

MySQL perfectionnement

Les procédures stockées

Imane ait yahiatene

Definition d'une procédure stockée

Une procédure stockée est une série d'instructions SQL désignée par un nom. Lorsque l'on crée une procédure stockée, on l'enregistre dans la base de données que l'on utilise, au même titre qu'une table, par exemple.





Définition

Une fois la procédure créée, il est possible d'appeler celle-ci par son nom. Les instructions de la procédure sont alors exécutées. Elles sont comme leur nom l'indique, stockées de manière durable, et font bien partie intégrante de **la base de données** dans laquelle elles sont enregistrées.

Pourquoi les procédures stockées?

- Les procédures stockées sont rapides. Le serveur MySQL tire parti de la mise en cache. Le principal gain de vitesse provient de la réduction du trafic réseau. Si vous avez une tâche répétitive qui nécessite des vérifications, des boucles, de multiples instructions et aucune interaction avec l'utilisateur, faites-la avec un appel unique à une procédure stockée sur le serveur.

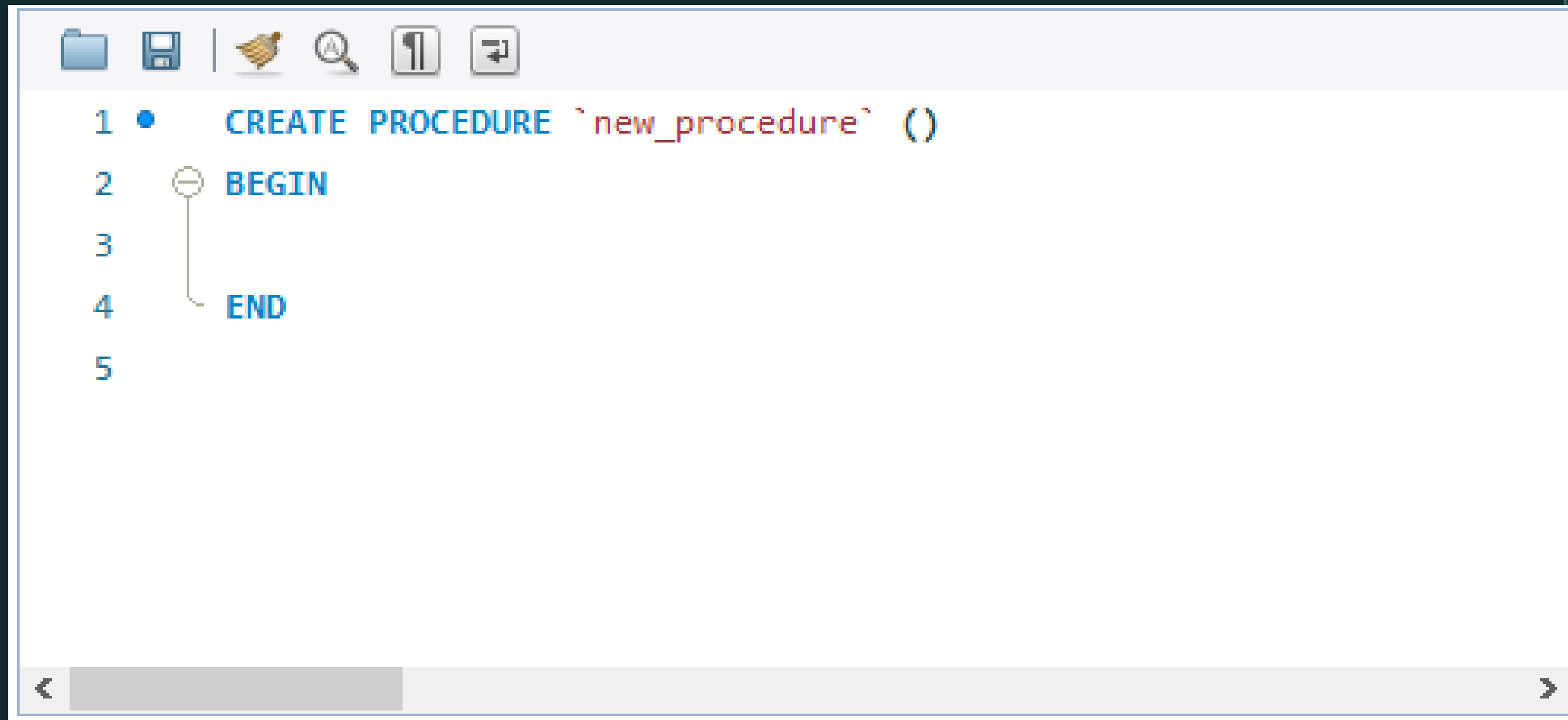
- Les procédures stockées sont portables. Lorsque vous écrivez votre procédure stockée en SQL, vous savez qu'elle fonctionnera sur toutes les plates-formes sur lesquelles MySQL fonctionne, sans vous obliger à installer un paquetage d'environnement d'exécution supplémentaire, à définir les autorisations d'exécution du programme dans le système d'exploitation ou à déployer différents paquetages si vous avez différents types d'ordinateurs.
- Les procédures stockées sont toujours disponibles en tant que "code source" dans la base de données elle-même. Et il est logique de lier les données aux processus qui opèrent sur ces données.

Création d'une procédure stockée

Par défaut, une procédure est associée à la base de données par défaut (base de données actuellement utilisée). Pour associer la procédure à une base de données donnée, spécifiez le nom comme `nom_de_base.nom_de_procedure_stockée` lorsque vous la créez.



Syntaxe



The screenshot shows a SQL IDE window with a toolbar at the top containing icons for file operations, execution, and navigation. The main text area displays the following SQL code:

```
1 • CREATE PROCEDURE `new_procedure` ()  
2   BEGIN  
3  
4   END  
5
```

Line numbers 1 through 5 are visible on the left. A blue dot is on line 1. A vertical line with a circle at the top connects the 'BEGIN' statement on line 2 to the 'END' statement on line 4, indicating a block structure. A scrollbar is visible at the bottom of the text area.

Appel d'une procédure

une fois la procédure créer et ajoutée à la base de données on pourras l'appeler chaque fois du'on en a besoin, pas la peine de la réécrire, la syntaxe d'appel est la suivante:

CALL new_procedure([parameter[,...]])

