ca. si trouvée, 0 sinon.  Rq: LOCATE peut prendre un 3e param facultatif : la position à partir de laquelle faire la recherche dans la chaine | POSITION(maChRef IN maChaine)  INSTR(maChaine, maChRef)  LOCATE(maChRef, maChaine)  LOCATE(maChRef, maChaine, 10) | |  | | | | |  | |
| INSERT( , , , )  REPLACE( , , ) | remplace une partie de la chaine | INSERT(maChaine, position, longueur, chaineAInserer)  REPLACE(maChaine, chaineARemplacer, chaineAInserer) | | INSERT(maChaine, 8, 12, '')  REPLACE(maChaine, 'ex:', 'exemple :') | | | | |  | |
|  |  |  | |  | | | | |  | |
| Fonctions d'agrégations | traitement portant sur plusieurs lignes mais ne retournant qu'une seule valeur |  | |  | | | | |  | |
| COUNT(\*)  COUNT( ) | compte le nombre de ligne  compte le nb de valeur **non NULL** | SELECT COUNT(\*) FROM maTable;  SELECT COUNT(maColonne2) FROM maTable; | | SELECT COUNT(DISTINCT maColonne2)  FROM maTable; | | | | |  | |
| MIN( )  MAX( ) | renvoie la valeur minimale | maximale de la colonne  Rq: s'utilise sur tous les types (nb, chaine, booléen, date, …) | SELECT MIN(maColonne2)  FROM maTable; | | *cas particulier avec les booléens :*  *agrégation && (ET)*  **MIN(CAST(maColBool1 as INT))** as atLeastOneFalse  *agrégation || (OU)*  **MAX(CAST(maColBool2 as INT))** as allTrue | | | | | Rq: en BDD, boolean = BIT => false = 0 et true = 1  => ET (si au moins une des valeurs est false, alors le résultat est false)  => OU (si au moins une des valeurs est true, alors le résultat est true) | |
| SUM( ) | calcule la somme des valeurs d'une colonne | SELECT SUM(maColonne2)  FROM maTable; | |  | | | | |  | |
| AVG( ) | calcule la moyenne des valeurs d'une colonne | SELECT AVG(maColonne2)  FROM maTable; | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| GROUP\_CONCAT( ) | concatène les valeurs d'une colonne | SELECT GROUP\_CONCAT(maColonne2)  FROM maTable;  SELECT GROUP\_CONCAT(maC2, maC3)  FROM maTable; | | *en ajoutant des chaines de ca. :*  SELECT GROUP\_CONCAT(maC2, **' ('**, maC3, **') '**)  FROM maTable; | | | | |  | |
| DISTINCT  ORDER BY  SEPARATOR | options facultatives pouvant être utilisées dans GROUP\_CONCAT | SELECT GROUP\_CONCAT(  **DISTINCT** maC2  **SEPARATOR** ' - ')  FROM maTable; | | SELECT GROUP\_CONCAT(  DISTINCT maC2, maC3  **ORDER BY** maC2  SEPARATOR ' - ')  FROM maTable; | | | | |  | |
|  |  |  | |  | | | | |  | |
|  | **Dates** |  | |  | | | | |  | |
| CURDATE() | donne la date actuelle | CURDATE()  CURRENT\_DATE() | |  | | | | |  | |
| CURTIME() | donne l'heure actuelle | CURTIME()  CURRENT\_TIME() | |  | | | | |  | |
| NOW()  SYSDATE() | donne la date et l'heure actuelle | NOW()  SYSDATE() | | LOCALTIME()  CURRENT\_TIMESTAMP()  LOCALTIMESTAMP() | | | | | Rq: NOW() ou SYSDATE() peuvent servir à remplir indifféremment un champ de type DATE, TIME ou DATETIME sans erreur | |
| MAKEDATE( , ) | créer une date à partir de l'année et du jour dans l'année (entre 1 et 366) | MAKEDATE(2020, 60) | |  | | | | |  | |
| MAKETIME( , , ) | créer un time à partir de l'heure, des minutes, des secondes | MAKETIME(15, 45, 21) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| DATE( ) | extrait la partie "date" d'un DATETIME | DATE(monDateTime) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| DAY( ) | extrait le jour du mois  de 1 à 31 | DAY(monDateTime)  DAYOFMONTH(monDateTime) | |  | | | | |  | |
| DAYOFWEEK( ) | donne l'index du jour de la semaine  de 1 = Dimanche à 7 = Samedi | DAYOFWEEK(monDateTime) | |  | | | | |  | |
| WEEKDAY( ) | donne l'index du jour de la semaine  de 0 = Lundi à 6 = Dimanche | WEEKDAY(monDateTime) | |  | | | | |  | |
| DAYNAME( ) | donne le nom du jour de la semaine | DAYNAME(monDateTime) | |  | | | | |  | |
| DAYOFYEAR( ) | donne le numéro du jour dans l'année  de 1 à 366 | DAYOFYEAR(monDateTime) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| WEEK( ) | donne le numéro de la semaine  de 1 à 53 | WEEK(monDateTime)  WEEKOFYEAR(monDateTime) | |  | | | | | Rq: peuvent prendre un second paramètres pour spécifier si début de la semaine = dimanche ou lundi et la première semaine de l'année (=> voir documentation SQL) | |
|  |  |  | |  | | | | |  | |
| MONTH( ) | donne le numéro du mois  de 1 à 12 | MONTH(monDateTime) | |  | | | | |  | |
| MONTHNAME( ) | donne le nom du mois | MONTHNAME(monDateTime) | |  | | | | |  | |
