

Initiation au langage SQL procédural (grâce au SGBD MariaDB)



Objectifs

Etre capable d'améliorer une base de données relationnelle à l'aide des fonctionnalités avancées du langage SQL.

Rappel

Le manuel officiel de MariaDB est en ligne et accessible à cette adresse :

<https://mariadb.com/kb/en/documentation/>

SQL procédural – MariaDB SQL / PSM (Persistent Stored Modules)

Les structures de contrôle habituelles d'un langage de programmation (IF, WHILE, CASE, ...) ne font pas partie de la norme SQL au départ mais sont apparues dans une sous-partie optionnelle de la norme (ISO/IEC 9075-5:1996, flow-control statements).

MariaDB les prend en compte, cf documentation officielle MariaDb sur les structures de contrôle (« flow-control statements ») :

<https://mariadb.com/kb/en/programmatic-compound-statements/>

1) Variables

1.1) Variables de session

En SQL, les variables définies par l'utilisateur ont un nom **commençant par @** et il existe plusieurs possibilités pour affecter une valeur à une variable :

- l'affectation ⇒ **SET** @ma_variable := ...
- la directive **INTO** d'une requête ⇒ **SELECT** ... **INTO** @variable **FROM**...

Exemple :

SET @prix :=0;

ACTIVITE 1 : en utilisant la petite base de test fournie sur Moodle,

- récupérez, dans une variable, l'identifiant auto-incrémenté de la ligne du profil (/ compte) nouvellement insérée dans la base pour pouvoir le réutiliser pour insérer le compte d'un administrateur / formateur.
- Affectez à une variable @mdp_sel le résultat de la concaténation du mot de passe choisi avec une chaîne de caractères appelé « sel » puis dans la variable @mdp_sel_hash récupérez l'empreinte sha256 et modifiez le mot de passe du compte « vmarc » en lui appliquant l'empreinte calculée.

1.2) Variables déclarées

Ces variables sont déclarées (via **DECLARE**) en local, dans les sous-programmes (fonctions ou procédures). Il faut utiliser la directive **DEFAULT** pour prévoir une valeur par défaut à ce genre de variable. Si aucune valeur par défaut n'est spécifiée, la valeur initiale est alors NULL.

2) Structures de contrôle

2.1) La commande IF

```
IF search_condition THEN statement_list
  [ELSEIF search_condition THEN statement_list] ...
  [ELSE statement_list]
END IF;
```

2.2) La commande CASE

```
CASE case_value
  WHEN when_value THEN statement_list
  [WHEN when_value THEN statement_list] ...
  [ELSE statement_list]
END CASE;
```

2.3) La boucle WHILE

```
WHILE search_condition DO
  statement_list
END WHILE;
```

2.3) La boucle FOR

```
FOR var_name IN [ REVERSE ] lower_bound .. upper_bound
DO statement_list
END FOR;
```

Les vues

Documentation officielle MariaDb sur les vues (« views ») :
<https://mariadb.com/kb/en/views/>

Une **vue (view)** est une présentation particulière (d'une partie) des données de la base de données sous forme d'une table virtuelle qui n'a pas d'existence propre.

Une vue est une sorte de fenêtre qui autorise aux utilisateurs la vision d'une partie de la base. Les données d'une vue sont extraites **à la demande** par une requête de définition et sont rechargées à chaque nouvelle interrogation.

```
CREATE
  [OR REPLACE]
  [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
  [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
  [SQL SECURITY { DEFINER | INVOKER }]
  VIEW [IF NOT EXISTS] view_name [(column_list)]
  AS select_statement
  [WITH [CASCADED | LOCAL] CHECK OPTION]
```

Utilisations d'une vue :

- pour la confidentialité de certaines informations,
- pour réaliser des contrôles de contraintes d'intégrité,
- pour simplifier la formulation de requêtes complexes.

Exemples :

1) Création d'une liste des salariés sans mention du salaire ni des informations personnelles :

```
CREATE VIEW LISTE_PERSONNEL
AS SELECT  NOM, PRENOM, POSTE
FROM    SALARIE;
```

--puis visualisation du contenu de la vue comme si c'était une nouvelle table
SELECT * FROM LISTE_PERSONNEL ;

2) Constitution d'une commande complète :

```
CREATE VIEW COMMANDE_COMPLETE(NCOM, NCLI, NOMCLI, LOC, DATECOM)
AS SELECT  NCOM, COM.NCLI, NOM, LOCALITE, DATECOM
FROM    CLIENT CLI, COMMANDE COM
WHERE   COM.NCLI = CLI.NCLI;
```

3) Restriction en lecture d'une table CLIENT les samedis et dimanches

```
CREATE VIEW CLIENT_JRS_FERIES
AS SELECT * FROM CLIENT
WHERE   DATE_FORMAT(SYSDATE(), %W)
IN(SUNDAY, SATURDAY);
```

ACTIVITE 2 : en utilisant la petite base de test fournie sur Moodle,

- créez une vue contenant uniquement les noms et prénoms du contenu de la table de gestion des profils.
- Visualisez alors le contenu de cette nouvelle vue.