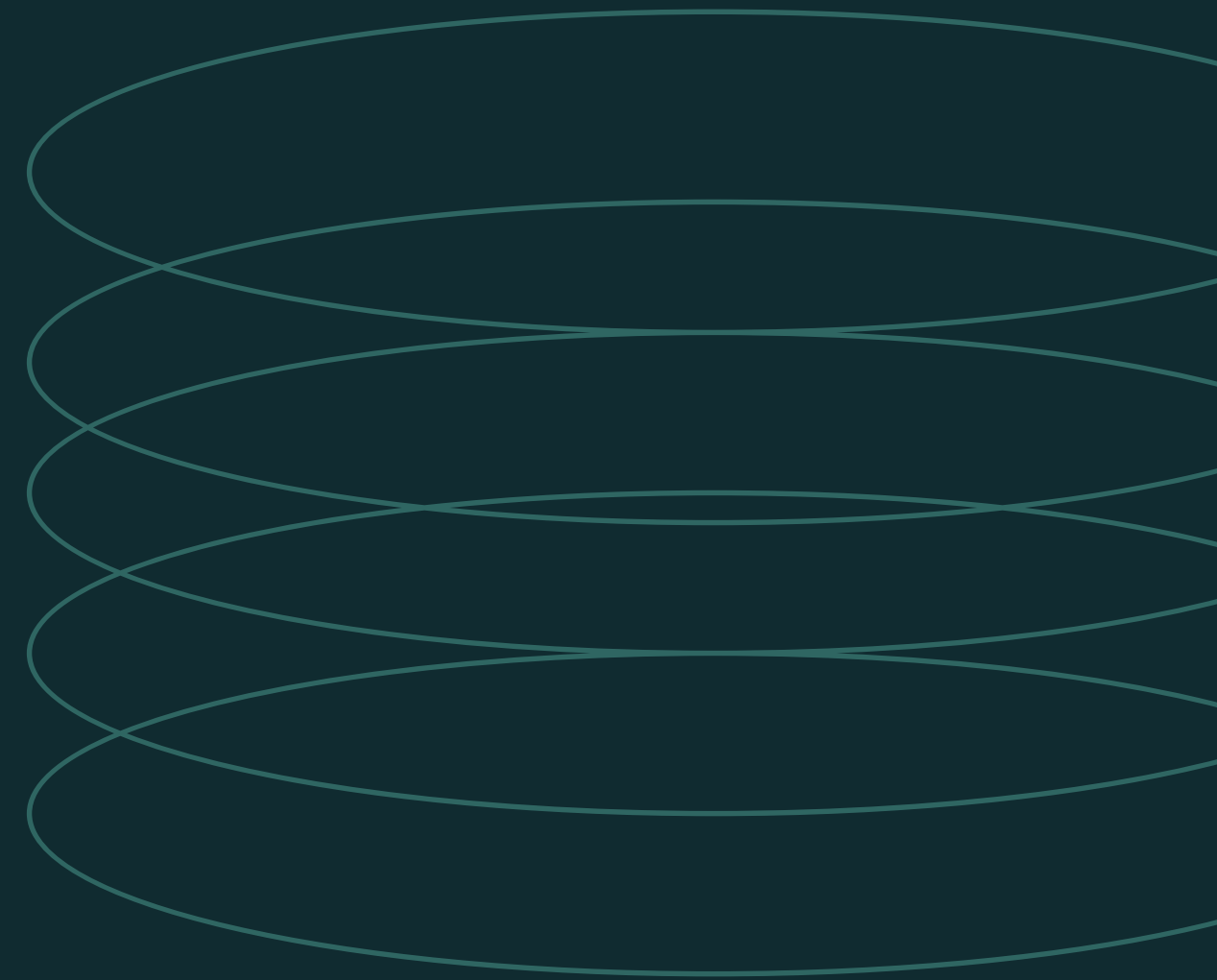
Afficher les procédures existantes

a

```
SHOW CREATE PROCEDURE new_procedure;
```


Exemple à tester

```
CREATE PROCEDURE afficher_titres_bloc()  
  -- pas de paramètres dans les parenthèses  
BEGIN  
  SELECT title FROM film;  
END;
```



Essayons maintenant avec:

DELIMITER | -- On change le délimiteur

CREATE PROCEDURE afficher_races()

-- toujours pas de paramètres, toujours des parenthèses

BEGIN

SELECT id, nom, espece_id, prix

FROM Race; -- Cette fois, le ; ne nous embêtera pas

END| -- Et on termine bien sûr la commande CREATE PROCEDURE

par notre nouveau délimiteur

DELIMITER;

Le délimiteur

Le délimiteur est le caractère ou la chaîne de caractères qui est utilisé pour compléter une instruction SQL. Par défaut, nous utilisons le point-virgule (;) comme délimiteur. Mais cela pose un problème dans les procédures stockées car une procédure peut contenir de nombreuses instructions, et toutes doivent se terminer par un point-virgule. Donc, pour votre délimiteur, choisissez une chaîne qui apparaît rarement dans une déclaration ou dans une procédure. Ici, nous avons utilisé le pipe, c'est-à-dire | . Vous pouvez utiliser ce que vous voulez.

Les labels

Il est possible de donner un label (un nom) à une boucle, ou à un bloc d'instructions défini par BEGIN... END. Il suffit pour cela de faire précéder l'ouverture de la boucle/du bloc par ce label, suivi de ':'

```
1 • USE `entreprise`;  
2 • DROP procedure IF EXISTS `new_procedure`;  
3  
4 DELIMITER $$  
5 • USE `entreprise`$$  
6 • CREATE PROCEDURE `new_procedure` ()  
7   bloc:BEGIN  
8     select* from employee;  
9  
10  END bloc;$$  
11  
12 DELIMITER ;  
13
```

Application

Créer une procédure stockée qui retourne à chaque fois combien d'employés ont un salaire supérieur à 2000.00 tout en ajoutant un label au bloc BEGIN-END

DECLARE

L'instruction DECLARE est utilisée pour définir divers éléments locaux à un programme, par exemple des variables locales, des conditions et des gestionnaires, des curseurs. DECLARE n'est utilisée qu'à l'intérieur d'une instruction composée BEGIN END et doit être placée au début de celle-ci, avant toute autre instruction. Les déclarations suivent l'ordre suivant :

- Les déclarations de curseurs doivent apparaître avant les déclarations de gestionnaires.
- Les déclarations de variables et de conditions doivent apparaître avant les déclarations de curseurs ou de gestionnaires.



Les variables dans les programmes stockés

SET nom_variable = valeur | opération;

Les variables système et les variables définies par l'utilisateur peuvent être utilisées dans les procédures stockées, tout comme elles peuvent être utilisées en dehors de ce contexte. Les programmes stockés utilisent DECLARE pour définir les variables locales, et les routines stockées (procédures et fonctions) peuvent être déclarées pour prendre des paramètres qui communiquent des valeurs entre la routine et son appelant.

Declarer une variable

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

Pour fournir une valeur par défaut à une variable, incluez une clause DEFAULT. La valeur peut être spécifiée sous forme d'expression ; elle ne doit pas nécessairement être constante. Si la clause DEFAULT est absente, la valeur initiale est NULL.

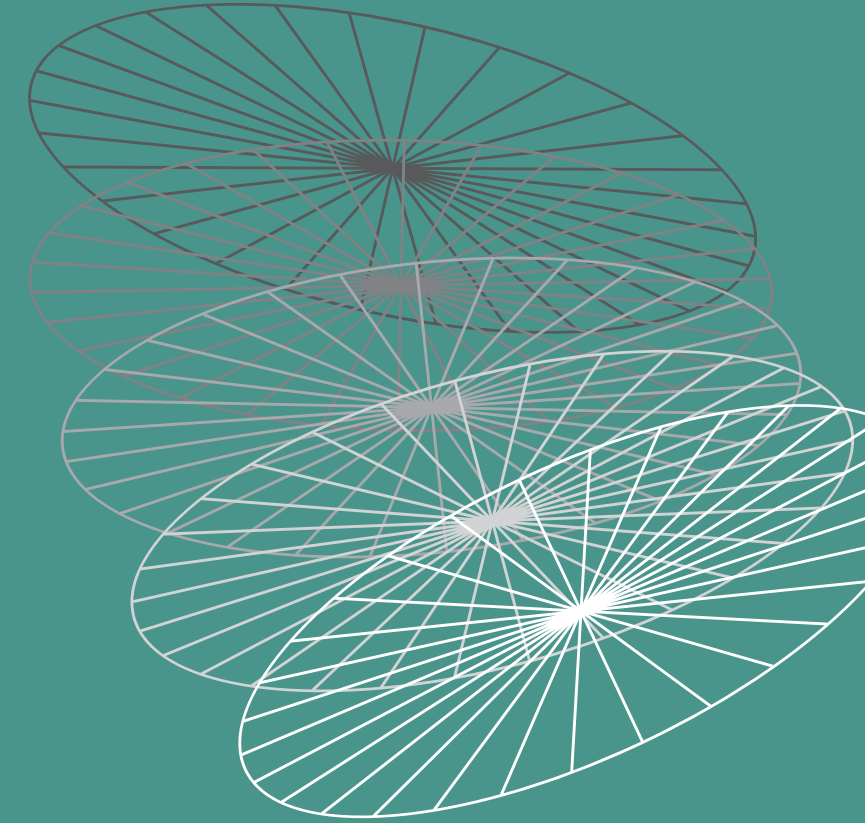
Exemple avec variables locales

```
DELIMITER $$
USE `entreprise`$$
CREATE PROCEDURE `new_procedure` ()
bloc:BEGIN
    DECLARE v_date DATE DEFAULT CURRENT_DATE();
    -- On déclare une variable locale et on lui met une valeur par défaut

    SELECT DATE_FORMAT(v_date, '%W %e %M %Y') AS Aujourd'hui;

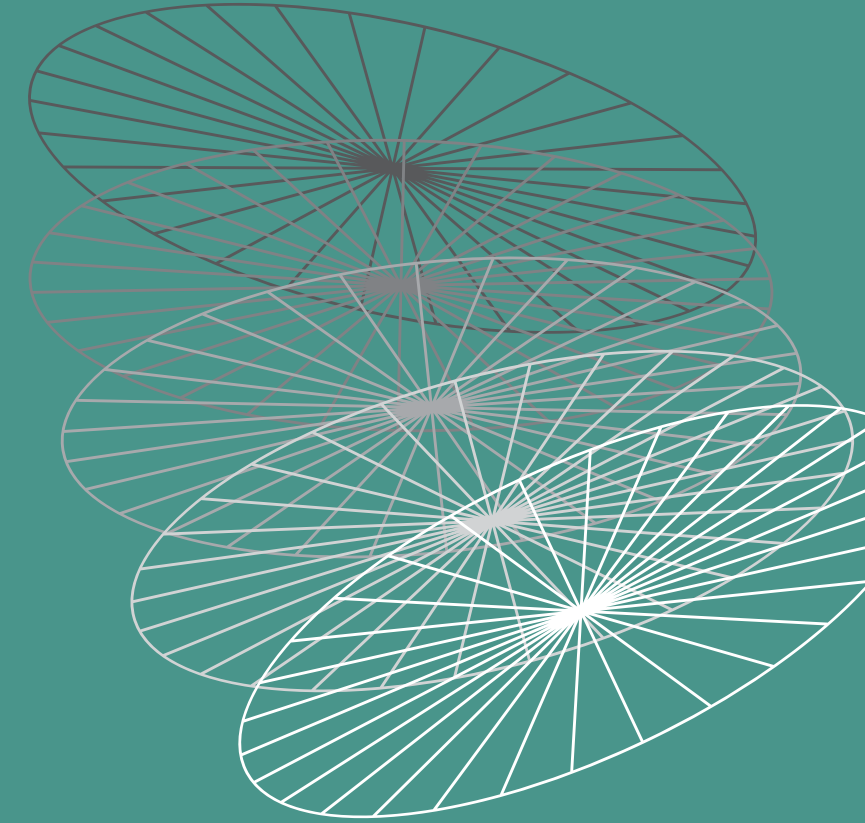
    SET v_date = v_date + INTERVAL 1 DAY;
    -- On change la valeur de la variable locale
    SELECT DATE_FORMAT(v_date, '%W %e %M %Y') AS Demain;
END bloc;$$

DELIMITER ;
```



