Les fonction personnalisées

Vues et procédures stockées

Sommaire

- Fonction scalaire
- Fonction système
- Procédure stockée
- Consultation de la structure des bases
- Le langage de contrôle des fluxLes vues

Introduction

Au delà des fonctions natives existantes dans le langage SQL, un SGBD offre la possibilité de travailler avec des fonctions « utilisateurs » créées pour un besoin spécifique.

Cela fonctionne comme pour un langage de programmation : entrée, traitement, sortie.

Deux type de fonction :

- Les fonctions scalaires.
- Les fonctions systèmes.
- Les procédures stockées

Potentiel besoin de mettre à jour une variable global dans MySQL :
SET GLOBAL log_bin_trust_function_creators = 1;

```
SET GLOBAL log_bin_trust_function_creators = 1;
```

Fonction scalaire

Ce type e fonction permet de réaliser un traitement sur des données de la base.

- Il retourne une seule valeur.
- Le type de la valeur retourné est définie lors de la création de la fonction.

```
DELIMITER ||
create function ANCIENNETE (D DATE)
RETURNS int
BEGIN
DECLARE A int default 1;
SET A = round(datediff(now(), D)/365, 0);
RETURN A;
END ||
DELIMITER ;

select first_name, ANCIENNETE(create_date) as anciennete from customer;
```

first_name	anciennete
MARY	14
PATRICIA	14
LINDA	14
BARBARA	14
ELIZABETH	14
JENNIFER	14

Fonction scalaire

- **DELIMITER** : Ajouter avant la création de la fonction cette commande permet de définir que pour cette fonction le caractère exprimant la fin d'instruction est || (Indispensable pour que la fonction ne s'arrête pas au premier « ; »).
- **DECLARE** : Permet de déclarer une variable dont on définit le type (int dans l'exemple).
- **SET** : Permet d'affecter une variable (la mettre à jour).
- Pour manipuler une fonction : **DROP** pour la supprimer et **ALTER** pour la modifier.

Fonction système

Les fonctions système non modifiables sont fournie par le moteur de base de données. La liste varie selon le SGBD, mais on retrouve couramment les fonctions suivante :

- **SYSDATE :** retourne la date du serveur
- **NOW() :** retourne la date et l'heure.
- **DAY(), MONTH(), YEAR() :** retourne la partie citée d'une date.
- **DATEDIFF()** : retourne la différence entre 2 dates.

Procédure stockée

Une procédure stockée est une série d'instruction SQL regroupée sous un même nom.

- Une foi créer dans la base de données elle est utilisable au même titre qu'une table ou une fonction.
- L'appelle d'une procédure stockée déclenche l'ensemble des instruction SQL qui la compose

L'usage de ces procédures est donc pertinent dans le cas de traitement récurrents.

Procédure stockée

Exemple :

```
DELIMITER ||
CREATE PROCEDURE ANCIENNETE(IN NOM VARCHAR(55), OUT ANCIE_CUSTOMER INT)
BEGIN
    SELECT round(datediff(now(), create_date)/365, 0) AS ANCIENNETE
    INTO ANCIE_CUSTOMER
    FROM CUSTOMER
    WHERE NOM = first_name;
END||
DELIMITER ;

CALL ANCIENNETE('MARY', @ANCIE_CUSTOMER);
SELECT @ANCIE_CUSTOMER;
```

	@ANCIE_CUSTOMER
▶	14

Chacun des paramètres d'entrée est défini selon trois éléments :

- Le sens : entrant (**IN**), sortant(**OUT**), ou les deux (**INOUT**)
- Le nom : permet de l'utiliser dans la procédure
- Le type : INT, DATE, CHAR()

Procédure stockée

- Comme pour les fonctions une procédures stockée est également capable d'utiliser des variables créé avec la commande **DECLARE** et initialisé avec la commande **SET**.
- La commande **INTO** permet d'affecter le résultat de la requête à une variable et le rendre disponible pour une utilisation ultérieur.
- La suppression se fait via un **DROP**.
- Le **CALL** permet de déclencher l'appel de la procédure ~~avant de pouvoir l'utiliser via un **SELECT**~~.

Consultation de la structure des bases

La consultation de la structure d'une fonction ou d'une procédure stockée se fait par l'interrogation du catalogue système du SGBD.

