ceu-economics-and-business.github.io

Stored Procedures. – ECBS 5146 SQL and Different Shapes of Data

9–12 minutes

Overview

Teaching: 90 min

Questions

- How can simplify the maintainable of long queries with similarities?
- How can you build business logic with SQL?
- How can you clean/fix a database with corrupted/bad quality data unfit for your analytics?

Objectives

- Introducing procedural elements of SQL databases
- Introducing Stored Procedures with parameters
- Introducing IF/LOOP/CURSOR
- Understanding the difference of processing data in the database vs. outside of database engine
- Understanding the advantages and disadvantages of stored

procedures

- Example with fixing data

Keywords

#STORED PROCEDURES

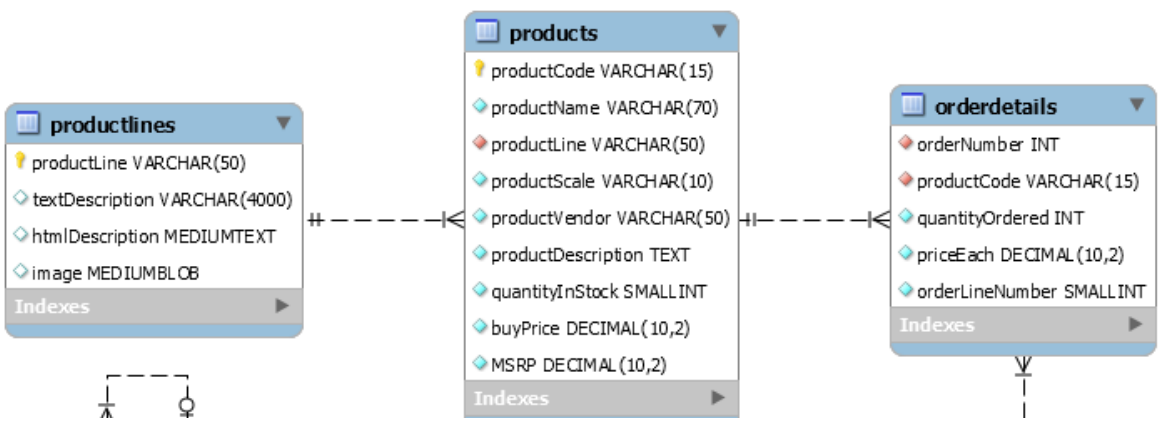
#REUSE CODE

#CLEANING DATA

Table of Content

Session setup

No need to load new data, in this chapter we will use the same sample db we used in the last chapter:



A basic stored procedure

Creating the stored procedure

```
DROP PROCEDURE IF EXISTS GetAllProducts;

DELIMITER //

CREATE PROCEDURE GetAllProducts()
BEGIN
        SELECT *  FROM products;
END //

DELIMITER ;
```

`NOTE1:` Mind the delimiter: the default delimiter in SQL is ";". In a stored procedure, you'll have potentially multiple statements ending with ";" - so you need the define a second delimiter to end the whole stored procedure. On the end of the routine, we will set the default delimiter back to ";"

`NOTE2:` You cannot edit a stored procedure, once created, you need to drop and recreate: `DROP PROCEDURE IF EXISTS ...`

Executing the stored procedure

A stored procedures with parameters

Input parameter with IN

The following example creates a stored procedure that finds all offices that locate in a country specified by the input parameter countryName

```
DROP PROCEDURE IF EXISTS GetOfficeByCountry;

DELIMITER //

CREATE PROCEDURE GetOfficeByCountry(
        IN countryName VARCHAR(255)
)
BEGIN
        SELECT *
                FROM offices
                        WHERE country = countryName;
END //
```

```
DELIMITER ;
```

Executing with multiple parameters

```
CALL GetOfficeByCountry('USA');
```

```
CALL GetOfficeByCountry('France');
```

```
CALL GetOfficeByCountry();
```

You will get error, because the paramter is mandatory

Exercise1

Create a stored procedure which displays the first X entries of payment table. X is IN parameter for the procedure.