Les fonctions stockées

Documentation officielle MariaDb sur les fonctions (« stored functions ») :

<https://mariadb.com/kb/en/stored-functions/>

Une fonction SQL est un sous-programme SQL codé en SQL procédural ou en langage externe qui accepte des paramètres en entrée et retourne une seule valeur.

Le langage SQL (depuis SQL:1999) permet des programmes côté serveur qui sont gérés comme des objets de la base de données à part entière.

```
CREATE [OR REPLACE]
  [DEFINER = {user | CURRENT_USER | role | CURRENT_ROLE }]
  [AGGREGATE] FUNCTION [IF NOT EXISTS] func_name ([func_parameter[,...]])
  RETURNS type
  [characteristic ...]
  RETURN func_body
func_parameter:
  param_name type
type:
  Any valid MariaDB data type
characteristic:
  LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'

BEGIN
  [DECLARE ...;]
  <bloc_instructions>
END ;
```

Exemples :

1) Fonction d'affichage d'un message :

```
DELIMITER //
CREATE FUNCTION hello_world(choix INT) RETURNS TEXT
BEGIN
  IF choix=1 THEN
    RETURN 'Hello World !';
  ELSE
    RETURN 'Bonjour tout le monde !';
  END IF;
END;
//
DELIMITER ;
```

Systemes d'information

--puis ensuite pour appeler la fonction :
SELECT hello_world(1);

2) Fonction de nettoyage d'une table :

```
DELIMITER //
CREATE FUNCTION nettoie() RETURNS INT
BEGIN
    DELETE FROM `t_catalogue_cat`
        WHERE `cat_old` LIKE 'OLD';
    RETURN 0;
END;
//
DELIMITER ;
```

--puis ensuite pour appeler la fonction autant de fois que nécessaire :
SELECT nettoie();

3) Suppression de la fonction nettoie() si elle existe dans le base :

```
DROP FUNCTION IF EXISTS nettoie;
```

ACTIVITE 3 : en utilisant la petite base de test fournie sur Moodle,

- créez une fonction retournant l'âge du titulaire d'un profil d'après sa date de naissance passée en paramètre (ajoutez, en utilisant la commande ALTER TABLE, une colonne à votre table de gestion des profils si cela est nécessaire).
- Appelez la fonction créée.

Les procédures stockées

Documentation officielle MariaDb sur les procédures (« stored procedures ») :
<https://mariadb.com/kb/en/stored-procedures/>

Une procédure SQL est un sous-programme SQL codé en SQL procédural ou en langage externe qui est invoqué par CALL, qui réalise des traitements et peut avoir des paramètres en entrée, en sortie et en entrée/sortie.

Dans une procédure, il est possible d'appeler des fonctions que l'on a créées.

```
CREATE
    [OR REPLACE]
    [DEFINER = { user | CURRENT_USER | role | CURRENT_ROLE }]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MariaDB data type

characteristic:
    LANGUAGE SQL
    | [NOT] DETERMINISTIC
```

```
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }  
| SQL SECURITY { DEFINER | INVOKER }  
| COMMENT 'string'
```

```
BEGIN  
  [DECLARE ...;]  
  <bloc_instructions>  
END ;
```

Exemples :

1) Procédure qui exécute une requête de recherche de tous les éléments d'une table :

```
DELIMITER //  
CREATE PROCEDURE affichage_catalogue()  
BEGIN  
  SELECT * FROM `t_catalogue_cat`;  
END;  
//  
DELIMITER ;  
  
--puis ensuite pour appeler la procédure autant de fois que nécessaire :  
CALL affichage_catalogue();
```

2) Procédure qui appelle la fonction hello_world() créée auparavant :

```
DELIMITER //  
CREATE PROCEDURE affichage_message(IN VALEUR INT)  
BEGIN  
  SELECT hello_world(VALEUR);  
END;  
//  
DELIMITER ;  
  
--puis ensuite pour appeler la procédure autant de fois que nécessaire :  
SET @val=1;  
CALL affichage_message(@val);
```

3) Suppression d'une procédure :

```
DROP PROCEDURE IF EXISTS affichage_message;
```

ACTIVITE 4 : en utilisant la petite base de test fournie sur Moodle,

- créez une procédure renvoyant l'âge du titulaire d'un profil en sortie à partir de son identifiant en entrée.
- Utilisez la procédure créée pour connaître l'âge du titulaire d'un profil d'identifiant **1** dans la base de données.
- Puis écrivez une autre procédure utilisant la fonction créée dans l'activité 3 dans le but d'afficher le message « majeur » si le titulaire d'un profil a plus de 18 ans ou « mineur » si le titulaire du profil a moins de 18 ans.
- Créez une nouvelle vue contenant les noms, prénoms et âges des titulaires des profils du contenu de la table de gestion des profils
- et créez une procédure qui renvoie en sortie l'âge moyen des profils dans la base.

