THỰC HÀNH QUẢN TRỊ DỮ LIỆU (CT467)

BUỔI 2 - MYSQL NÂNG CAO

Nội dung:

- Tạo thủ tục, hàm và con trỏ
- Tạo transaction và trigger

I. TẠO THỦ TỤC, HÀM VÀ CON TRỎ

1.1 Thủ tục (Stored Procedure)

Thủ tục là một đối tượng trong hệ quản trị CSDL bao gồm các câu lệnh SQL, chúng được kết hợp lại với nhau thành một khối lệnh, dùng để thực hiện một số công việc nào đó như cập nhật, thêm mới, xóa, hiển thị, tính toán và có thể trả về các giá trị.

Thủ tục hệ thống: những thủ tục do SQL cung cấp, tên tiếp đầu ngữ ***sp_***

Thủ tục người dùng: do người dùng tạo ra, tên tiếp đầu ngữ ***usp_***

Tạo thủ tục:

CREATE PROCEDURE sp_name ([IN | OUT | INOUT] param_name type)

BEGIN

<Đoạn chương trình xử lý>

END;

Lưu ý: có 3 loại tham số

❖ **IN:** chế độ mặc định, không bị thay đổi nếu như trong thủ tục có tác động đến.

Ví dụ:

```
1  DELIMITER $$
2  CREATE PROCEDURE getById(IN id INT(11))
3  BEGIN
4      /*Code*/
5  END; $$
6  DELIMITER;
```

Để không bị báo lỗi thêm câu lệnh trong khung đỏ

Ghi chú: Nếu muốn truyền vào nhiều hơn một tham số thì ta sẽ ngăn cách nó bởi dấu phẩy (,)

Chạy Procedure (thực thi - gọi hàm): CALL getById();

❖ **OUT:** chế độ này nếu như trong thủ tục có tác động thay đổi thì nó sẽ thay đổi theo. Biến truyền vào phải có chữ @ đằng trước (ví dụ @title)

Ví dụ:

```
DELIMITER $$
DROP PROCEDURE IF EXISTS changeTitle $$
CREATE PROCEDURE changeTitle(OUT title VARCHAR(255))
BEGIN
    SET title = 'Hoc lap trinh MySQL';
END;
DELIMITER;
```

Chạy Procedure (thực thi - gọi hàm):

```
CALL changeTitle(@title);

SELECT @title;
```

❖ **INOUT:** đây là sự kết hợp giữa IN và OUT. Nghĩa là có thể gán giá trị trước và có thể thay đổi nếu trong thủ tục có tác động tới.

Ví dụ:

```
DELIMITER $$

DROP PROCEDURE IF EXISTS counter $$

CREATE PROCEDURE counter(INOUT number INT(11))
BEGIN
    SET number = number + 1;
END;
DELIMITER;
```

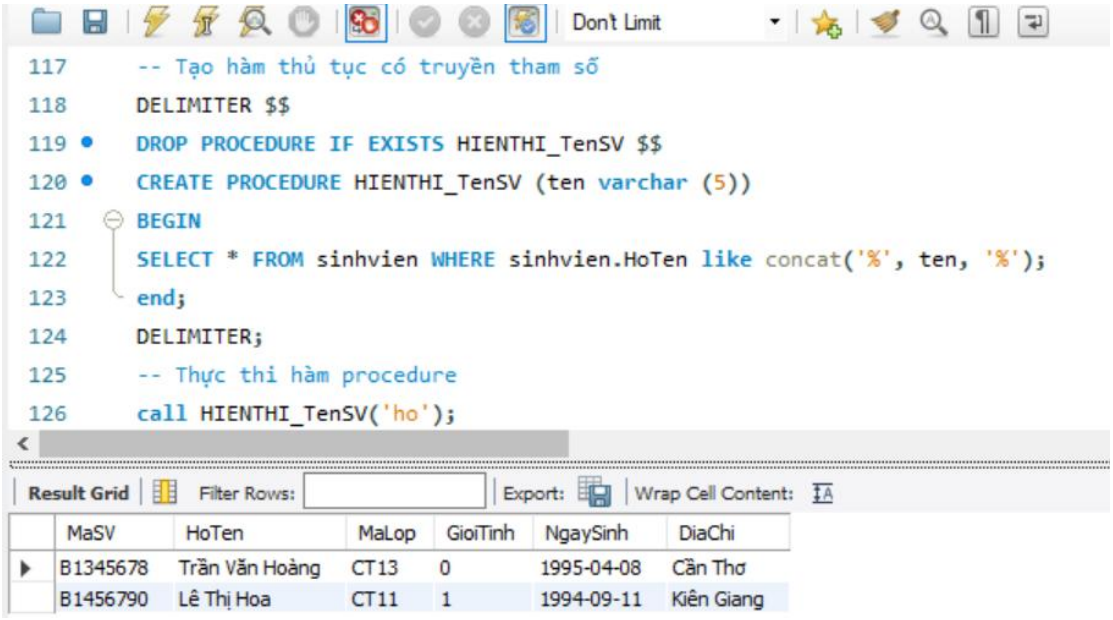
Chạy Procedure (thực thi - gọi hàm):

