Fichiers et bases de données

## Gestion des utilisateurs

### Tables système

information\_schema : infos sur les tables, les colonnes, le type des colonnes et les procédures.

performance\_schema : infos sur les actions faites sur le serveur.

user : infos sur les utilisateurs et leurs privilèges.

db : privilèges par bd.

tables\_priv : privilèges au niveau des tables.

columns\_priv : privilèges au niveau des colonnes.

proc\_priv : privilèges au niveau des procédures.

### Gestion des utilisateurs et des privilèges

* Créer un utilisateur :

CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';

* Création d'un utilisateur nommé "newuser" avec un mot de passe "password".

Remarques :

* IDENTIFIED BY pas obligatoire. L'utilisateur n'a pas de mdp si rien n'est précisé.
* Mdp hashé avant d'être stocké dans mysql.user.
* Peut accepter différentes options :
* Méthodes d'authentificat° : IDENTIFIED [WITH MYSQL\_NATIVE\_PASSWORD/SHA256\_PASSWORD/CACHING\_SHA2\_PASSWORD] BY 'password';
* Rôle : DEFAULT ROLE role1, [role2]. L'utilisateur aura tous les droits liés à ce rôle. Pour créer un rôle et ses droits : CREATE ROLE role1; + GRANT SELECT ON \*.\* TO role1;
* Options WITH : MAX\_QUERIES\_PER\_HOUR, MAX\_UPDATES\_PER\_HOUR, MAX\_CONNECTIONS\_PER\_HOUR, MAX\_USER\_CONNECTIONS. Ttes suivient par une valeur (0 = pas de limite). Pour mettre +ieurs options, les mettre les unes après les autres sans les séparer par une virgule.
* PASSWORD EXPIRE INTERVAL x DAY, PASSWORD HISTORY x (Retenir les x derniers mdp afin que l'utilisateur ne réutilise pas le même), FAILED\_LOGIN\_ATTEMPTS x (x tentatives de connexion avant blocage du compte), PASSWORD\_LOCK\_TIME x (Nbre de jours de blocage du compte).
* ACCOUNT LOCK/UNLOCK
* Modifier mdp d'un utilisateur :

ALTER USER 'newuser'@'localhost' IDENTIFIED BY 'new\_password';

* Modification du mdp de "newuser" en "new\_password".
* Supprimer un utilisateur:

DROP USER 'user1'@'localhost';

* Renommer un utilisateur :

RENAME USER 'anciennom'@'localhost' TO 'nouveaunom'@'localhost';

* Attribution de privilèges :

GRANT ALL PRIVILEGES ON \*.\* TO 'user1'@'localhost';

* Attribution de tous les privilèges sur ttes les bases de données à "user1".

GRANT ALL PRIVILEGES ON dbexercices\_q2.\* TO 'user1'@'localhost';

* Attribution de tous les privilèges sur la base de données "db\_exercices\_q2" à "user1".

GRANT SELECT, INSERT, UPDATE, DELETE ON dbexercices\_q2.\* TO 'user1'@'localhost';

* Attribution de des privilèges spécifiques SELECT, INSERT, UPDATE et DELETE sur la base de données "db\_exercices\_q2" à "user1".

GRANT SELECT, INSERT ON dbexercices\_q2.client TO 'user1'@'localhost';

* Attribution de des privilèges spécifiques SELECT et INSERT sur la table "client" de la base de données "db\_exercices\_q2" à "user1".

GRANT SELECT, UPDATE ON dbexercices\_q2.client, dbexercices\_q2.employe TO 'user1'@'localhost';

* Attribution de des privilèges spécifiques SELECT et INSERT sur les tables "client" et "employe" de la base de données "db\_exercices\_q2" à "user1".