Exemple avec variable utilisateur

```
CREATE DEFINER='root'@'localhost' PROCEDURE `var_utilisateur`()  
BEGIN  
  SET @x = 15;  
  SET @y = 10;  
  SELECT @x, @y, @x-@y;  
END
```



Portée des variables locales dans un bloc d'instructions

Les variables locales n'existent que dans le bloc d'instructions dans lequel elles ont été déclarées. Dès que le mot-clé `END` est atteint, toutes les variables locales du bloc sont détruites.

```
BEGIN
  SELECT 'Bloc d''instructions principal';

  BEGIN
    SELECT 'Bloc d''instructions 2, imbriqué dans le bloc principal';

    BEGIN
      SELECT 'Bloc d''instructions 3, imbriqué dans le bloc d''instructions 2';
    END;
  END;

  BEGIN
    SELECT 'Bloc d''instructions 4, imbriqué dans le bloc principal';
  END;
END;
```

Exemple 1

Appelez cette procédure et remarquez ce qu'elle fait

```
• CREATE DEFINER='root'@'localhost' PROCEDURE `bloc1`()
  BEGIN
    DECLARE v_test1 INT DEFAULT 1;

    BEGIN
      DECLARE v_test2 INT DEFAULT 2;

      SELECT 'Imbriqué' AS Bloc;
      SELECT v_test1, v_test2;
    END;
    SELECT 'Principal' AS Bloc;
    SELECT v_test1, v_test2;
  END
```

Exemple 2

Appelez cette procédure et remarquez ce qu'elle fait

```
CREATE PROCEDURE `exemple1` ()  
BEGIN  
    DECLARE a INT DEFAULT 10;  
    DECLARE b, c INT;    /* Declaration de la variable locale */  
    SET a = a + 100;  
    SET b = 2;  
    SET c = a + b;  
    BEGIN    /* variable locale dans un bloc imbriqué */  
        DECLARE c INT;  
        SET c = 5;  
        /* variable locale c est prioritaire sur celle du même nom déclarée  
        dans le bloc englobant. */  
        SELECT a, b, c;  
    END;  
    SELECT a, b, c;  
END
```

Exercice 1

1. Créer une procédure qui ajoute un bonus de 10% à un salaire. La variable salaire est une variable locale.
2. Et si la variable salaire était la colonne salaire de la table employé?



Exercice 2

Créer une procédure qui affiche dans un bloc le nombre d'employés de l'entreprise et dans l'autre la moyenne de leurs salaires



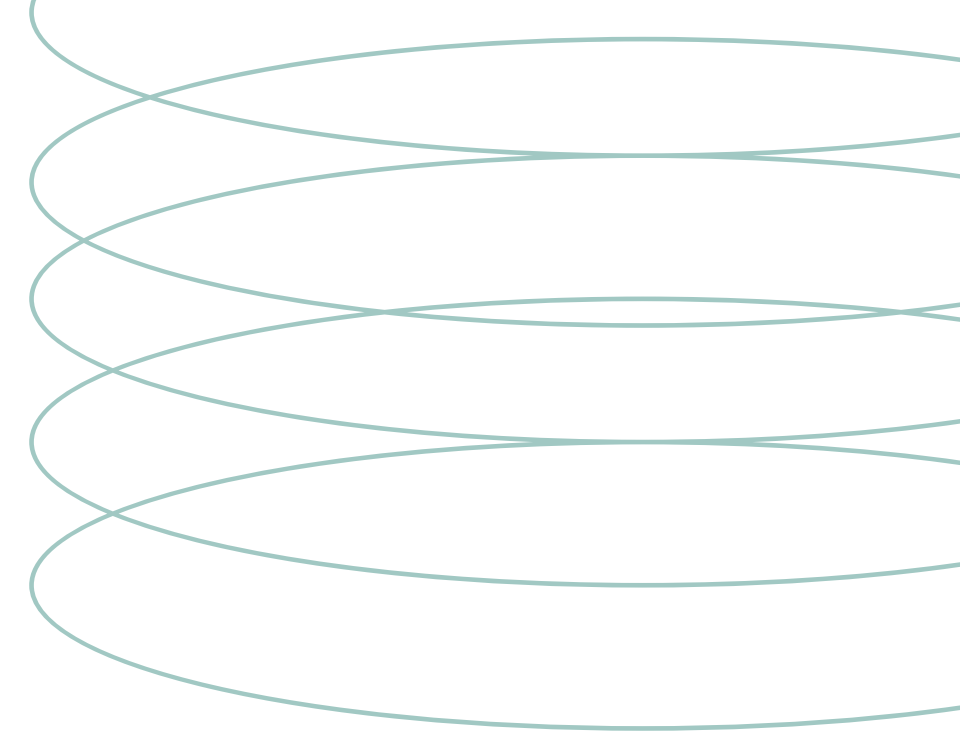
Les paramètres

Un paramètre peut être de trois sens différents : entrant (IN), sortant (OUT), ou les deux (INOUT).

IN : c'est un paramètre "entrant". C'est-à-dire qu'il s'agit d'un paramètre dont la valeur est fournie à la procédure stockée. Cette valeur sera utilisée pendant la procédure (pour un calcul ou une sélection, par exemple).

OUT : il s'agit d'un paramètre "sortant", dont la valeur sera établie au cours de la procédure et qui pourra ensuite être utilisé en dehors de cette procédure.

INOUT : un tel paramètre sera utilisé pendant la procédure, verra éventuellement sa valeur modifiée par celle-ci, et sera ensuite utilisable en dehors.



Syntaxe



Lorsque l'on crée une procédure avec un ou plusieurs paramètres, chaque paramètre est défini par trois éléments.

Son sens : entrant, sortant, ou les deux. Si aucun sens n'est donné, il s'agira d'un paramètre IN par défaut.

Son nom : indispensable pour le désigner à l'intérieur de la procédure.

Son type : INT, VARCHAR(10)...

[IN | OUT | INOUT] nom_paramètre type[(taille)]



Exemple IN

```
CREATE PROCEDURE my_proc_IN (IN var1 INT)
```

```
BEGIN
```

```
SELECT * FROM jobs LIMIT var1;
```

```
END$$
```

ce qui donnera le résultat:

```
mysql> CALL my_proc_in(2)$$
```

JOB_ID	JOB_TITLE	MIN_SALARY	MAX_SALARY
AD_PRES	President	20000	40000
AD_VP	Administration Vice President	15000	30000

```
2 rows in set (0.00 sec)Query OK, 0 rows affected (0.03 sec)
```

Exemple OUT

```
CREATE PROCEDURE my_proc_OUT (OUT highest_salary INT)
BEGIN
SELECT MAX(MAX_SALARY) INTO highest_salary FROM JOBS;
END$$
```

ce qui donnera le résultat:

```
mysql> CALL my_proc_OUT(@M)$$
Query OK, 1 row affected (0.03 sec)
```

```
mysql< SELECT @M$$+-----+
| @M      |
+-----+
| 40000   |
+-----+
1 row in set (0.00 sec)
```

Exemple INOUT

```
CREATE PROCEDURE my_proc_INOUT (INOUT mfgender INT, IN emp_gender CHAR(1))  
BEGIN  
    SELECT COUNT(gender) INTO mfgender FROM user_details WHERE gender = emp_gender;  
END$$
```

ce qui donnera le résultat:

```
mysql> CALL my_proc_INOUT(@C, 'M')$$  
Query OK, 1 row affected (0.02 sec)
```

```
mysql> SELECT @C$$  
+-----+  
| @C    |  
+-----+  
|      3 |  
+-----+  
1 row in set (0.00 sec)
```

```
mysql> CALL my_proc_INOUT(@C, 'F')$$  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT @C$$  
+-----+  
| @C    |  
+-----+  
|      1 |  
+-----+  
1 row in set (0.00 sec)
```

Les instructions conditionnelles

Les structures conditionnelles permettent de déclencher une action ou une série d'instructions lorsqu'une condition préalable est remplie.

