Repaso Unidad II

IS501-0900 Bases de Datos I

Ing. José Manuel Inestroza

Alumno

- Edgar Josué Benedetto Godoy
- 20171033802

III PAC 2020

11 - Diciembre - 2020

009 Factorial MySQL con procedimientos Almacenados

Recursividad Máxima en MySQL:

Por defecto existe un **máximo de recursividad en MySQL** y se debe modificar la **variable max_sp_recursion_depht** que se encuentra en la variable SESSION que al ser una **varible especial se usa @@**

Pasos para la creación del procedimiento almacenado **factorial** :

```
1.  Ampliar la recursión máxima
SET @@SESSION.max_sp_recursion_depht = 25$$

2.  Borrar el procedimiento previamente --
DROP PROCEDURE IF EXISTS sp_factorial$$

3. Valor de entrada IN y valor de salida OUT
CREATE PROCEDURE sp_factorial(IN N INT, OUT FACT INT)

4. Inicio de procedimiento --
BEGIN
    IF N = 1 THEN
        SELECT 1 INTO FACT;
    ELSE
        CALL sp_factorial(N-1, @TEMP);
        SELECT N * @TEMP INTO FACT;
    END IF;

5. Fin del procedimiento --
END$$
```

Los procedimientos almacenados usan el prefijo "sp", ejem: *sp_nombreProcedimiento*. Para guardar valores en una variable se puede usar el INTO o el " = ".

Pasos para el llamado del procedimiento almacenado **factorial** :

```
-- Crear variable --
SET @fact = 0;

-- Llamado procedimiento --
CALL sp_factorial(5,@fact);

-- Mostrar el resultado en pantalla --
SELECT @fact AS "Factorial de 5";
```

---

010 Evaluaciones y comparaciones en SQL usando CASE

La sentencia CASE:

- Es útil para mostrar diferentes tipos de información.
- Es un procedimiento en el cual se aplica un caso de selección / uso.
- Una de las implementaciones del CASE es transformar algo a boolean.

Ejemplo del uso del CASE:

```
-- Se define una variable con el contenido de un caso de uso  --
-- Creación de variable --
SET @sampleCategory = "folder";

-- Se aplica un caso de selección / uso --
SELECT
    CASE
        -- Se compara la variable con la igualdad y el condicional generará una
respuesta en el Case
        WHEN @sampleCategory = "folder" THEN "01f02"
        WHEN @sampleCategory = "file" THEN "01f03"
        ELSE "UNK000"
    END AS "Type of item as a Code"
;
```

Extracción del value de un JSON:

De una variable tipo json se puede obtener el value de alguna key usando **JSON_VALUE**, se especifica la tupla (que en alguna de sus pocisiones posea un json) y se especfica la Key de la cual se extraera el Value. Se usa la nomenclatura **$** para el nombre de la *key*. También se puede usar **JSON_UNQUOTE** junto con **JSON_EXTRACT** para obtener un resultado similar.

Ejemplo de extracción del **value** de un **JSON** asignando el resultado a una variable:

```
-- Usando JSON_VALUE
SET @lastCommand = JSON_VALUE(@lastRequest, "$.command");

-- Usando JSON_UNQUOTE junto a JSON_EXTRACT
SET @lastCommand = JSON_UNQUOTE(JSON_EXTRACT(@lastRequest, "$.command"))
```

Ejemplo del CASE usando JSON:

```
DROP DATABASE IF EXISTS AdvancedSQLProcedures;

-- Crear base de datos --
CREATE DATABASE AdvancedSQLProcedures;

USE AdvancedSQLProcedures;

DROP TABLE IF EXISTS RequestQueue;

CREATE TABLE RequestQueue(
    id INT AUTO_INCREMENT PRIMARY KEY,
    jso_request JSON NOT NULL COMMENT "Petición API en forma de JSON",
    bit_read BIT NOT NULL COMMENT "Estado de la petición: atendida o no atendida"
) COMMENT "Tabla de peticiones de usuario en forma de cola";

INSERT INTO RequestQueue(jso_request, bit_read) VALUES
    ('{"service": "00f21x2", "user": "bdi", "command": "INBOX"}', 1 ),
    ('{"service": "00f21x3" , "user": "bdi", "command": "TRASH"}', 0),
;

-- Seleccione ultimo registro de peticion no atendido almacenando en un espacio de
memoria temporal --
SET @lastRequest = (SELECT jso_request FROM RequestQueue WHERE bit_read = 0 ORDER
BY id ASC LIMIT 1);

-- Obtener del último registro no atendido el comando requerido --
SET @lastCommand = JSON_VALUE(@lastRequest, "$.command");
SET @lastCommand = REPLACE(@lastCommand, '\"', "");

-- Se demuestra que las variables contienen la info deseada --
SELECT @lastRequest AS "La última petición en Queue", @lastCommand AS "Último
comando";

-- Se realiza un caso dependiendo del dato obtenido --
SELECT
    @lastCommand AS "Último comando",
    CASE
        WHEN @lastCommand = "INBOX" THEN "Solicitud de Inbox SMTP de la bandeja
```

```
del correo"
        WHEN @lastCommand = "TRASH" THEN "Solicitud de Trash SMTP de la bandeja
del correo"
        ELSE "Instrucción desconocida"
    END AS "Acción solicitada (=)",
    CASE
        WHEN @lastCommand LIKE "INBOX" THEN "Solicitud de Inbox SMTP de la bandeja
del correo"
        WHEN @lastCommand LIKE "TRASH" THEN "Solicitud de Trash SMTP de la bandeja
del correo"
        ELSE "Instrucción desconocida"
    END AS "Acción solicitada (LIKE)",
    CASE
        WHEN STRCMP(@lastCommand LIKE "INBOX") THEN "Solicitud de Inbox SMTP de la
bandeja del correo"
        WHEN STRCMP(@lastCommand LIKE "TRASH") THEN "Solicitud de Trash SMTP de la
bandeja del correo"
        ELSE "Instrucción desconocida"
    END AS "Acción solicitada (STRCMP)"
;

-- Se realiza un caso dependiendo del dato obtenido, limpiando la cadena con un
TRIM --
SELECT
    @lastCommand AS "Último comando",
    CASE
        WHEN TRIM(@lastCommand) = "INBOX" THEN "Solicitud de Inbox SMTP de la
bandeja del correo"
        WHEN TRIM(@lastCommand) = "TRASH" THEN "Solicitud de Trash SMTP de la
bandeja del correo"
        ELSE "Instrucción desconocida"
    END AS "Acción solicitada (=)",
    CASE
        WHEN TRIM(@lastCommand) LIKE "INBOX" THEN "Solicitud de Inbox SMTP de la
bandeja del correo"
        WHEN TRIM(@lastCommand) LIKE "TRASH" THEN "Solicitud de Trash SMTP de la
bandeja del correo"
        ELSE "Instrucción desconocida"
    END AS "Acción solicitada (LIKE)",
    CASE
        WHEN STRCMP(TRIM(@lastCommand) LIKE "INBOX") THEN "Solicitud de Inbox SMTP
de la bandeja del correo"
        WHEN STRCMP(TRIM(@lastCommand) LIKE "TRASH") THEN "Solicitud de Trash SMTP
de la bandeja del correo"
        ELSE "Instrucción desconocida"
    END AS "Acción solicitada (STRCMP)"
;
```