Remarques :

* Privilèges pouvant être attribués :
* Gestion des enregistrements : SELECT, INSERT, UPDATE, DELETE.
* Gestion de la structure des tables : CREATE, CREATE TEMPORARY TABLES, CREATE VIEW, ALTER, DROP.
* Autres éléments de la db : CREATE ROUTINE, ALTER ROUTINE, EXECUTE, INDEX, TRIGGER, LOCK TABLES, CREATE USER.
* Lors de la créat° d'une procédure stockée/trigger/vue, possibilité de définir l'user responsable de la définit° de celle-ci : CREATE DEFINER = 'user1'.'localhost' PROCEDURE/TRIGGER/VIEW… Pour que la procédure, le trigger ou la view vérifie les droits de celui qui invoque celle-ci plutôt que celui qui la définit : SQL SECURITY INVOKER.
* Retrait de privilèges :

REVOKE INSERT, UPDATE ON dbexercices\_q2.client FROM 'user1'@'localhost';

* Attribution de des privilèges spécifiques INSERT et UPDATE sur la table "client" de la base de données "db\_exercices\_q2" à "user1".

## Transactions

Regroupe +ieurs requêtes en un bloc. Si une des requêtes échoue ou si interrompue => Annulation de tte la transaction.

MySQL valide chaque requête automatiquement. Pour désactiver cela : SET autocommit = 0. Il faut que autocommit soit à 1 pour utiliser les transactions.

START TRANSACTION;

requête1;

requête2;

COMMIT; (Valider) / ROLLBACK; (Annuler)

Remarques:

* Possibilité de placer un point de sauvegarde :

SAVEPOINT nom\_pt\_de\_sauvegarde;

Pour annuler les requêtes jusqu'à ce point de sauvegarde :

ROLLBACK TO nom\_pt\_sauvegarde;

* Si une transaction est en cours et qu'elle modifie des lignes => un verrou est posé sur celles-ci tant qu'elle n'est pas terminée.
* Les transactions doivent respecter le principe ACID (Atomicité, cohérence, isolation et durabilité).

## Verrous

* Verrouiller des tables

LOCK TABLES nom\_table [READ/WRITE] [, nom\_table2];

READ => Lecture dans la table tjrs possible, mais pas d'écriture.

WRITE => Lecture et ecriture impossible pour les autres utilisateurs.

Remarques :

* LOCK libère ts les verrous en cours de l'utilisateur => Il faut tous les placer en une fois s'il y en a +ieurs.
* START TRANSACTION retire aussi ts les verrous de l'utilisateur.
* LOCK TABLES valide une transaction en cours d'écriture (Apres un START TRANSACTION).
* Si l'on veut utiliser LOCK TABLES ds une transaction => SET autocommit = 0;
* Déverrouiller les tables :

UNLOCK TABLES;

Remarques :

* On ne peut déverouiller les tables une à une.

## Procédures stockées

* Créat° d'une procédure stockée :

DELIMITER |

CREATE PROCEDURE nom\_procedure (IN/OUT/INOUT arg1 [,IN/OUT/INOUT arg2, …])

BEGIN

instr1;

instr2;

END |

DELIMITER ;

Remarques :

* Parenthèses obligatoires.
* Si une seule instruction => Placée directement après les parenthèses. Si +ieurs instruct°s => Placées entre le BEGIN et le END (Chacune suivi d'un ;).
* Raison de DELIMITER | : Serveur MySQL va considérer que la procédure se termine au premier ; rencontré => Nécessité de changer le délimiteur de le session en cours.
* 3 types d'arguments:
  + IN : arguments reçus par la procédure utilisés dans celle-ci;
  + OUT : Valeurs retournées par la procédure, utilisable hors de celle-ci;
* INOUT : arguments reçus par la procédure et qui peuvent être modifiés et retournés dans celle-ci.

1 argument = 1 sens (IN, OUT, INOUT) + 1 nom + 1 type. Le nom ne peut être un nom de colonne d'une des tables utilisées.

