NOVA SCHOOL OF SCIENCE & TECHNOLOGY

MDE

TP1 – Class 4

SQL/PSM

2023 - 2024

TOPICS

- IF statement, CASE statement
- LOOP statement, WHILE loop and REPEAT loop
- CURSORS, cursor for loop
- SELECT INTO …
- TRIGGERS
- PROCEDURE
- FUNCTION
- Raise errors

SQL/PSM

- SQL/PSM is a procedural language extension to SQL used by MySQL. Its purpose is to combine database language and procedural programming language.

- SQL/PSM is derived directly from Oracle's PL/SQL.

Triggers

- In MySQL, a trigger is a stored program invoked automatically in response to an event such as insert, update, or delete that occurs in the associated table. For example, you can define a trigger that is invoked automatically before a new row is inserted into a table.

- MySQL supports triggers that are invoked in response to the INSERT, UPDATE or DELETE event.

https://www.mysqltutorial.org/mysql-triggers/

Triggers in MySQL

- Create triggers
- Drop triggers

- Create a BEFORE INSERT trigger
- Create an AFTER INSERT trigger

- Create a BEFORE UPDATE trigger
- Create an AFTER UPDATE trigger

- Create a BEFORE DELETE trigger
- Create an AFTER DELETE trigger

- Create multiple triggers for a table that have the same trigger event and time – MySQL 8.0 allows you to define multiple triggers for a table that have the same trigger event and time.

- Show triggers – list triggers in a database, table by specific patterns.

https://www.mysqltutorial.org/mysql-triggers/

Create Triggers

CREATE TRIGGER trigger_name
{**BEFORE** | **AFTER**} {**INSERT** | **UPDATE**| **DELETE** }
ON table_name **FOR EACH ROW**
trigger_body;

To distinguish between the value of the columns **BEFORE** and **AFTER** the DML has fired, you use the **NEW** and **OLD** modifiers.

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

https://www.mysqltutorial.org/create-the-first-trigger-in-mysql.aspx

Create Triggers – Example BEFORE INSERT

- Assume the invoice table

```
CREATE TABLE invoice (
  idinvoice INT AUTO_INCREMENT PRIMARY KEY,
  invoice_number VARCHAR(9) NOT NULL,
  date DATETIME NOT NULL,
  package VARCHAR(45) NOT NULL,
  state VARCHAR(45) NOT NULL
);
```

- Let's create a trigger that whenever a new invoice is inserted the date column is automatically filled in with the current date

```
CREATE TRIGGER before_invoice_insert
BEFORE INSERT
ON invoice FOR EACH ROW
   SET NEW.date = NOW();
```

- Show triggers:

Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer	character_set_client	collation_connection	Database Collation
before_invoice_insert	INSERT	invoice	SET NEW.date = NOW()	BEFORE	2023-03-29 18:17:06.52	NO_ZERO_IN_DATE,NO_ZERO_DATE,NO_ENGI...	root@localhost	utf8mb4	utf8mb4_general_ci	utf8mb4_general_ci

- Now, let's insert a new invoice:

```
INSERT INTO invoice (invoice_number, package, state)
VALUES('2023MAR04', 'basic', 'issued');
```

idinvoice	invoice_number	date	package	state
1	2023MAR04	2023-03-29 18:19:03	basic	issued
NULL	NULL	NULL	NULL	NULL

IF and CASE STATEMENT

- MySQL simple IF-THEN statement

```
IF condition THEN
  statements;
END IF;
```

- MySQL simple IF-THEN-ELSE statement

```
IF condition THEN
  statements;
ELSE
  else-statements;
END IF;
```

https://www.mysqltutorial.org/mysql-if-statement/
https://www.mysqltutorial.org/mysql-case-function/

- MySQL simple IF-THEN-ELSEIF-ELSE statement

```
IF condition THEN
  statements;
ELSEIF elseif-condition THEN
  elseif-statements;
...
ELSE
  else-statements;
END IF;
```

- MySQL simple CASE statement

```
CASE value
  WHEN value1 THEN result1
  WHEN value2 THEN result2
  …
  [ELSE else_result]
END
```