Output parameter with OUT

The following stored procedure returns the number of orders by order status.

```
DROP PROCEDURE IF EXISTS GetOrderCountByStatus;

DELIMITER $$

CREATE PROCEDURE GetOrderCountByStatus (
        IN  orderStatus VARCHAR(25),
        OUT total INT
)
BEGIN
        SELECT COUNT(orderNumber)
        INTO total
```

```
        FROM orders
        WHERE status = orderStatus;
END$$
DELIMITER ;
```

Executing the procedure and displaying the result

```
CALL GetOrderCountByStatus('Shipped',@total);
SELECT @total;
```

Exercise2

Create a stored procedure which returns the amount for Xth entry of payment table. X is IN, amount is OUT parameter for the procedure. Display the returned amount.

Using the INOUT parameter

In this example, the stored procedure SetCounter() accepts one INOUT parameter (counter) and one IN parameter (inc). It increases the counter (counter) by the value of specified by the inc parameter.

```
DROP PROCEDURE IF EXISTS SetCounter;

DELIMITER $$

CREATE PROCEDURE SetCounter(
        INOUT counter INT,
        IN inc INT
)
```

```
BEGIN
        SET counter = counter + inc;
END$$
DELIMITER ;
```

Initializing the input parameter and repeating the execution and displaying result several times

```
SET @counter = 1;
CALL SetCounter(@counter,1);
SELECT @counter;
CALL SetCounter(@counter,1);
SELECT @counter;
CALL SetCounter(@counter,1);
SELECT @counter;
```

Example with IF and declaring variables

The IF syntax can have different forms:

- IF-THEN
- IF-THEN-ELSE
- IF-THEN-ELSEIF-ELSE

Assigning Customer Level based on credit. Mind the usage of credit variable used the procedure.

```
DROP PROCEDURE IF EXISTS GetCustomerLevel;

DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(
```

```
        IN  pCustomerNumber INT,
        OUT pCustomerLevel  VARCHAR(20)
)
BEGIN
        DECLARE credit DECIMAL DEFAULT 0;

        SELECT creditLimit
                INTO credit
                        FROM customers
                                WHERE customerNumber
= pCustomerNumber;

        IF credit > 50000 THEN
                SET pCustomerLevel = 'PLATINUM';
        ELSE
                SET pCustomerLevel = 'NOT PLATINUM';
        END IF;
END$$
DELIMITER ;
```

Execution for a specific customer

Calling the stored procedure for customer number 447 and show the value of the OUT parameter pCustomerLevel:

```
CALL GetCustomerLevel(447, @level);
SELECT @level;
```

Note: CASE instruction is also available. We will skip CASE because you can do the same with IF. Sometimes CASE looks

nicer or might be even faster for the interpreter.

Exercise3

Create a stored procedure which returns category of a given row in payments. Row number is IN parameter, while category is OUT parameter. Display the returned category. CAT1 - amount > 100.000, CAT2 - amount > 10.000, CAT3 - amount <= 10.000

Iterating with LOOP

A basic loop:

```
DROP PROCEDURE IF EXISTS LoopDemo;

DELIMITER $$
CREATE PROCEDURE LoopDemo()
BEGIN

        ceuloop: LOOP
                SELECT * FROM offices;
                IF TRUE THEN
                        LEAVE ceuloop;
                END IF;
        END LOOP ceuloop;
END$$
DELIMITER ;

CALL LoopDemo();
```

Exercise4

> Create a loop which counts to 5 and displays the actual count in each step as SELECT (eg. SELECT count)

Debugging/Logging

As you could see in Exercise4, displaying with SELECT is not ideal if you have a long loop. You better create a simple log table named “messages” and write your logs into it:

```
CREATE TABLE messages (message varchar(100) NOT NULL);
```

and add the next line instead of `SELECT x;`:

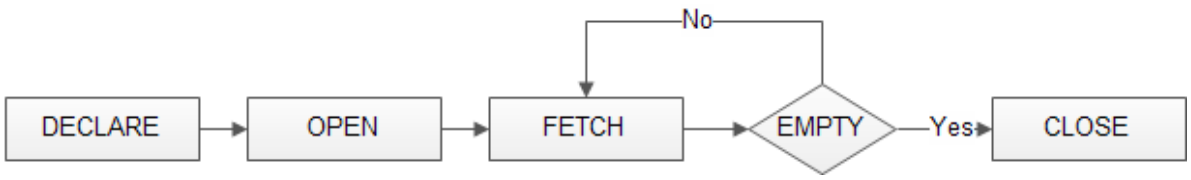
```
INSERT INTO messages SELECT CONCAT('x:',x);
```

also add `TRUNCATE messages;` before the loop.