Les déclencheurs (« triggers »)

Documentation officielle MariaDb sur les déclencheurs (« triggers ») :
<https://mariadb.com/kb/en/triggers/>

Un déclencheur (ou « trigger ») est un objet de la base **relatif à une table** contenant du code SQL exécuté automatiquement en fonction d'un ordre de mise à jour SQL associé.

On peut considérer que les triggers sont des sous-programmes résidents associés à un événement particulier (insertion, modification, suppression) sur **une table** (ou une vue).

```
CREATE TRIGGER <nom_trigger>
{BEFORE | AFTER }
{INSERT | UPDATE | OF <liste_colonne> | DELETE }
ON <nom_table> [REFERENCING <liste_alias_valeurs_anciennes_nouvelles>]
[FOR EACH {ROW | STATEMENT}]
[WHEN (<predicat>)]
BEGIN
  <code_sql>
END
<liste_alias_valeurs_anciennes_nouvelles>::=
OLD [ROW][AS] <nom_correlation_valeurs_anciennes>
NEW [ROW][AS] <nom_correlation_valeurs_nouvelles>
OLD TABLE[AS] <alias_table_valeurs_anciennes>
NEW TABLE[AS] <alias_table_valeurs_nouvelles>
```

ATTENTION :

Un trigger peut modifier et/ou insérer des données dans n'importe quelle table mais en ce qui concerne la table à laquelle le trigger est attaché (qui est forcément utilisée par l'événement déclencheur), le trigger peut lire et modifier uniquement la ligne insérée, modifiée ou supprimée qu'il est en train de traiter.

Les mots-clés **OLD** et **NEW** permettent d'accéder aux valeurs des colonnes concernées avant ou après l'événement déclencheur.

Par exemple, si vous **mettez à jour (UPDATE)** la valeur de colonne pfl_date_naissance d'une ligne de la table de gestion des profils, vous pouvez avoir accès à l'ancienne et à la nouvelle valeur contenue dans la colonne en utilisant la syntaxe suivante :

- **OLD.**pfl_date_naissance
- **NEW.** pfl_date_naissance.

Avant la modification de la ligne :

2	MARC	Valérie	vmarc@univ-brest.fr	A	A	2020-10-11	1974-05-25
---	------	---------	---------------------	---	---	------------	------------

Requête SQL-DML de modification de la ligne :

UPDATE t_profil_pfl SET pfl_date_naissance='1976-05-24' WHERE pfl_id=2;

Au moment de l'exécution de cette requête, **l'ancienne date de naissance de la ligne concernée** est stockée dans **OLD.pfl_date_naissance** et vaut **'1974-05-25'**.

Et la nouvelle date de naissance de la ligne concernée est stockée dans **NEW.pfl_date_naissance** et vaut '1976-05-24'.

Après la modification de la ligne :

2	MARC	Valérie	vmarc@univ-brest.fr	A	A	2020-10-11	1976-05-24
---	------	---------	---------------------	---	---	------------	------------

Exemples :

1) Trigger qui se déclenche lors de l'insertion d'une ligne dans une table :

```
DELIMITER //
CREATE TRIGGER maj_catalogue
AFTER INSERT ON t_miseAJour_maj
FOR EACH ROW
BEGIN
UPDATE t_catalogue_cat SET cat_timestamp = CURDATE( );
END;
//
DELIMITER ;
```

2) Suppression d'un trigger :

```
DROP TRIGGER maj_catalogue;
```

ACTIVITE 5 : en utilisant la petite base de test fournie sur Moodle,

- créez un « trigger » qui applique à la date de création du profil la date du jour d'insertion des données du nouveau profil.
- Créez un autre « trigger » qui met à jour la date associée au profil suite à la mise à jour d'une ligne de la table de gestion des comptes (ex : mise à jour du mot de passe).
- Activez les 2 déclencheurs créés.

A prévoir pour la base de données du S.I « Contestify »

A prévoir et à ajouter à votre fichier individuel de requêtes SQL :

ACTIVITE 6 :

- Ecrire **une fonction** qui retourne l'identifiant du dernier concours (/édition) ajouté dans la table de gestion des concours.
- Testez cette fonction.
- Ecrire alors **une procédure** qui insère une actualité à la date d'aujourd'hui à partir du dernier concours (/édition) créé dans la table de gestion des concours en indiquant comme texte de l'actualité le nom du concours, sa date de début et le petit texte introductif. L'auteur de l'actualité sera l'organisateur responsable du concours.

Systemes d'information

- Testez cette procédure.
- Puis, en réutilisant ce qui a été fait précédemment, créez **un déclencheur (trigger)** ajoutant une actualité dès la création d'un nouveau concours (/ édition).
- Activez ce trigger.

ACTIVITE 7 :

- Ecrivez une fonction qui retourne le nom de la phase actuelle d'un concours (/ édition) dont on passe l'identifiant en paramètre (ex : « à venir », « inscriptions », « sélection », « finale », « terminé »).
- Testez cette fonction.