Create Triggers – Example BEFORE UPDATE

- Let's create a trigger that whenever the date of an invoice is updated it is checked if the new date if earlier to the current date

```
DELIMITER $$$
CREATE TRIGGER update_invoice_date
BEFORE UPDATE
ON invoice FOR EACH ROW

BEGIN
   DECLARE current_day datetime;

   SET current_day = NOW();
   IF (date(NEW.date) < current_day) THEN
         signal sqlstate '45000' set message_text = 'Date cannot be earlier than current date';
   END IF;

END; $$$
DELIMITER ;
```

https://www.mysqltutorial.org/mysql-signal-resignal/

- Try:

```
UPDATE invoice set date = '2023-03-23 12:17:04'
WHERE idinvoice =  1 ;
```

and

```
UPDATE invoice set date = '2023-03-30 18:47:54'
WHERE idinvoice =  1 ;
```

PROCEDURE

- Sintaxe

```
CREATE PROCEDURE procedure_name (list_parameters)
BEGIN
   [declaration_section] % local variables
END [procedure_name];
```

The parameters are optional.
There are three types of parameters:
IN,
OUT,
IN OUT

- A stored procedure is a segment of declarative SQL statements stored inside the MySQL Server.
- A stored procedure can have parameters so you can pass values to it and get the result back. For example, you can have a stored procedure that returns customers by country and city. In this case, the country and city are parameters of the stored procedure.
- A stored procedure may contain control flow statements such as IF, CASE, and LOOP that allow you to implement the code in the procedural way.
- A stored procedure can call other stored procedures or stored functions, which allows you to modulate your code.

https://www.mysqltutorial.org/introduction-to-sql-stored-procedures.aspx

PROCEDURE - EXAMPLE

```
DELIMITER $$$
CREATE PROCEDURE contract_price_average(IN startDate datetime, OUT average_price decimal(7,2))
BEGIN
    SELECT avg(price) INTO average_price
    FROM contract
    WHERE date_start > startDate;
END; $$$
DELIMITER ;
```

https://www.mysqltutorial.org/mysql-stored-procedure/mysql-delimiter/

contract

- idcontract INT
- date_start DATETIME
- date_end DATETIME
- price DECIMAL(7,2)

Indexes

	idcontract	date_start	date_end	price
▶	1	2017-11-01 ...	2022-10-01...	36.25
	2	2018-08-01 ...	2023-09-01...	86.45
	3	2019-09-05 ...	2024-10-01...	36.25
	4	2020-09-02 ...	2025-10-01...	54.85
	5	2018-09-01 ...	2023-10-02...	54.85
	6	2017-09-02 ...	2022-10-03...	36.25
•	NULL	NULL	NULL	NULL

```
call contract_price_average ('2018-08-01 00:00:00', @avg_price);
select @avg_price;
```

	@avg_price
▶	48.65

PROCEDURE - EXAMPLE

```
DELIMITER $$$
CREATE PROCEDURE insert_contract(IN startDate datetime, IN endDate datetime, IN price decimal(7,2), IN idClient int, IN idPack int)

BEGIN
    INSERT INTO contract(date_start, date_end, price, client_idclient, package_type_idpackage_type)
    VALUES (startDate, endDate, price, idClient, idPack) ;

    SELECT * FROM contract;

END; $$$
DELIMITER ;

call insert_contract('2023-03-29', '2024-02-01', 75.23, 1, 1);
```

FUNCTION

- Sintaxe

```
CREATE FUNCTION function_name (list_parameters)
   RETURN return_datatype
BEGIN
      [declaration_section]
     executable_section
END;
```

A stored function is a special kind of stored program that **returns a single value.** Typically, you use stored functions to encapsulate common formulas or business rules that are reusable among SQL statements or stored programs.

https://www.mysqltutorial.org/mysql-stored-function/

The parameters are optional.
There are three types of parameters:
IN,
OUT,
IN OUT

FUNCTION - EXAMPLE

```
DELIMITER $$$
CREATE FUNCTION get_total_contract_price()
RETURNS decimal(7,2)
BEGIN
    DECLARE total DECIMAL(7,2);

    SELECT sum(price) INTO total
    FROM contract;

    RETURN total;

END; $$$
DELIMITER ;
```

```
select get_total_contract_price();
```

	get_total_contract_price()
▶	304.90

contract

- idcontract INT
- date_start DATETIME
- date_end DATETIME
- price DECIMAL(7,2)