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

Ví dụ:

Tạo thủ tục: Hiển thị tất cả những sinh viên có chứa mã chỉ định. Điều kiện là trong đó tên sinh viên có chứa ký tự "ho"

Lời giải:



```
117    -- Tạo hàm thủ tục có truyền tham số
118    DELIMITER $$
119    DROP PROCEDURE IF EXISTS HIENTHI_TenSV $$
120    CREATE PROCEDURE HIENTHI_TenSV (ten varchar (5))
121    BEGIN
122    SELECT * FROM sinhvien WHERE sinhvien.HoTen like concat('%', ten, '%');
123    end;
124    DELIMITER;
125    -- Thực thi hàm procedure
126    call HIENTHI_TenSV('ho');
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

MaSV	HoTen	MaLop	GioiTinh	NgaySinh	DiaChi
B1345678	Trần Văn Hoàng	CT13	0	1995-04-08	Cần Thơ
B1456790	Lê Thị Hoa	CT11	1	1994-09-11	Kiên Giang

❖ Các câu lệnh điều khiển

Các câu lệnh điều khiển là một thành phần không thể thiếu của bất kỳ ngôn ngữ lập trình nào. Trong phần này sẽ giới thiệu cú pháp của các lệnh điều khiển trong MySQL. Các câu lệnh điều khiển tương tự với các ngôn ngữ lập trình đã học như C và Java chỉ được trình bày cú pháp. Chi tiết về các lệnh này có thể được tham khảo tại địa chỉ: https://dev.mysql.com/doc/refman/5.7/en/flow-control-statements.html

Lệnh rẽ nhánh **IF … ELSE**:

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

Lệnh lựa chọn **CASE**:

Lựa chọn 1 trong các khối lệnh để thực hiện tùy và giá trị của một biến hay mộtbiểu thức điều kiện.

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Hoặc:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Lệnh **ITERATE**:

- Dùng để thực hiện chu kỳ lặp mới (tương tự lệnh **CONTINUE** trong các ngôn ngữ lập trình C hoặc Java). Lệnh này chỉ có thể xuất hiện trong các lệnh lặp **LOOP**, **REPEAT** và **WHILE**.

  Cú pháp: ITERATE *label*

Lệnh **LEAVE**:

- Dùng để thoát ra khỏi 1 vòng lặp hoặc cả chương trình. Lệnh này có thể sử dụng trong các lệnh lặp **LOOP**, **REPEAT** và **WHILE** hoặc cặp lệnh **BEGIN ... END**. Lệnh này tương tự như lệnh BREAK trong ngôn ngữ lập trình C và Java trong trường hợp sử dụng bên trong vòng lặp.

  Cú pháp: LEAVE *label*

Lệnh lặp **LOOP**:

- Đây là một lệnh lặp đơn giản, cho phép thực hiện lặp lại một đoạn lệnh. Lệnh này không có điều kiện lặp nên phải sử dụng cùng với các lệnh **LEAVE** để thoát khỏi vòng lặp (kết hợp với lệnh **IF** hoặc **CASE** để kiểm tra điều kiện) và lệnh **ITERATE** để lặp lại vòng lặp tại 1 điểm nào đó trong vòng lặp.

  Cú pháp:

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

Ví dụ: Thủ tục sau nhận vào 1 số nguyên n và tính tổng từ 1 đến n.