| LAST\_DAY( ) | renvoie le dernier jour du mois courant sous la forme d'une date | LAST\_DAY(monDateTime) | | LAST\_DAY(2100-02-03) *=> 2100-02-28*  *donc l'année 2100 ne sera pas bissextile* | | | | |  | |
|  |  |  | |  | | | | |  | |
| YEAR( ) | extrait l'année (à 4 chiffres) | YEAR(monDateTime) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| TIME( ) | extrait la partie "heure" d'un DATETIME | TIME(monDateTime) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| HOUR( ) | extrait l'heure | HOUR(monDateTime) | |  | | | | |  | |
| MINUTE( ) | extrait les minutes | MINUTE(monDateTime) | |  | | | | |  | |
| SECOND( ) | extrait les secondes | SECOND(monDateTime) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  | **formatage** |  | |  | | | | |  | |
| SET lc\_time\_names | indique le langage à utiliser pour restituer le nom des mois et des jours | SET lc\_time\_names = 'fr\_FR'; | |  | | | | |  | |
| GET\_FORMAT( , ) | utiliser un des 5 formats standards :  'ISO', 'USA', 'EUR', 'JIS', 'INTERNAL' | GET\_FORMAT(DATETIME, 'ISO')  GET\_FORMAT(TIME, 'EUR') | | DATE\_FORMAT(maDate, GET\_FORMAT(DATE, 'ISO')) | | | | |  | |
| DATE\_FORMAT( , ) | prend un DATE, TIME ou DATETIME en param1 et renvoie une chaine formatée avec le format passé en param2 | DATE\_FORMAT(monDateTime, 'le %W %e %M %Y')  *=> retourne 'le jeudi 2 avril 2020'* | | *=> voir détails dans tableau à la fin du document* | | | | |  | |
| STR\_TO\_DATE( , ) | prend une chaine en param1 formatée au format en param2 et retourne un DATETIME | STR\_TO\_DATE('03/04/2011 à 09h17', '%d/%m/%Y à %Hh%i')  STR\_TO\_DATE('15blabla', '%Hblabla') | | => 2011-04-03 09:17:00  => 15:00:00 | | | | |  | |
|  |  |  | |  | | | | |  | |
|  | **conversion** |  | |  | | | | |  | |
| TIME\_TO\_SEC( ) | convertit un TIME en secondes | **TIME\_TO\_SEC(monTime)** | |  | | | | |  | |
| SEC\_TO\_TIME( ) | convertit un nombre de secondes en TIME | **SEC\_TO\_TIME(102554)** | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  | **calculs** |  | |  | | | | |  | |
| DATEDIFF( , ) | calcule le temps écoulé entre 2 dates   * param1: DATETIME | DATE * param2: DATETIME | DATE * retour : nb de jour | DATEDIFF(monDateTime1, monDateTime2)  DATEDIFF(maDate1, maDate2)  DATEDIFF(monDateTime1, maDate2) | |  | | | | |  | |
| TIMEDIFF( , ) | idem   * param1: DATETIME | TIME * param2: même type que param1 * retour : TIME | TIMEDIFF(monDateTime1, monDateTime2)  TIMEDIFF(monTime1, monTime2) | |  | | | | |  | |
| TIMESTAMPDIFF( , , ) | idem   * param1: unité du retour * param2: DATETIME | DATE * param3: DATETIME | DATE * retour : la valeur dans l'unité indiquée en param1 | TIMESTAMPDIFF(DAY, monDateTime1, maDate2)  SELECT NOM, PRENOM, TIMESTAMPDIFF(YEAR, DATE\_NAISSANCE, CURDATE()) AS Age  FROM users; | | |  |  | | --- | --- | | Unités | | | SECOND | WEEK | | MINUTE | MONTH | | HOURS | QUARTER *(= trimestres)* | | DAY | YEAR | | | | | |  | |
|  |  |  | |  | | | | |  | |
| INTERVAL | définit une valeur temporelle à ajouter à une date ou à un time | INTERVAL 7 DAY  INTERVAL '2-5' YEAR\_MONTH | | *=> voir détails dans tableau à la fin du document* | | | | |  | |
| ADDDATE( , ) | ajoute un intervalle de temps à une date   * param1: DATETIME | DATE * param2: INTERVAL | nb de jours | ADDDATE(maDate, INTERVAL '3 02:10:32' DAY\_SECOND)  ADDDATE(monDateTime, 14) | |  | | | | |  | |
| DATE\_ADD( , )  + | idem   * param1: DATETIME | DATE * param2: INTERVAL | DATE\_ADD(maDate, INTERVAL 3 MONTH)  ⇔  maDate + INTERVAL 3 MONTH | |  | | | | |  | |
| TIMESTAMPADD( , , ) | idem   * param1: unité (les même que TIMESTAMPSDIFF( ) ) * param2: nb * param3: DATETIME | DATE | TIMESTAMPADD(MINUTE, 35, monDateTime) | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| ADDTIME( , ) | ajoute un intervalle de temps à un time   * param1: DATETIME | TIME * param2: TIME | ADDTIME(monDateTime, monTime)  ADDTIME(NOW(), '01:00:00') AS DansUneHeure | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| SUBDATE( , ) | soustrait un intervalle de temps à une date  s'utilise comme ADDDATE( ) |  | |  | | | | |  | |
| DATE\_SUB( , ) | idem  s'utilise comme DATE\_SUB( ) |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
| SUBTIME( , ) | soustrait un intervalle de temps à un time  s'utilise comme ADDTIME( ) |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |
|  |  |  | |  | | | | |  | |