After re-execution check messages:

Note: You can you other iterative commands instead of LOOP, but with the LOOP you can cover every case.

Iterating trough a table with CURSOR



Listing phones of customers:

```
DROP PROCEDURE IF EXISTS CursorDemo;

DELIMITER $$
```

```
CREATE PROCEDURE CursorDemo()
BEGIN
        DECLARE phone varchar(50);
        DECLARE finished INTEGER DEFAULT 0;
        -- DECLARE CURSOR
        DECLARE curPhone CURSOR FOR SELECT
customers.phone FROM classicmodels.customers;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET
finished = 1;
        -- OPEN CURSOR
        OPEN curPhone;
        TRUNCATE messages;
        myloop: LOOP
                -- FETCH CURSOR
                FETCH curPhone INTO phone;
                INSERT INTO messages SELECT
CONCAT('phone:',phone);
                IF finished = 1 THEN LEAVE myloop;
                END IF;
        END LOOP myloop;
        -- CLOSE CURSOR
        CLOSE curPhone;
END$$
DELIMITER ;


CALL CursorDemo();

SELECT * FROM messages;
```

> ## Exercise5
>
> Loop through orders table. Fetch orderNumber + shippedDate. Write in both fields into messages as one line.

Fixing US phones in customer table

The aim of the next snippet is to add the international prefix to US domestic format.

These are the possible formats in US:

- 754-3010 Local
- (541) 754-3010 Domestic
- +1-541-754-3010 International
- 1-541-754-3010 Dialed in the US

```
DROP PROCEDURE IF EXISTS FixUSPhones;

DELIMITER $$

CREATE PROCEDURE FixUSPhones ()
BEGIN
        DECLARE finished INTEGER DEFAULT 0;
        DECLARE phone varchar(50) DEFAULT "x";
        DECLARE customerNumber INT DEFAULT 0;
        DECLARE country varchar(50) DEFAULT "";

        -- declare cursor for customer
        DECLARE curPhone
                CURSOR FOR
```

```
                        SELECT
customers.customerNumber, customers.phone,
customers.country
                                FROM
classicmodels.customers;

        -- declare NOT FOUND handler
        DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET finished = 1;

        OPEN curPhone;

        -- create a copy of the customer table
        DROP TABLE IF EXISTS
classicmodels.fixed_customers;
        CREATE TABLE classicmodels.fixed_customers
LIKE classicmodels.customers;
        INSERT fixed_customers SELECT * FROM
classicmodels.customers;

        fixPhone: LOOP
                FETCH curPhone INTO
customerNumber,phone, country;
                IF finished = 1 THEN
                        LEAVE fixPhone;
                END IF;

                -- insert into messages select
concat('country is: ', country, ' and phone is: ',
phone);
```

```
                IF country = 'USA'  THEN
                        IF phone NOT LIKE '+%' THEN
                                IF LENGTH(phone) = 10
THEN
                                        SET  phone =
CONCAT('+1',phone);
                                        UPDATE
classicmodels.fixed_customers
                                                SET
fixed_customers.phone=phone

WHERE fixed_customers.customerNumber =
customerNumber;
                                END IF;
                        END IF;
                END IF;

        END LOOP fixPhone;
        CLOSE curPhone;

END$$
DELIMITER ;
```

Execute the procedure:

Check the resulted new table:

```
SELECT * FROM fixed_customers where country = 'USA';
```

Advantages/disadvantages of stored procedures

Advantages

- Embedded processing, no need to extract data to process it with an external procedural language or tool - this is potentially faster and reduces network traffic
- Maintainable code, avoiding duplicates
- Better security, better control over data access

Disadvantages

- Impact over server resources (CPU, memory)
- Debugging / Trouble shooting is not the most advanced
- Overall the business logic written in stored procedures can be written easier/nicer in other languages

Homework 5

- Continue the last script: complete the US local phones to international using the city code. Hint: for this you need to find a data source with domestic prefixes mapped to cities, import as a table to the database and add new business logic to the procedure.
- Upload the solution to your GitHub repo in a folder called HW5
- Submit GitHub repo link to moodle when you are ready