```
CREATE PROCEDURE testLoop(n INT)
BEGIN
  DECLARE tong INT DEFAULT 0;
  DECLARE i INT DEFAULT 1;
  count_loop: LOOP
    IF i > n THEN
      LEAVE count_loop;
    END IF;
    SET tong=tong+i;
    SET i=i+1;
  END LOOP count_loop;
  SELECT tong as 'Tong 1->n';
END
```

```
MariaDB [dbms]> call testLoop(3);
+-----------+
| Tong 1->n |
+-----------+
|         6 |
+-----------+
1 row in set (0.00 sec)
```

Lệnh lặp **REPEAT:**

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

- Lệnh **REPEAT** thực hiện lặp lại thân hàm cho đến khi đến khi điều kiện trong mệnh đề **UNTIL** thỏa thì thoát ra khỏi vòng lặp. Vòng lặp này có thể kết hợp với nhãn và các lệnh **LEAVE**, **ITERATE** để ngưng hoặc lặp lại theo một điều kiện ngoại lệ nào đó khác với điều kiện trong mệnh đề **UNTIL**.

Ví dụ (https://dev.mysql.com/doc/refman/5.7/en/repeat.html):

```
mysql> delimiter //
mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->     SET @x = 0;
    ->     REPEAT
    ->       SET @x = @x + 1;
    ->     UNTIL @x > p1 END REPEAT;
    -> END
    -> //

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
```

Lệnh lặp **WHILE**:

- Lệnh lặp này khác lệnh lặp **REPEAT** ở 2 điểm sau:
  1) Điều kiện được kiểm tra trước khi thực hiện thân vòng lặp
  2) Thân vòng lặp sẽ được thực hiện trong khi điều kiện lặp **ĐÚNG** (trong khi vòng lặp **REPEAT** thì thực hiện thân vòng lặp khi điều kiện trong mệnh đề **UNTIL** sai).

Cú pháp:

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

Lệnh **RETURN**:

- Lệnh này được sử dụng trong hàm, dùng để trả về giá trị cho lời gọi hàm.

Cú pháp: RETURN *expr*

Ví dụ: Hàm sau đây nhận vào 1 chuỗi là họ và tên và trả về phần tên:

```
CREATE FUNCTION firstName(fullName VARCHAR(100))
RETURNS VARCHAR(10)
BEGIN
  DECLARE temp VARCHAR(100);
  DECLARE namePosition INT;
  SET temp = TRIM(fullName);
  SET namePosition = LOCATE(' ', REVERSE(temp));

  RETURN RIGHT(temp, namePosition - 1);
END
```

1.2 Hàm (Function)

Hàm là một đối tượng trong hệ quản trị CSDL, tương tự như thủ tục.

Điểm khác biệt giữa hàm và thủ tục là hàm trả về một giá trị. Giá trị trả về có thể là một bảng có được từ một câu truy vấn.

Hàm hệ thống: System Function.

Hàm người dùng: Do người dùng tạo ra gồm 3 dạng:

✧ Scalar_valued Function

Giá trị trả về là kiểu dữ liệu cơ sở (int, varchar, float, datetime…)

✧ Table_valued Function:

Giá trị trả về là một Table có được từ một câu truy vấn

✧ Aggregate Function:

Giá trị trả về là một bảng mà dữ liệu có được nhờ tích lũy dần sau một chuỗi thao tác xử lý và insert

Tạo hàm:

```
CREATE FUNCTION fn_name ([func_parameter[,...]])
    RETURNS type
    routine_body


func_parameter:
    param_name type


type:
 Any MySQL data types, except resultset
```

Lưu ý:

- Mỗi lệnh trong thân hàm, thủ tục kết thúc bằng dấu ";"
- Khi tạo hàm/thủ tục từ dòng lệnh, do mỗi lệnh SQL kết thúc bằng dấu ";", trùng với ký hiệu kết thúc 1 lệnh trong chế độ dòng lệnh nên ta cần phải thay đổi ký hiệu kết thúc lệnh của chế độ dòng lệnh để tránh trùng nhau giữa hai ký hiệu này bằng lệnh:

DELIMITER "chuỗi/ký tự kết thúc lệnh"

Trong MySQL còn có 1 loại biến được gọi là user-defined variable. Đây có thể được xem như là 1 biến toàn cục của phiên làm việc (session). Biến này tồn tại từ khi khai báo cho đến hết phiên làm việc và có thể truy xuất mọi nơi trong phiên làm việc đó. Một user-defined variable được thêm dấu "@" vào trước tên biến.