Pour utiliser OUT, attribuer la valeur dans une des requête (ne peut être utilisé qu’avec des requêtes ne renvoyant qu’une seule ligne). Utiliser INTO

CREATE PROCEDURE nom\_procedure (IN arg1, OUT arg2)

BEGIN

SELECT COUNT(\*) INTO arg2 FROM table WHERE col1 = arg1 ;

* Utiliser une procedure stockée:

CALL nom\_procedure();

Remarques :

* Valeurs passées soit en utilisant les variables utilisateur soit en utilisant directement la valeur.

SET @valeur := 1;

CALL nom\_procedure(@valeur);

* Variables locales :

DECLARE nomVariable typeVariable [DEFAULT valeur];

Pour changer sa valeur : soit SET… soit SELECT … INTO …

Remarques :

* Déclarat° doit se trouver en début du bloc (Après le begin).
* Utilisable uniquement dans le bloc (Jusqu'au END).
* Condit°s :

IF:

IF condition

THEN instructions

[ELSEIF condition2 THEN instructions]

[ELSE instructions]

END IF ;

Remarques :

* +ieurs ELSEIF possibles, mais pas après le ELSE.
* Structure plus compacte : IF(condit°, valeur\_si\_vrai, valeur\_si\_faux).

CASE:

CASE variable\_a\_evaluer

WHEN valeur1 THEN instructions

[WHEN valeur2 THEN instructions]

[ELSE instructions]

END;

Remarques :

* Peut s'utiliser dans la sélection d'une colonne (Utiliser AS pour lui donner un alias).

SELECT col1,

CASE

WHEN col2 = val1 THEN 'message1'

WHEN col2 = val2 THEN 'message2'

ELSE 'message3'

END AS nouveau\_nom

FROM table;

* Boucles :

WHILE condition

DO instructions

END WHILE;

Boucle exécutée tant que la condit° est vraie.

REPEAT instructions

UNTIL condition

END REPEAT;

Exécute la boucle jusqu’à ce que la condition soit vraie (Tjrs exécutée au moins une fois car la condit° est vérifiée à la fin de la boucle.)

nom\_bloc :

BEGIN

instr1;

instr2;

END nom\_bloc ;

Donne un nom à un bloc d'instruct°s. Pour sortir du bloc : LEAVE nom\_bloc; Pour passer à l'itération suivante : ITERATE nom\_bloc;

[nom\_boucle : ]

LOOP

instr1;

instr2;

END LOOP [nom\_boucle];

Ne s'utilise qu'avec LEAVE pour pouvoir en sortir.

## Triggers

Servent à contrôler l'intégrité des données ou leur archivage plutôt que leur suppression. Peuvent aussi permettre de garder un historique des actions.

* Création d'un trigger :

DELIMITER |

CREATE TRIGGER nom moment evenement ON table

FOR EACH ROW

BEGIN

instr1;

instr2;

END |

DELIMITER ;

Moment : Soit BEFORE, soit AFTER. Moment où le trigger est déclenché.

Evenement : Soit INSERT, soit UPDATE, soit DELETE.

Remarques :

* Un et un seul évenement par trigger.
* Pas de doublon moment/evenement.
* Nom du trigger = moment\_evenement\_table.
* Manipulat° de valeurs :

Pour récupérer les valeurs de l'évenement déclencheur : OLD (Ne peuvent qu'être lues)

Pour récupérer les valeurs après l'évenement déclencheur : NEW (Peuvent être lues et modifiées)

OLD.col1

NEW.col1

Remarques :

* OLD et NEW ne sont utilisés ensemble que lors d'un UPDATE. OLD est seulement utilisé lors d'un DELETE et NEW est seulement utilisé lors d'un INSERT.
* START TRANSACTION, COMMIIT, ROLLBACK interdits ds un trigger. Annulat° jusqu'à un point de sauvegarde possible (Cela ne met pas fin à la transact°).
* Valeur d'une colonne NEW modifiable avec BEFORE mais pas AFTER.
* Si erreur ds un trigger BEFORE => évenement déclencheur pas exécuté => éventuel trigger AFTER pas exécuté.
* Suppression d'un trigger :