Il est lui-même une base de données composées de **VUES**. Contenant toutes les informations sur toutes les bases existantes. Cette base se nome : **INFORMATION_SCHEMA**.

```
select routine_definition
from information_schema.routines
where specific_name = 'ANCIENNETE'
and routine_type = 'FUNCTION';
```

ROUTINE_DEFINITION
BEGIN DECLARE A int default 1; SET A = round(datediff(now(), D)/365, 0); RETURN A; END

Consultation de la structure des bases

La base INFORMATION_SCHEMA est très riche et permet, à tout moment, de connaitre :

- La structure d'une base : **SCHEMATA**
- La structure d'une base : **TABLES**
- Le détail d'une colonne (champ) : **COLUMNS**
- Les informations sur les index : **STATISTICS**
- Les informations sur les contraintes : **TABLE_CONSTRAINTS**
- Les informations sur les fonctions et procédures : **ROUTINES**
- Les informations sur les triggers : **TRIGGERS**
- Les informations sur les utilisateurs : **USER_PRIVILEGES**

Le langage de contrôle de flux

Il est possible d'utiliser le langage de contrôle de flux à l'intérieur de ces procédures pour conditionner certains traitements.

```
DELIMITER ||
CREATE PROCEDURE ANCIENNETE(IN NOM VARCHAR(55), OUT ANCIE_CUSTOMER INT)
BEGIN
    IF length(NOM) = 1 THEN
        SELECT round(datediff(now(), create_date)/365, 0) AS ANCIENNETE
        INTO ANCIE_CUSTOMER
        FROM CUSTOMER
        WHERE NOM = LEFT(first_name, 1) LIMIT 1;
    ELSEIF length(NOM) > 1 THEN
        SELECT round(datediff(now(), create_date)/365, 0) AS ANCIENNETE
        INTO ANCIE_CUSTOMER
        FROM CUSTOMER
        WHERE NOM = first_name;
    ELSE
        SELECT 0 as error
        INTO ANCIE_CUSTOMER;
    END IF;
END||
DELIMITER ;

CALL ANCIENNETE('M', @ANCIE_CUSTOMER);
SELECT @ANCIE_CUSTOMER;
```

La commande IF

Le langage de contrôle de flux

```
DROP PROCEDURE ANCIENNETE;

DELIMITER ||
CREATE PROCEDURE ANCIENNETE(IN NOM VARCHAR(55), OUT ANCIE_CUSTOMER VARCHAR(55))
BEGIN
    SELECT
        CASE
            WHEN round(datediff(now(), create_date)/365, 0) > 2 THEN 'Ancien'
            WHEN round(datediff(now(), create_date)/365, 0) <= 2 THEN 'Réscent'
        END
    INTO ANCIE_CUSTOMER
    FROM CUSTOMER
    WHERE NOM = first_name;
END||
DELIMITER ;

CALL ANCIENNETE('MARY', @ANCIE_CUSTOMER);
SELECT @ANCIE_CUSTOMER;
```

La commande CASE

Le langage de contrôle de flux

- Boucle **WHILE**
- Boucle **REPEAT** : comme la boucle WHILE, mais la condition de sortie est évalué en fin de boucle.
- Boucle **LOOP** : Pas de critère de sortie. La sortie se fait par la commande **LEAVE** couplé avec un **IF**.
- **ITERATE** : permet de relancer la boucle dans la quel on se trouve en ignorant les instructions qui la suivent.
- **DEFAULT** : Pour affecter une valeur par défaut à une variable.
- …

Voir Transact-SQL.

Les vues

Un autre types d’objet qui présente plusieurs avantage pour l’utilisateur est la **VUE**.

Une vue dans une base de données est une synthèse d'une requête d'interrogation de la base. On peut la voir comme une table virtuelle, définie par une requête.

Une vue ne stocke pas le résultat de la requête, mais exécute sont schéma à chaque sollicitation. Elle ne rend donc pas l’exécution d’une requête plus performante.

Cela se fait avec la commande :

CREATE VIEW <nom> **AS**

SELECT …

Les vues

Les avantages des vues sont :

- Eviter de taper une requête très longue : la vue sert à donner un nom à la requête pour l'utiliser souvent.
- Masquer certaines données à certains utilisateurs. En SQL, les protections d'une vue ne sont pas forcément les mêmes que celles des tables sous-jacentes.
- Fournir une « table » résultats de traitement préalable.
- Augmenter la protection des données en masquant certaine données à l’utilisateur.