MySQL propose deux structures conditionnelles : IF et CASE.

L'instruction IF

L'instruction IF implémente une structure conditionnelle de base dans un programme enregistré et doit être terminée par un point-virgule.

- Si la condition est vraie, la ou les clauses THEN ou ELSEIF correspondantes sont exécutées.
- Si aucune condition ne correspond, l'instruction ou les instructions de la clause ELSE s'exécutent.
- Chaque instruction consiste en une ou plusieurs instructions SQL ; les instructions vides ne sont pas autorisées.

Syntaxe

```
IF condition THEN statement(s)  
[ELSEIF condition THEN statement(s)] ...  
[ELSE statement(s)]  
END IF
```

Exemple 1

```
CREATE PROCEDURE controle_if(IN var INT)
BEGIN
    IF var = 1 THEN
        SELECT 'Il est vrai de dire que 1 = 1 :D';
    ELSEIF var = 2 THEN
        SELECT 'Il est vrai de dire que 2 = 2 :D';
    ELSE
        SELECT 'La variable passée en paramètre vaut autre chose que 1 et 2';
    END IF;
END
```


Exemple 2

```
CREATE PROCEDURE controle_if(IN var INT)
BEGIN
    IF var = 1 THEN
        SELECT 'Il est vrai de dire que 1 = 1 :D';
    ELSEIF var = 2 THEN
        SELECT 'Il est vrai de dire que 2 = 2 :D';
        SELECT 'Je suis 2';
    ELSE
        SELECT 'La variable passée en paramètre vaut autre chose que 1 et 2';
        SELECT var;
    END IF;
END |
```

Example 3

```
• CREATE DEFINER=`root`@`localhost` PROCEDURE `GetUserName`(INOUT user_name varchar(16),  
  IN user_id varchar(16))  
  BEGIN  
    DECLARE uname varchar(16);  
    SELECT last_name INTO uname  
    FROM actor  
    WHERE actor_id = user_id;  
    IF user_id = "scott123"  
    THEN  
      SET user_name = "Scott";  
    ELSEIF user_id = "ferp6734"  
    THEN  
      SET user_name = "Palash";  
    ELSEIF user_id = "diana094"  
    THEN  
      SET user_name = "Diana";  
    END IF;  
  END
```

L'instruction CASE

- cette instruction permet d'exprimer des actions pour plusieurs cas de figure
- Il existe deux syntaxes afin de le faire:
 - **Première syntaxe : conditions d'égalité**
 - **Seconde syntaxe : toutes conditions**

Syntaxe 1

```
CASE valeur_a_comparer  
  WHEN possibilite1 THEN instructions  
  [WHEN possibilite2 THEN instructions] ...  
  [ELSE instructions]  
END CASE;
```

Exemple 1

```
CREATE PROCEDURE case1(IN var INT)
BEGIN
    CASE var
    WHEN 1 THEN SELECT 'Je suis 1';
    WHEN 2 THEN SELECT 'Je suis 2';
    ELSE SELECT 'Je suis autre chose que 1 et 2';
    END CASE;
END
```

Application

```
CREATE DEFINER='root'@'localhost' PROCEDURE `pref_acteurs`(IN num_acteur INT)
BEGIN
    DECLARE prenom VARCHAR(10);

    SELECT first_name INTO prenom
    FROM actor
    WHERE actor_id = num_acteur;

    CASE prenom
        WHEN 'LIZA' THEN -- Première possibilité
            SELECT 'j'aime la musique' AS first_name;
        WHEN 'M' THEN -- Deuxième possibilité
            SELECT 'j'aime lire' AS first_name;
        ELSE -- Défaut
            SELECT 'Je suis en plein questionnement existentiel...' AS first_name;
    END CASE;
END
```

Syntaxe 2

CASE

WHEN condition THEN instructions

[WHEN condition THEN instructions] ...

[ELSE instructions]

END CASE

Explications

- Chaque expression de condition de recherche de la clause **WHEN** est évaluée jusqu'à ce que l'une d'entre elles soit vraie, auquel cas la liste d'instructions de la clause **THEN** correspondante est exécutée.
- Si aucune condition de recherche n'est égale, la clause **ELSE** de la liste d'instructions s'exécute, si elle existe.
- Chaque **statement_list** est constituée d'une ou plusieurs instructions **SQL** ; une **statement_list** vide n'est pas autorisée.

Exemple 1

```
CREATE PROCEDURE case2(IN var INT)
BEGIN
    CASE
        WHEN var = 1 THEN SELECT 'Je suis 1';
        WHEN var = 2 THEN SELECT 'Je suis 2';
        ELSE SELECT 'Je suis autre chose que 1 et 2';
    END CASE;
END
```

Application

Ecrire une procedure stockée qui prend un id d'un film et rend s'il est sortie avant, aprée ou en 2010

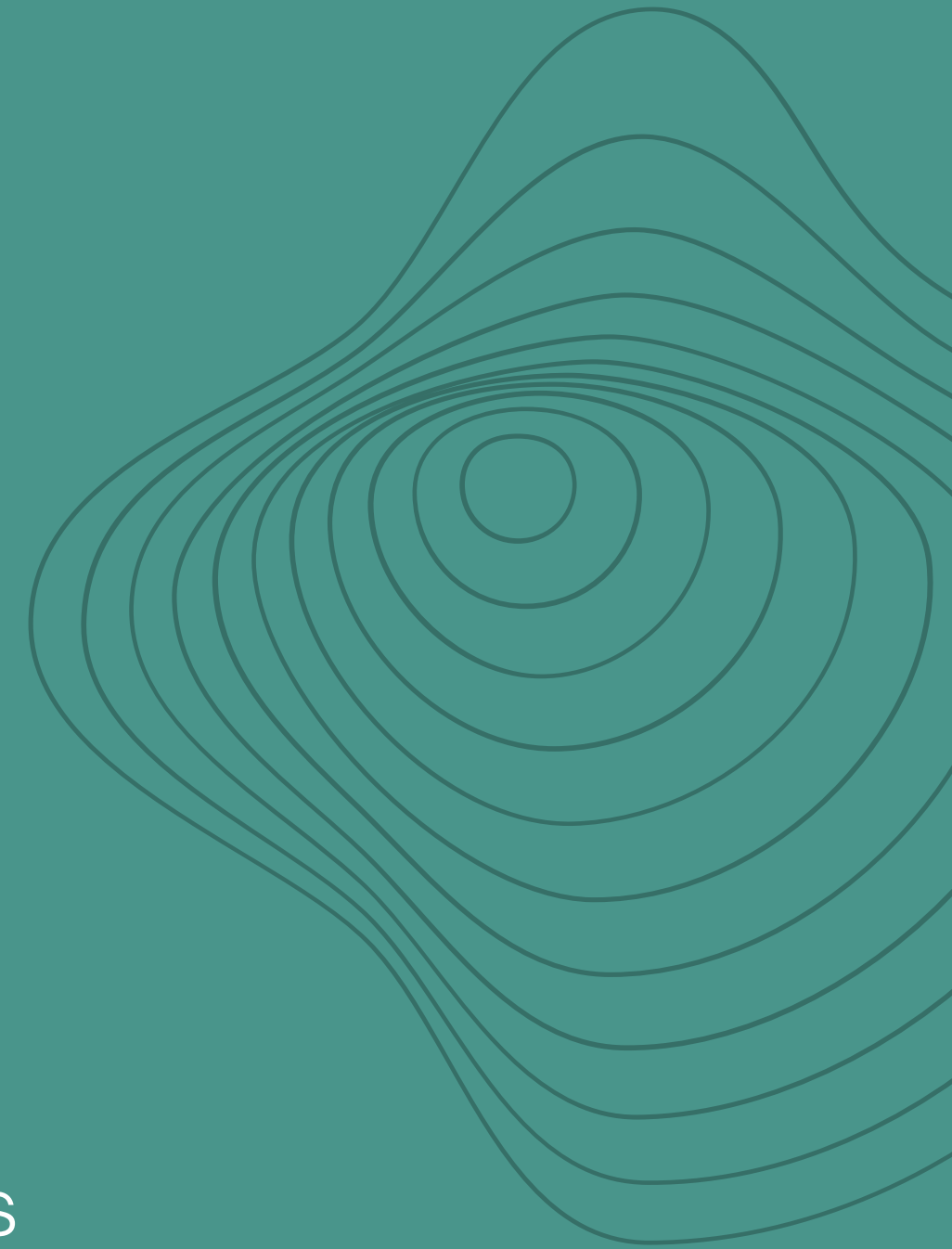
Correction

```
CREATE PROCEDURE `avant_apres_2010_case` (IN p_film_id INT, OUT p_message VARCHAR(100))  
BEGIN  
    DECLARE v_annee INT;  
  
    SELECT release_year INTO v_annee  
    FROM film  
    WHERE film_id = p_film_id;  
  
    CASE  
        WHEN v_annee < 2010 THEN  
            SET p_message = 'Je suis sortie avant 2010.';  
        WHEN v_annee = 2010 THEN  
            SET p_message = 'Je suis sortie en 2010.';  
        ELSE  
            SET p_message = 'Je suis sortie après 2010.';  
        END CASE;  
END
```

L'instruction LOOP

```
[begin_label:]  
    LOOP  
statement_list  
    END LOOP  
[end_label]
```

La boucle est utilisée pour répéter l'exécution d'instructions, ces instructions sont terminées par " ;". En général, l'instruction LEAVE est utilisée pour quitter la construction de la boucle. Dans une fonction stockée, RETURN peut également être utilisé, ce qui permet de sortir entièrement de la fonction. Une instruction LOOP peut être étiquetée.