Indexes ▶

	idcontract	date_start	date_end	price
▶	1	2017-11-01 ...	2022-10-01...	36.25
	2	2018-08-01 ...	2023-09-01...	86.45
	3	2019-09-05 ...	2024-10-01...	36.25
	4	2020-09-02 ...	2025-10-01...	54.85
	5	2018-09-01 ...	2023-10-02...	54.85
	6	2017-09-02 ...	2022-10-03...	36.25
*	NULL	NULL	NULL	NULL

LOOP, WHILE and REPEAT loop STATEMENT

- MySQL LOOP statement

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

- MySQL terminate LOOP statement (using LEAVE statement)

```
[label]: LOOP
   ...
   -- terminate the loop
   IF condition THEN
      LEAVE [label];
   END IF;
   ...
END LOOP;
```

https://www.mysqltutorial.org/stored-procedures-loop.aspx

- MySQL WHILE loop statement

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

https://www.mysqltutorial.org/mysql-stored-procedure/mysql-while-loop/

- MySQL REPEAT loop statement

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

https://www.mysqltutorial.org/mysql-stored-procedure/mysql-repeat-loop/

CURSORS, cursor for loop

- To handle a result set inside a stored procedure, you use a cursor. A cursor allows you to iterate a set of rows returned by a query and process each row individually.

- You can use MySQL cursors in stored procedures, stored functions, and triggers.

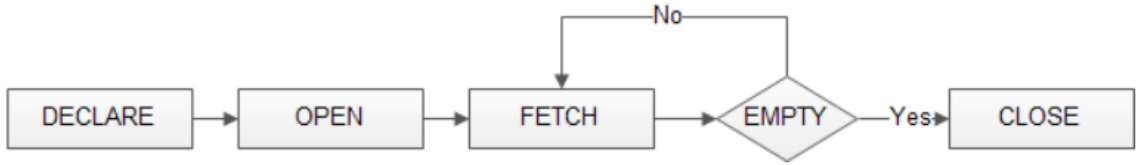
- How MySQL CURSOR works:

```
DECLARE cursor_name CURSOR FOR SELECT_statement;

OPEN cursor_name;

FETCH cursor_name INTO variables list;

CLOSE cursor_name;
```



https://www.mysqltutorial.org/mysql-cursor/

CURSORS, cursor for loop - EXAMPLE

Procedure that creates a telephone list of all clients in the client table

```
DELIMITER $$
CREATE PROCEDURE createTelephoneList (
    INOUT telephoneList varchar(1000)
)
BEGIN
    DECLARE is_ended INTEGER DEFAULT 0;
    DECLARE tel varchar(25) DEFAULT "";

    -- declare cursor for client telephone
    DECLARE currentPhone
        CURSOR FOR
            SELECT telephone FROM client;

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

    OPEN currentPhone;

    getTelephone: LOOP
        FETCH currentPhone INTO tel;
        IF is_ended = 1 THEN
            LEAVE getTelephone;
        END IF;
        -- build telephone list
        SET telephoneList = CONCAT(tel,";",telephoneList);
    END LOOP getTelephone;

    CLOSE currentPhone;

END$$
DELIMITER ;
```

```
SET @telephoneList = "";
CALL createTelephoneList(@telephoneList);
SELECT @telephoneList;
```

	@telephoneList
▶	+351 91 452 77 22;+351 91 333 77 22;+351 91 788 77 22;+351 91 666 77 22;+351 91 777 77 22;+351 91 444 77 22;

RECALLING…

- Triggers:
  - Use to program the response/reaction to events that occur in a DB.
  - Guarantee the integrity of the information, when this is difficult to guarantee through the ERD.

- Procedures:
  - Use when the objective is to modify information in a DB schema.
  - Use when it is necessary to use more than one “OUT” parameter.
  - Use to allow indirect access to information and not to tables.

- Anonymous Procedures:
  - Block of code without an assigned name, executed immediately and not saved in the database.

- Functions:
  - Use when the objective is just to obtain information from the database and perform calculations. **DO NOT CHANGE BD INFORMATION WITHIN A FUNCTION.**
  - Only one output parameter (with the “return” instruction).
  - Use to allow indirect access to information and not to tables.

FINAL REMARKS

- Finish the remaining requirements that use triggers, functions and procedures
- OPTIONAL: Build a Web application with a friendly interface for visualizing and manipulating the data stored in the database
- This comprises everything to finish it all.
- Enjoy!!