---

011 Expresiones regulares en MySQL

Importante:

1. **RLIKE** Es un **LIKE** pero con expresiones regulares.
2. Para buscar exactitud al inicio y al final de la expresión regular se usa **"^ exp $"**.
3. El **"+"** implica que se tendran **1 o más digitos**. Al escribir la expresión regular como cadena se tiene que escapar la pleca.

Ejemplo de busqueda de patrones

```
SET @record = '{"name" : "María García", "age" : 20, "uid" : "0801199926544"}';

-- Patron usando expresiones regulares --
SET @pattern = "^08\\d+$";

SELECT
    JSON_UNQUOTE(JSON_EXTRACT(@record, "$.uid")) AS "UID",
    CASE
        WHEN (JSON_UNQUOTE(JSON_EXTRACT(@record, "$.uid")) RLIKE @pattern) = 0
THEN "FALSE"
        ELSE "TRUE"
    END AS "Cumple con el patrón"
;
```

Ejercicio

Se desea hacer un recorrido por ciertos departamentos del país para convencer a estudiantes que hagan estudios sobre STEM (Science, technology, engineering, and math). Para ello se requieren estudiantes de dichos departamentos.

Se desea hacer una tabla que muestre claramente los estudiantes que estan involucrados en los siguientes recorridos.

- **Recorrido 1** : Atlantida(01), El paraiso(07), Francisco Morazan(08) y Yoro(18)
- **Recorrido 2** : La Paz(12), Comayagua(03), Cortez(05)

Código en MySQL que cumple con los requisitos del ejercicio

```
DROP DATABASE IF EXISTS AdvancedSQLProcedures;

-- Crear base de datos --
CREATE DATABASE AdvancedSQLProcedures;

USE AdvancedSQLProcedures;

DROP TABLE IF EXISTS Student;
CREATE TABLE Student(
    id INT AUTO_INCREMENT PRIMARY KEY,
    jso_record JSON NOT NULL COMMENT "Documento con name, age y uid"
```

```
) COMMENT "Tabla de estudiantes EA";

INSERT INTO Student(jso_record) VALUES
    ('{"name" : "María García", "age" : 20, "uid" : "1801199926544"}'),
    ('{"name" : "Juan García", "age" : 20, "uid" : "0301199825644"}'),
    ('{"name" : "Laura Sanchez", "age" : 20, "uid" : "0507199901188"}'),
    ('{"name" : "María Martinez", "age" : 20, "uid" : "0801199920044"}'),
    ('{"name" : "Pedro Lopez", "age" : 20, "uid" : "0709199923262"}'),
    ('{"name" : "Fredy Pereira", "age" : 20, "uid" : "0101199236659"}'),
    ('{"name" : "Marlon García", "age" : 20, "uid" : "0708199952555"}');

-- Se usa un CASE para mostrar a que recorrido pertenece --
SELECT
    (JSON_UNQUOTE(JSON_EXTRACT(jso_record, "$.name"))) AS "Nombre del estudiante",
    CASE
        WHEN (JSON_UNQUOTE(JSON_EXTRACT(jso_record, "$.name"))) RLIKE "^((0[178])|
(18))\\d{11}$" THEN "Recorrido 1"
        WHEN (JSON_UNQUOTE(JSON_EXTRACT(jso_record, "$.name"))) RLIKE "^((0[35])|
(12))\\d{11}$" THEN "Recorrido 2"
    END AS "Recorrido"
FROM
    Student
WHERE
    (JSON_UNQUOTE(JSON_EXTRACT(jso_record, "$.name"))) RLIKE "^((0[13578])|
(1[28]))\\d{11}$"
ORDER BY
    ASC
;
```

Observaciones

- Para reconocer todas las identidades se puede usar esta expresión regular --> **\d{13}**
- El **peor uso** de expresiones regulares es **buscar literales** especificos --> (1801199926544)|(0301199825644)|(0507199901188)...
- Para reconocer lo que se pide en el ejercicio --> ((01)|(07)|(08)|(18)\d{11})
- Optimizando la expresión anterior --> **((0[178])|(18)\d{11})**