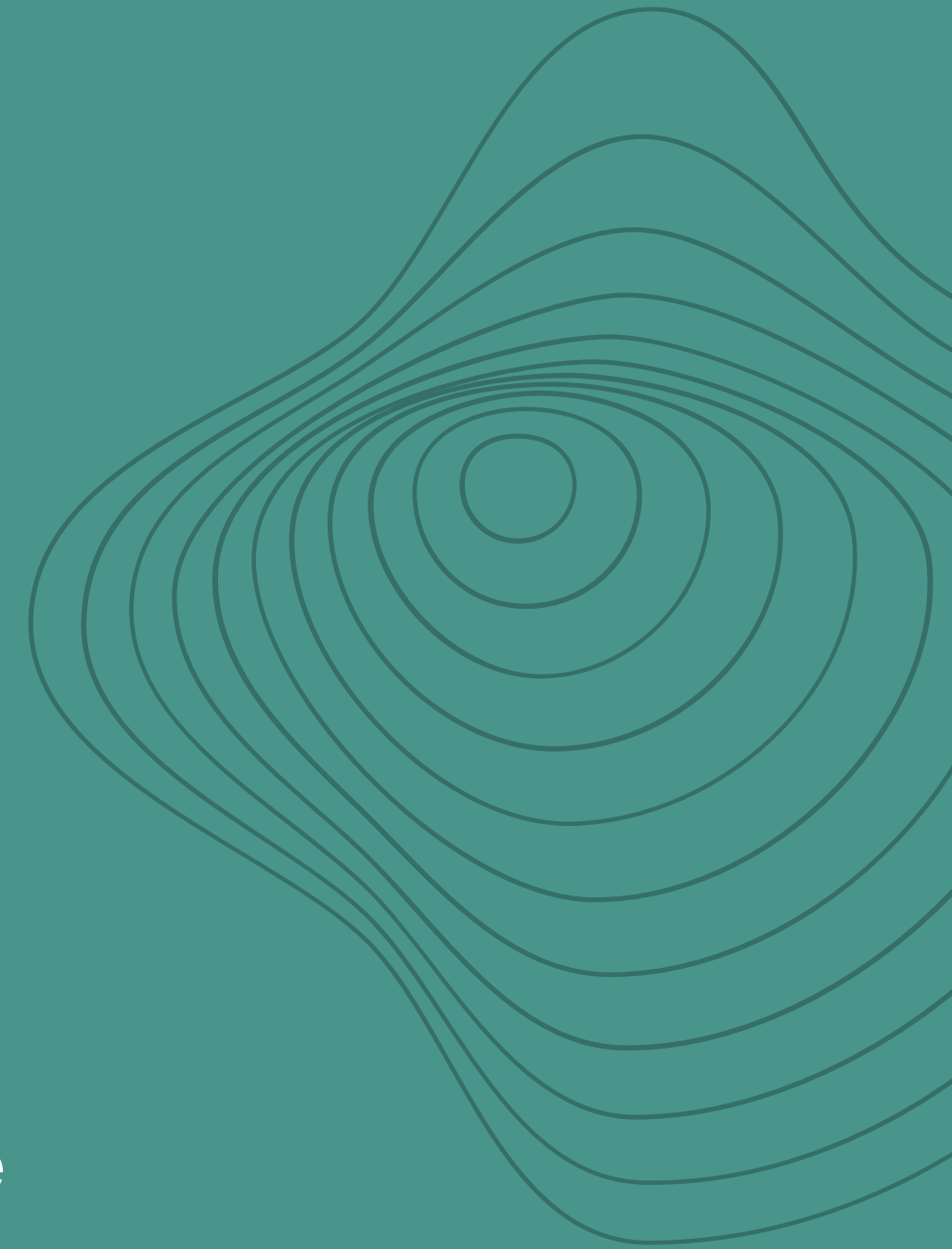
Example

```
DELIMITER $$
CREATE PROCEDURE `my_proc_LOOP` (IN num INT)
BEGIN
    DECLARE x INT;
    SET x = 0;
loop_label: LOOP
    INSERT INTO number VALUES (rand());
    SET x = x + 1;
    IF x >= num
    THEN
        LEAVE loop_label;
    END IF;
END LOOP;
END$$
```

L'instruction REPEAT

```
[begin_label:] REPEAT  
    statement_list  
    UNTIL search_condition  
END  
REPEAT [end_label]
```

L'instruction REPEAT exécute la ou les instructions de manière répétée tant que la condition est vraie. La condition est vérifiée à chaque fois à la fin des instructions.



Exemple

```
DELIMITER |
CREATE PROCEDURE compter_jusque_repeat(IN p_nombre INT)
BEGIN
    DECLARE v_i INT DEFAULT 1;

    REPEAT
        SELECT v_i AS nombre;

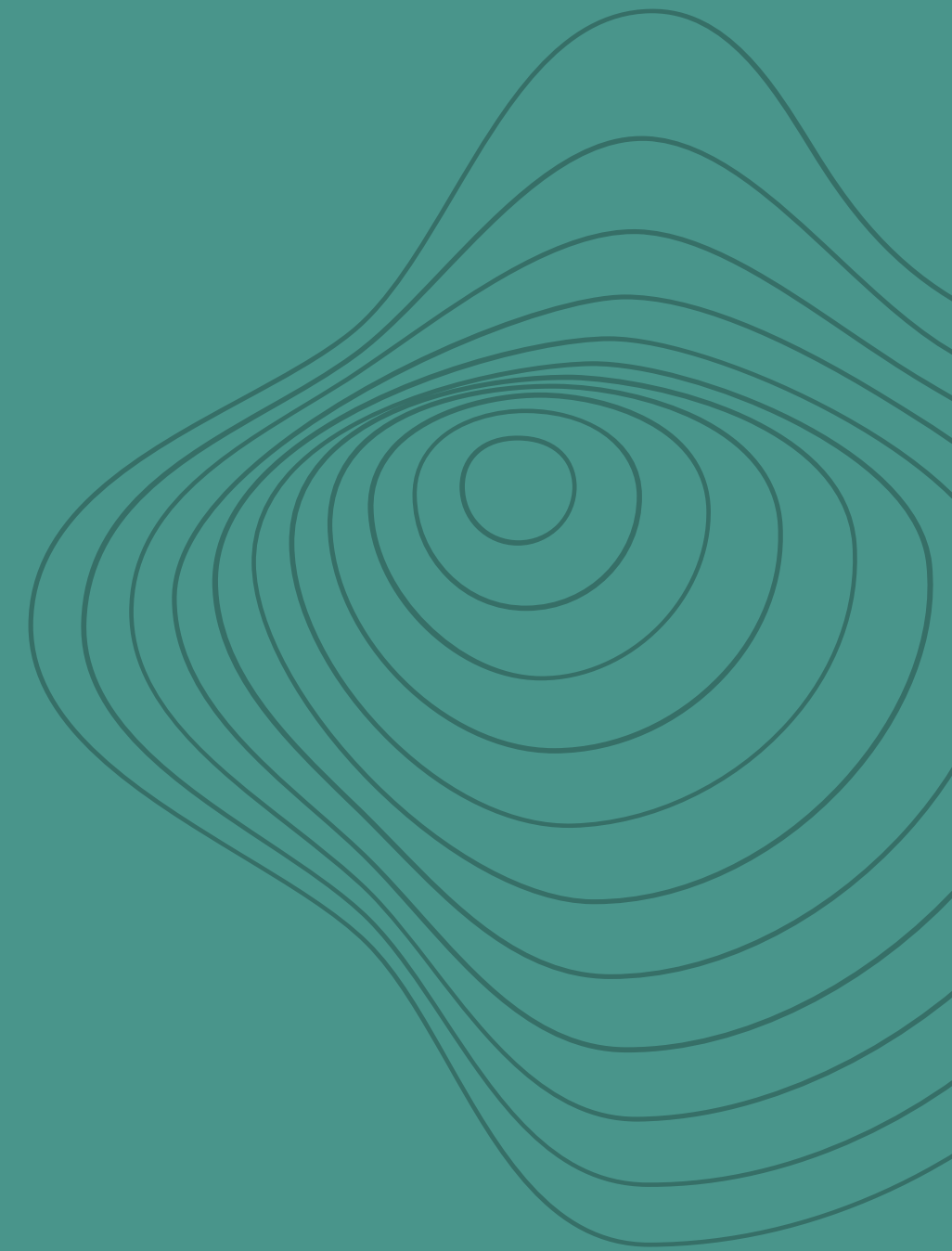
        SET v_i = v_i + 1;
        -- À ne surtout pas oublier, sinon la condition restera vraie
    UNTIL v_i > p_nombre END REPEAT;
END |
DELIMITER ;

CALL compter_jusque_repeat(3);
```

L'instruction WHILE

```
[begin_label:] WHILE search_condition DO  
    statement_list  
END WHILE [end_label]
```

L'instruction WHILE exécute la ou les instructions tant que la condition est vraie. La condition est vérifiée à chaque fois au début de la boucle.