```
CREATE PROCEDURE prc_test()
BEGIN
    DECLARE var2 INT DEFAULT 1;
    SET var2 := var2 + 1;
    SET @var2 := @var2 + 1;
    SELECT var2, @var2;
END;
```

```
SET @var2 = 1;
CAL prc_test();@var2
L
var2
---  ---
2  3
```

```
CALL prc_test();@var2
var2  ---
---

2  3
```

❖ **Nhãn (label):**

- Nhãn dùng để đánh dấu (bookmark) 1 vị trí trong subroutine. Nhãn thường được sử dụng kết hợp với các lệnh điều khiển (lặp, rẽ nhánh). Một nhãn bao gồm 1 tên, theo sau bởi dấu hai chấm ":".
- Một nhãn có thể có nhãn kết thúc tương ứng. Nhãn kết thúc của một nhãn có tên trùng với tên nhãn nhưng không có dấu hai chấm theo sau. Do nhãn thường được sử dụng với các lệnh điều khiển nên ví dụ về nhãn sẽ được giới thiệu trong phần các câu lệnh điều khiển.

1.3 Con trỏ (Cursor)

Là một cấu trúc dữ liệu, ánh xạ đến một danh sách gồm các dòng dữ liệu từ một kết quả truy vấn (SELECT), cho phép duyệt tuần tự các dòng dữ liệu và đọc giá trị từng dòng trong danh sách kết quả.

Định nghĩa Con trỏ

DECLARE <Tên Con trỏ> CURSOR

FOR <Câu lệnh Truy vấn SELECT>

Con trỏ là cấu trúc toàn cục, duyệt theo một chiều từ đầu đế cuối, nội dung của Con trỏ có thể thay đổi.

Kiểm tra tình trạng Con trỏ:

Biến hệ thống @@FETCH_STATUS

Cho biết lệnh fetch vừa thực hiện có thành công hay không. Là cơ sở để biết đã duyệt đến cuối danh sách hay chưa

Nếu @@FETCH_STATUS =0 thì thành công, Con trỏ đang ở vị trí dòng thỏa mãn điều kiện trong kết quả truy vấn.

Nếu @@FETCH_STATUS <>0 thì KHÔNG thành công, Con trỏ đang ở vị trí vượt qua dòng cuối cùng kết quả truy vấn.

Các bước sử dụng Con trỏ trong lập trình

B1: Định nghĩa CURSOR từ một kết quả SELECT

DECLARE <tên Con trỏ> CURSOR FOR <Câu lệnh SELECT>

B2: Mở Cursor:

OPEN <Ten Con trỏ> , Con trỏ tham chiếu đến dòng 0

B3: Truy cập đến các bản ghi

FETCH NEXT FROM <Cursor_name> INTO <ds biến>

B4: Kểm tra có thành công không:

Nếu @FETCH_STATUS = 0 thì xử lý lệnh, quay lại B3

Nếu @FETCH_STATUS <> 0 thì sang B5

B5: Đóng Cursor:

CLOSE <Cursor_name>

B6: Xoá tham chiếu của Cursor:

DEALLOCATE <Cursor_name>

Sử dụng Con trỏ có thể đến vị trí một dòng nhất định trong tập kết quả.

Ví dụ: Thủ tục sau đây trả về danh sách các email của các nhân viên:

```
CREATE PROCEDURE build_email_list (INOUT email_list varchar(4000))
BEGIN
  DECLARE v_finished INTEGER DEFAULT 0;
  DECLARE v_email varchar(100) DEFAULT "";

  -- khai báo con trỏ cho employee email
  DEClARE email_cursor CURSOR FOR SELECT email FROM employees;

  -- khai báo NOT FOUND handler để xử lý lỗi không còn dữ liệu
  DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET v_finished = 1;
  OPEN email_cursor;

  get_email: LOOP
    FETCH email_cursor INTO v_email;