Unités disponibles pour définir un INTERVAL

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unités | | | | | |
| SECOND | WEEK | MINUTE\_SECOND | 'm:s' | DAY\_SECOND | 'J HH:mm:SS' |
| MINUTE | MONTH | HOUR\_SECOND | 'HH:mm:SS' | DAY\_MINUTE | 'J HH:mm' |
| HOURS |  | HOUR\_MINUTE | 'HH:mm' | DAY\_HOUR | 'J HH' |
| DAY | YEAR |  |  | YEAR\_MONTH | 'A-M' |

|  |  |  |
| --- | --- | --- |
| Exemples : | INTERVAL 7 DAY | INTERVAL '2-5' YEAR\_MONTH |

Formatage des dates et/ou times à utiliser dans la fonction DATE\_FORMAT( , )

|  |  |
| --- | --- |
| Symbole | Signification |
| %d | Jour du mois (2 chiffres, de 00 à 31) |
| %e | Jour du mois (1 ou 2 chiffres, de 0 à 31) |
| %D | Jour du mois, avec suffixe (1rst, 2nd,…, 31th) **en anglais** |
| %w | Numéro du jour de la semaine (de 0 = Dimanche à 6 = Samedi) |
| %W | Nom du jour de la semaine |
| %a | Nom du jour de la semaine en abrégé |
| %m | Mois (2 chiffres, de 00 à 12) |
| %c | Mois (1 ou 2 chiffres, de 0 à 12) |
| %M | Nom du mois |
| %b | Nom du mois en abrégé |
| %y | Année (2 chiffres) |
| %Y | Année (4 chiffres) |
| %r | Heure complète, format 12h (hh:mm:ss AM/PM) |
| %T | Heure complète, format 24h (hh:mm:ss) |
| %h | Heure (2 chiffres) sur 12 heures (de 00 à 12) |
| %H | Heure (2 chiffres) sur 24 heures (de 00 à 23) |
| %l | Heure (1 ou 2 chiffres) sur 12 heures (de 0 à 12) |
| %k | Heure (1 ou 2 chiffres) sur 24 heures (de 0 à 23) |
| %i | Minutes (de 00 à 59) |
| %s ou %S | Secondes (de 00 à 59) |
| %p | AM/PM |
|  |  |
| exemple | DATE\_FORMAT(monDateTime, 'Nous sommes aujourd''hui le %d %M de l''année %Y. Il est actuellement %l heures et %i minutes.') |