Exemple 1

```
DELIMITER |
CREATE PROCEDURE compter_jusque_while(IN p_nombre INT)
BEGIN
    DECLARE v_i INT DEFAULT 1;

    WHILE v_i <= p_nombre DO
        SELECT v_i AS nombre;

        SET v_i = v_i + 1;
        -- À ne surtout pas oublier, sinon la condition restera vraie
    END WHILE;
END |
DELIMITER ;

CALL compter_jusque_while(3);
```

Exemple 2

```
DELIMITER $$  
CREATE PROCEDURE my_proc_WHILE(IN n INT)  
BEGIN  
    SET @sum = 0;  
    SET @x = 1;  
    WHILE @x < n  
    DO  
        IF mod(@x, 2) <> 0 THEN  
            SET @sum = @sum + @x;  
        END IF;  
        SET @x = @x + 1;  
    END WHILE;  
END$$  
  
CALL my_proc_WHILE(5)$$  
  
SELECT @sum$$
```

L'instruction Iterate, Leave

Iterate Label | Leave Label

ITERATE signifie exécuter la boucle à nouveau sur une étiquette spécifique. ITERATE ne peut apparaître qu'à l'intérieur des instructions LOOP, REPEAT et WHILE.

L'instruction LEAVE est utilisée pour quitter la construction de contrôle de flux qui a l'étiquette donnée. Si l'étiquette correspond au bloc de programme stocké le plus éloigné, LEAVE quitte le programme. LEAVE peut être utilisé dans les constructions BEGIN END ou des constructions en boucle (LOOP, REPEAT, WHILE).



Exemple LEAVE

```
DELIMITER |
CREATE PROCEDURE test_leave1(IN p_nombre INT)
BEGIN
    DECLARE v_i INT DEFAULT 4;

    SELECT 'Avant la boucle WHILE';

    while1: WHILE v_i > 0 DO

        SET p_nombre = p_nombre + 1; -- On incrémente le nombre de 1

        IF p_nombre%10 = 0 THEN    -- Si p_nombre est divisible par 10,
            SELECT 'Stop !' AS 'Multiple de 10';
            LEAVE while1; -- On quitte la boucle WHILE.
        END IF;

        SELECT p_nombre; -- On affiche p_nombre
        SET v_i = v_i - 1; -- Attention de ne pas l'oublier

    END WHILE while1;

    SELECT 'Après la boucle WHILE';
END|
DELIMITER ;

CALL test_leave1(3); -- La boucle s'exécutera 4 fois
```

Exemple Iterate

```
DELIMITER |
CREATE PROCEDURE test_iterate()
BEGIN
    DECLARE v_i INT DEFAULT 0;

    boucle_while: WHILE v_i < 3 DO
        SET v_i = v_i + 1;
        SELECT v_i, 'Avant IF' AS message;

        IF v_i = 2 THEN
            ITERATE boucle_while;
        END IF;

        SELECT v_i, 'Après IF' AS message;
        -- Ne sera pas exécuté pour v_i = 2
    END WHILE;
END |
DELIMITER ;

CALL test_iterate();
```


Syntaxe

ALTER PROCEDURE nom_procedure

Exemple

```
ALTER PROCEDURE my_proc_WHILE  
COMMENT 'Modify Comment';
```

```
SHOW CREATE PROCEDURE; -- afin de voir les changements faits
```


Syntaxe

DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name

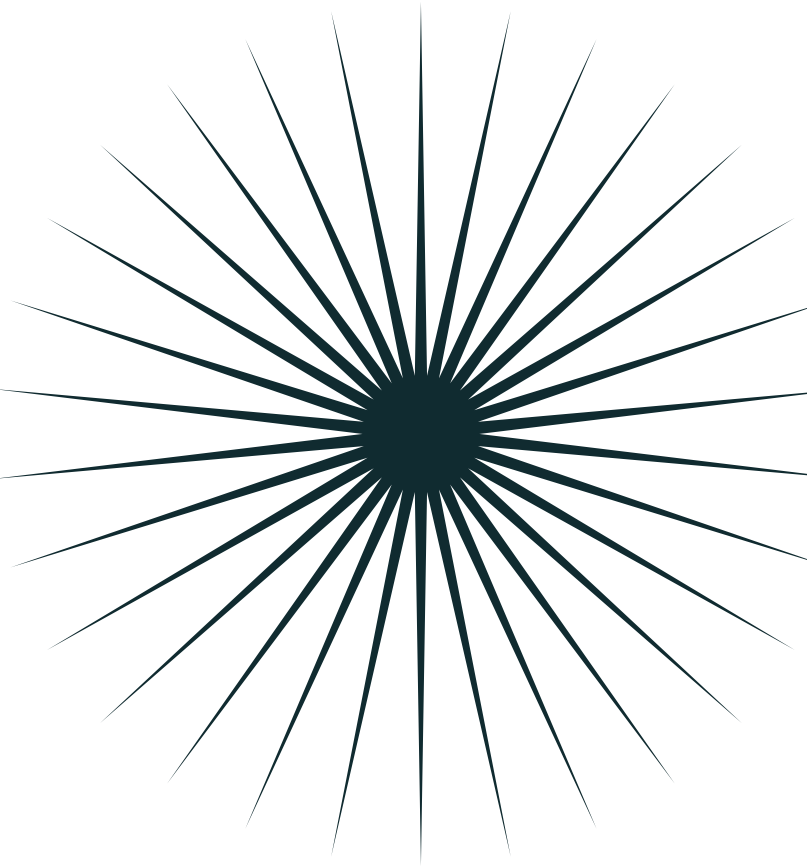

La clause IF EXISTS est une extension de MySQL. Elle empêche une erreur de se produire si la procédure ou la fonction n'existe pas.



Les curseurs



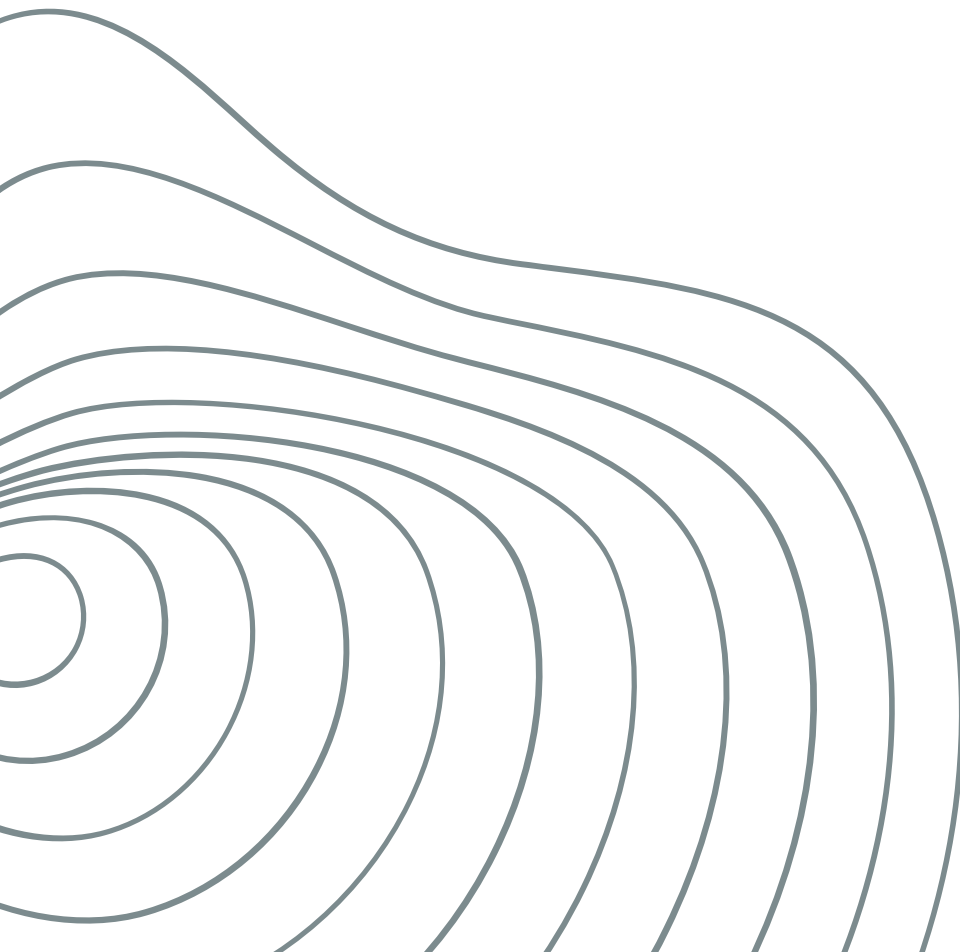

Il est possible de consulter les données retournées par un select en utilisant l'instruction `SELECT INTO`, mais elle n'est utilisée que pour les requêtes qui ne retournent qu'une seule ligne de résultat. Les curseurs, une structure de contrôle, résolvent ce problème car ils permettent d'itérer sur les résultats d'une requête select quel que soit le nombre de résultats retournés.