    IF v_finished = 1 THEN
      LEAVE get_email;
    END IF;

    -- build email list
    SET email_list = CONCAT(v_email,";",email_list);

  END LOOP get_email;

  CLOSE email_cursor;
END
```

II. TRANSACTION VÀ TRIGGER

2.1 Tạo một giao dịch

Transaction (giao tác hay giao dịch) là một tập hợp có thứ tự các thao tác và chúng chỉ có thể cùng nhau thành công hoặc cùng nhau thất bại. Một giao dịch sẽ thỏa các tính chất ACID, giúp đảm bảo tính toàn vẹn và an toàn của dữ liệu.

Một số lệnh xử lý giao dịch trong MySQL:

- Bắt đầu một giao dịch: **START TRANSACTION**
- Kết thúc và xác nhận giao dịch: **COMMIT**
- Hủy một giao dịch: **ROLLBACK [TO <savepoint_name>]**
- Tạo 1 điểm mốc: **SAVEPOINT <savepoint_name>**

Một số lệnh không được sử dụng trong giao dịch:

- **CREATE /ALTER / DROP / RENAME / TRUNCATE TABLE**
- **CREATE / DROP INDEX**
- **CREATE / DROP EVENT**
- **CREATE / DROP FUNCTION**
- **CREATE / DROP PROCEDURE**

Một ví dụ về giao dịch như sau: Giả sử một công ty bán hàng có một đặt hàng mới và cần ghi nhận đặt hàng vào CSDL. Các bước cần thiết để thực hiện tác vụ này bao gồm:

1) Truy vấn **Số đơn đặt hàng** gần nhất từ table **Orders** để tính được Số đơn đặt hàng cho đơn đặt hàng mới.
2) Chèn các thông tin chung của đơn đặt hàng mới vào table **Orders**.
3) Chèn các chi tiết của đơn đặt hàng mới vào table **OrderDetails**.
4) Lấy các thông tin đặt hàng của đơn đặt hàng mới từ cả hai bảng **Orders** và **OrderDetails** để xác nhận giao dịch với khách hàng.

Để đảm bảo tính toàn vẹn của dữ liệu, các bước trên phải thỏa mãn các tính chất của một giao dịch.

Ví dụ, trong khi thực hiện bước 1 cho một đặt hàng, nếu một xử lý cho đơn đặt hàng khác lại thực hiện song song thì sẽ dẫn đến hai đơn đặt hàng có cùng số đơn đặt hàng. Hoặc nếu một giao dịch đang thực hiện đến bước 3 chưa hoàn thành thì bị lỗi có thể dẫn đến thông tin chung của đơn đặt hàng đã được cập nhật vào CSDL nhưng chi tiết của đơn đặt hàng lại không có,…

Để tạo 1 giao dịch trong MySQL cho tác vụ đặt hàng nói trên, ta thực hiện các bước sau:

a) Tạo 1 giao dịch mới với lệnh BEGIN TRANSACTION
b) Dùng các lệnh SQL để thực hiện các tác vụ trong bước 1, 2 và 3

c) Xác nhận giao dịch bằng lệnh COMMIT

d) Truy vấn thông tin của đơn đặt hàng mới để xác nhận với người mua

Scrip SQL sau sẽ minh họa cho các bước trên:

```
-- start a new transaction
start transaction;

-- get latest order number
select @orderNumber := max(orderNUmber)
from orders;
-- set new order number
set @orderNumber = @orderNumber  + 1;

-- insert a new order for customer 145
insert into orders(orderNumber,
                   orderDate,
                   requiredDate,
                   shippedDate,
                   status,
                   customerNumber)
values(@orderNumber,
       now(),
       date_add(now(), INTERVAL 5 DAY),
       date_add(now(), INTERVAL 2 DAY),
       'In Process',
        145);
-- insert 2 order line items
insert into orderdetails(orderNumber,
                         productCode,
                         quantityOrdered,
                         priceEach,
                         orderLineNumber)
values(@orderNumber,'S18_1749', 30, '136', 1),
      (@orderNumber,'S18_2248', 50, '55.09', 2);
-- commit changes
commit;