DROP TRIGGER nom\_trigger;

## Vues

* Créat° d'une vue :

CREATE VIEW nom\_vue AS requete;

Remarques :

* CREATE peut être completé par ON REPLACE =>Si une vue avec le même nom existe, elle est remplacée par la nouvelle.
* Afficher le contenu d'une vue :

SELECT \*

FROM nom\_vue;

* Limites :
* Les résultats de la requête d’une vue ne sont pas stockés. Ils sont recalculés à chaque fois qu’on accède à la vue, ce qui peut affecter les performances si la requête est complexe.
* Si la structure d’une table sous-jacente est modifiée après la création d’une vue, la vue ne prend pas en compte ces modifications automatiquement. Par exemple, l’ajout d’une colonne à une table ne se reflète pas dans la vue sans recréer cette dernière.
* La requête d’une vue ne peut pas contenir de référence à une variable locale, utilisateur, ou système.
* Lors de la création d’une vue avec des jointures, si des colonnes ont le même nom, des alias doivent être utilisés, car deux colonnes ne peuvent pas avoir le même nom dans une vue.
* Si une vue contient un tri (ORDER BY), ce tri n’est respecté que si la requête utilisant la vue ne spécifie pas un autre tri. Ainsi, le tri dans la vue peut être ignoré.
* Il est déconseillé d’utiliser une clause LIMIT dans une vue car si la requête qui utilise la vue spécifie également un LIMIT, le comportement de MySQL peut devenir indéterminé.
* L’algorithme MERGE ne peut pas être utilisé avec les clauses DISTINCT, LIMIT, les fonctions d’agrégation, GROUP BY, HAVIN, UNION ou une sous-requête dans le SELECT . De plus, les modifications (UPDATE, INSERT) sur une vue nécessitent l’algorithme MERGE.
* Les modifications via une vue avec jointures ne peuvent porter que sur une seule table et uniquement si l’algorithme MERGE est utilisé.
* Les règles pour les insertions sont strictes : toutes les colonnes sans valeur par défaut et non NULL doivent être présentes dans la vue et l’insertion, les jointures doivent être internes et sur une seule table, et aucune colonne ne peut contenir d’expression.
* Lors de modifications ou insertions via une vue avec l’option WITH CHECK OPTION, les conditions de la requête doivent être respectées.
* La suppression à partir d’une vue n’est possible que si les modifications (UPDATE) sont possibles et que la vue ne concerne qu’une seule table.

## Tables temporaires

Une table temporaire n'existe que dans le session dans laquelle elle a été créée.

* Création :

CREATE TEMPORARY TABLE nom\_table

…

Remarques :

* Utilisation (INSERT, UPDATE, DELETE, ALTER, DROP …) identique aux tables permanentes.
* Conseillé de mettre TEMPORARY après ALTER ou DROP, afin d'éviter de modifier ou supprimer une table permanente portant le même nom par erreur.
* Si une table temporaire a le même nom qu'une table permanente dans une session, la table temporaire cache la permanente.
* Pas de contrainte de FK et pas plus d'une référence à une table temporaire dans une requête.
* CREATE, ALTER ou DROP d'une table temporaire => Pas de validat° de transaction. Annulat° avec ROLLBACK impossible.
* Utiles pour gagner de la performance si une même requête est exécutée +ieurs fois, car pas de recalcul nécessaire.
* Utiles pour faire des test avant de manipuler une table permanente.
* Remplir une table temporaire sur base de la structure et du contenu d'une autre table :

1: Créer une table temporaire suivant la structure de la table modèle :

CREATE TEMPORARY TABLE table\_temp LIKE table\_modele;

2: Remplir la table temporaire du contenu de la table modèle :

INSERT INTO table\_temp SELECT \* table\_modele;