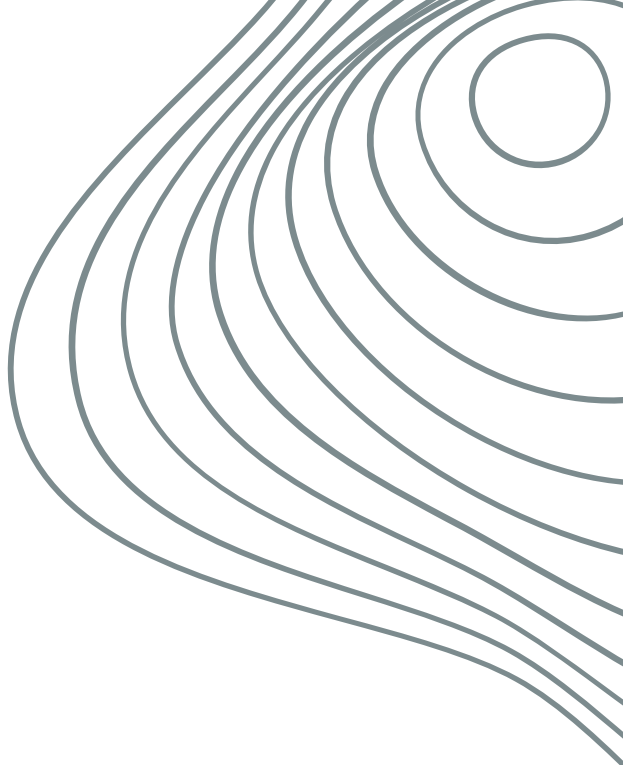
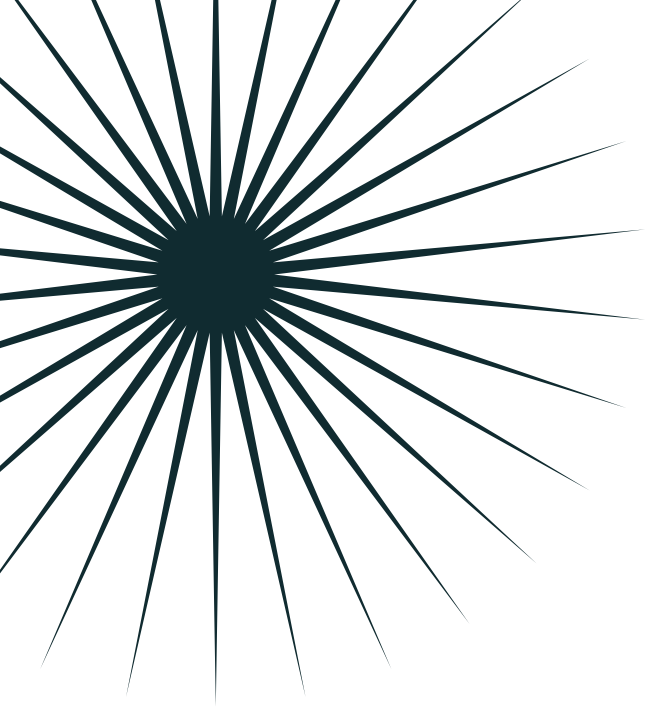




Syntaxe

Pour utiliser les curseurs dans les procédures MySQL, vous devez procéder comme suit :

- Déclarer un curseur
`DECLARE nom_curseur CURSOR FOR expression_select;`
 - Ouvrir un curseur
`OPEN nom_curseur;`
 - Récupérer la date dans les variables
`FETCH nom_curseur INTO variable(s);`
 - Une fois terminé, fermer le curseur
`CLOSE nom_curseur;`
- 
- 



```
DELIMITER |
CREATE PROCEDURE parcours_deux_clients()
BEGIN
    DECLARE v_nom, v_prenom VARCHAR(100);

    DECLARE curs_clients CURSOR
        FOR SELECT nom, prenom -- Le SELECT récupère deux colonnes
        FROM Client
        ORDER BY nom, prenom;
    -- On trie les clients par ordre alphabétique
```

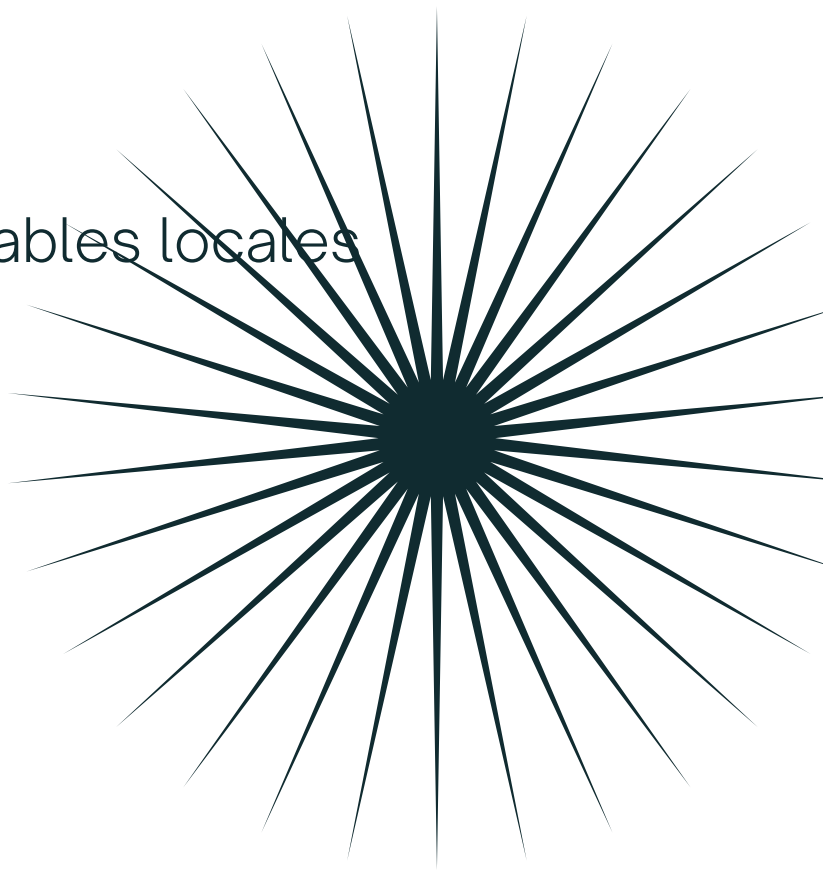
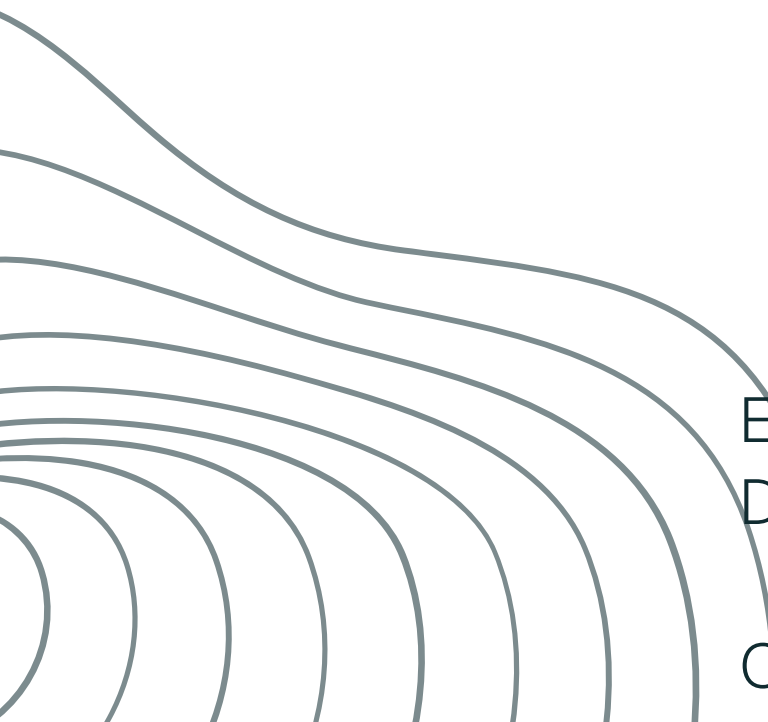
```
OPEN curs_clients; -- Ouverture du curseur
```

```
FETCH curs_clients INTO v_nom, v_prenom;
-- On récupère la première ligne et on assigne les valeurs récupérées à nos variables locales
SELECT CONCAT(v_prenom, ' ', v_nom) AS 'Premier client';
```