-- get the new inserted order
select * from orders a
inner join orderdetails b on a.ordernumber = b.ordernumber
where a.ordernumber = @ordernumber
```

Các mức độ cô lập được hỗ trợ bởi MySQL:

1) READ UNCOMMITTED: cho phép dirty read.
2) READ COMMITED: không cho phép dirty read nhưng cho phép nonrepeatable read và phantom.
3) REPEATABLE READ: không cho phép dirty read và nonrepeatable read nhưng có thể có phantom.
4) SERIALIZABLE: thực thi các giao dịch theo chế độ tuần tự, các ngoại lệ và bất thường trên sẽ không xảy ra.

Để thiết đặt mức độ cô lập, ta dùng lệnh sau:

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
< REPEATABLE READ | READ COMMITTED |
  READ UNCOMMITTED | SERIALIZABLE >
```

2.2 Trigger

Trigger là một loại stored Procedure không có tham số, có đặc điểm sau:

- Tự động thực hiện khi có lệnh **INSERT, DELETE** hoặc **UPDATE** trên DL
- Thường dùng để kiểm tra các ràng buộc toàn vẹn của CSDL
- Một TRIGGER được định nghĩa trên một bảng, nhưng các xử lý trong TRIGGER có thể sử dụng nhiều bảng khác.

Lưu ý: Sử dụng từ khóa OLD, NEW để lấy record tương ứng.

Cú pháp:

CREATE TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name **FOR EACH ROW**
    [trigger_order]
    trigger_body

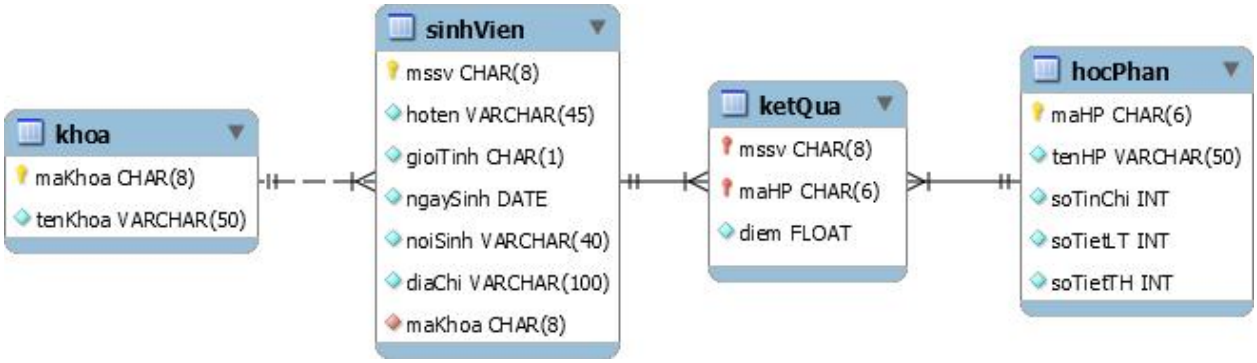
trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLWS | PRECEDES } other_trigger_name

III. THỰC HÀNH

Bài 1: Tạo các thủ tục và hàm sau (dựa trên **CSDL QLDIEM**) ở thực hành 1:



1. Thủ tục **THEM_SV** cho phép thêm sinh viên mới vào CSDL với các tham số: MSSV, họ tên, giới tính (nam/nữ), ngày sinh, nơi sinh, địa chỉ, tên khoa.
2. Thủ tục **XOA_SV** cho phép xóa sinh viên trong CSDL. Thủ tục này nhận vào 1 MSSV, nếu MSSV tồn tại thì xóa các kết quả của sinh viên đó trước khi xóa SV ra khỏi CSDL.
3. Thủ tục **DIEM_TB**, nhận vào MSSV và trả về điểm trung bình của sinh SV có MSSV tương ứng thông qua output parameter. Nếu MSSV truyền vào không có thì trả về -1.
4. Thủ tục **BANG_DIEM_TB** nhận vào một mã khoa và trả về bảng điểm trung bình của các SV thuộc khoa đó.
5. Viết hàm **TOT_NGHIEP** kiểm tra một sinh viên có đủ điều kiện tốt nghiệp hay không. Hàm nhận vào một MSSV và trả về TRUE (nonzero) nếu sinh viên đủ điều kiện tốt nghiệp và FALSE (zero) nếu không đủ điều kiện.
6. Viết hàm **LOAI_TOT_NGHIEP** nhận vào một MSSV và trả về loại tốt nghiệp của sinh viên (dựa vào qui chế học vụ của Trường ĐHCT).
7. Viết thủ tục **SV_TOT_NGHIEP** nhận vào mã khoa và liệt kê các sinh viên đủ điều kiện tốt nghiệp của khoa đó, có sử dụng hàm đã viết trong Câu 6) và 7) và con trỏ để xử lý. Các thông tin cần hiển thị là MSSV, họ tên, số tín chỉ đã học, điểm trung bình, loại tốt nghiệp.

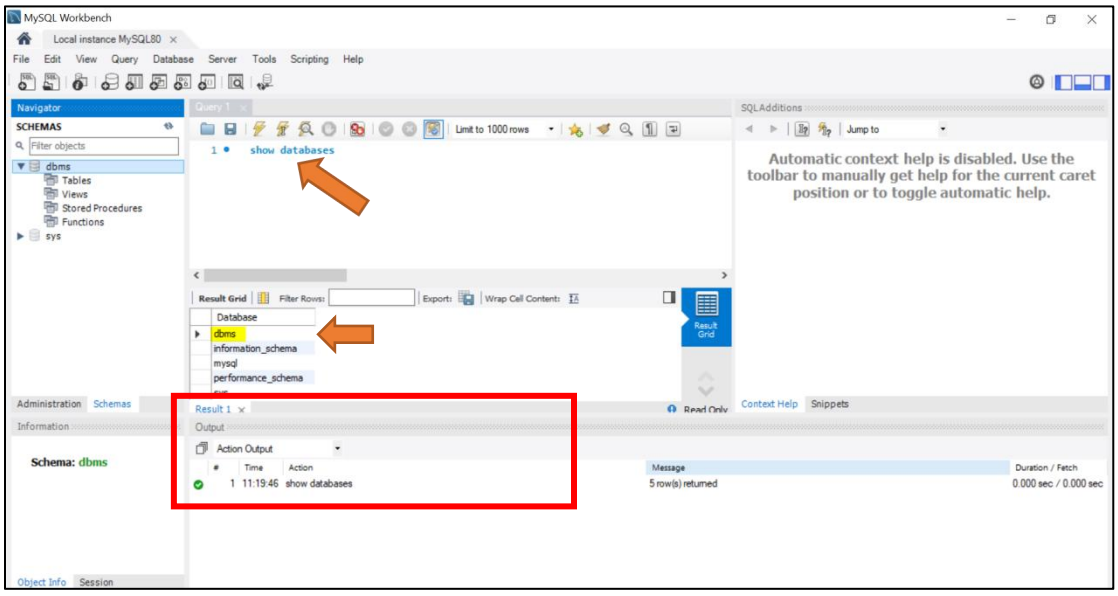
8. Viết hàm **SL_SV_KHOA** nhận vào tên Khoa và trả về số lượng SV của Khoa đó.
9. Viết thủ tục **SV_LOAI** nhận vào một tên loại (giỏi, khá, trung bình,…) và trả về danh sách sinh viên (bao gồm MSSV, họ tên) có loại tương ứng cho lời gọi thủ tục (sử dụng hàm **LOAI_TOT_NGHIEP** bên trên).
10. Tạo một **Trigger** kiểm tra điều kiện cho cột DIEM là <= 10

-----Hết Buổi 2-----

HƯỚNG DẪN CÁCH TRẢ LỜI VÀ NỘP FILE BÀI LÀM

- Các bạn làm thực hành từ phần **III. THỰC HÀNH**
- Mỗi câu chụp màn hình phải có câu lệnh truy vấn và kết quả hiện thị bên dưới (10 câu tương đương có 10 hình)

Ví dụ: Thực hiện lệnh show databases. Như **Hình 1** nhìn thấy được câu truy vấn và kết quả sau khi chạy lệnh show databases



Hình 1

- Nộp file có phần mở rộng **.pdf** và lưu với tên: <MSSV>_<Họ và tên>_TH2.pdf

Ví dụ: Sinh viên có MSSV là B1234567 tên Nguyễn Văn An, thì lưu:

B1234567_NguyenVanAn_TH2.pdf