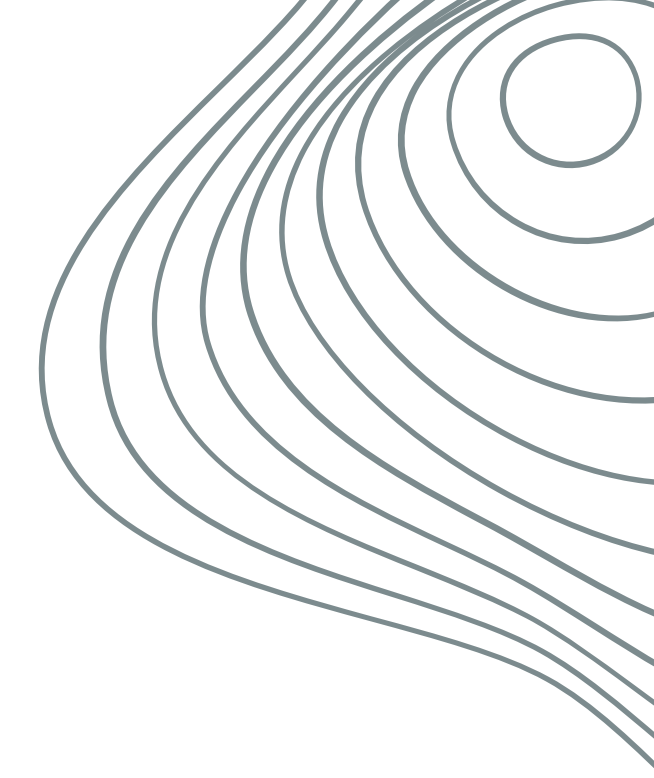
```
FETCH curs_clients INTO v_nom, v_prenom;
-- On récupère la seconde ligne et on assigne les valeurs récupérées à nos variables locales
SELECT CONCAT(v_prenom, ' ', v_nom) AS 'Second client';
```

```
CLOSE curs_clients; -- Fermeture du curseur
END|
DELIMITER ;
```

```
CALL parcours_deux_clients();
```



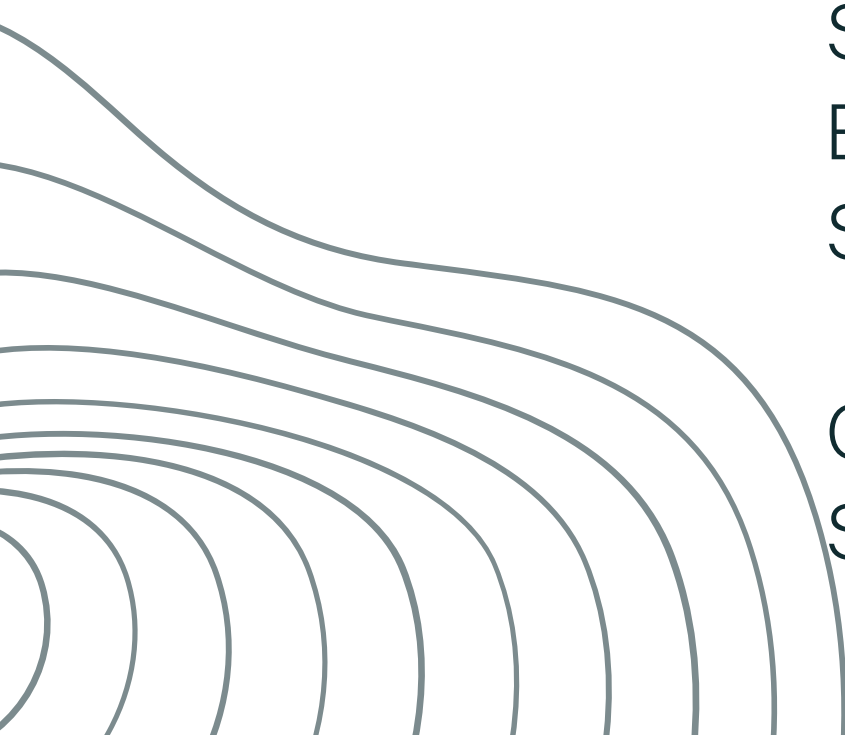
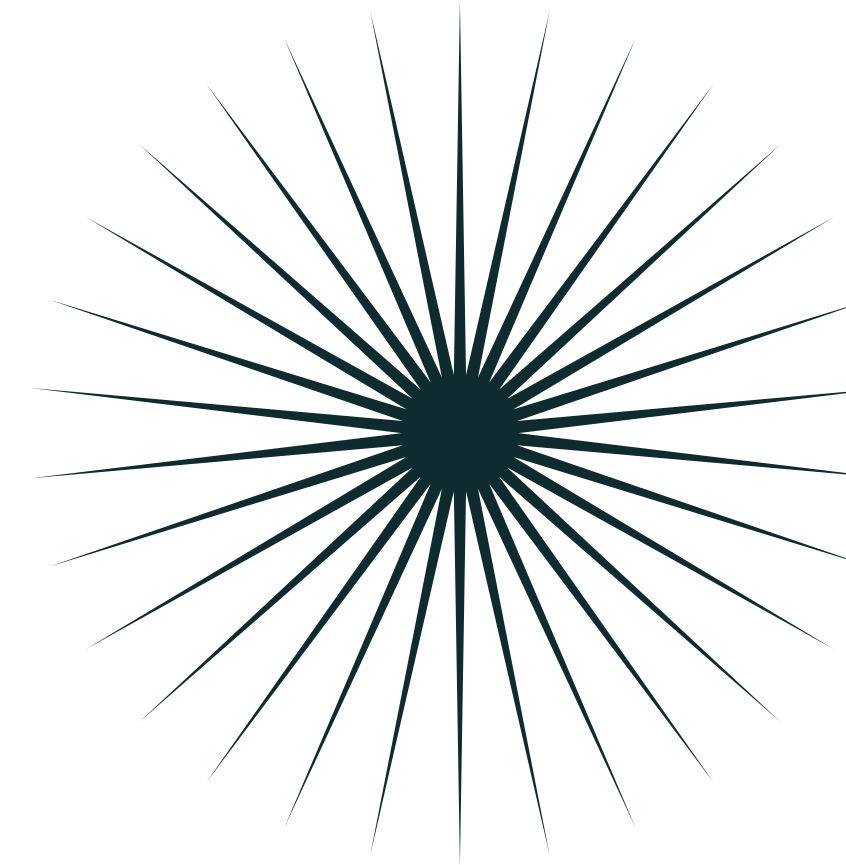
Exemple 1



Exemple 2

```
DELIMITER $$
CREATE PROCEDURE my_procedure_cursors(INOUT return_val INT)
BEGIN
  DECLARE a,b INT;
  DECLARE cur_1 CURSOR FOR
  SELECT max_salary FROM jobs;
  DECLARE CONTINUE HANDLER FOR NOT FOUNDSET b = 1;
  OPEN cur_1;REPEATFETCH cur_1 INTO a;
  UNTIL b = 1END REPEAT;
  CLOSE cur_1;
  SET return_val = a;
  END;
  $$

CALL my_procedure_cursors(@R)$$
SELECT @R$$
```



Explications

- SQL sécurité : SQL SECURITY, peut être défini comme SQL SECURITY DEFINER ou SQL SECURITY INVOKER pour spécifier le contexte de sécurité ; c'est-à-dire, si la routine s'exécute en utilisant les privilèges du compte nommé dans la clause routine DEFINER ou de l'utilisateur qui l'invoque.
- Type de routine : peut être soit (CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA), cela dépend si la procédure contient du SQL (CONTAINS SQL) ou pas (NO SQL), ou si la déclaration modifie des données SQL (MODIFIES SQL DATA) ou si elle lit seulement des données (READS SQL DATA).

Avez-vous des questions ?

Envoyez-les-moi ! J'espère que vous avez appris quelque chose de nouveau.

