

Εργαστήριο 5

Μεταβλητές, Συναρτήσεις και Διαδικασίες

Ανάπτυξη και Επιμέλεια: Ζ. Γαροφαλάκη, Α. Τσολακίδης, Π. Ανδρίτσος

Στόχος

Εξοικείωση με τον ορισμό και τη χρήση μεταβλητών (variables), τη δημιουργία συναρτήσεων (functions) και διαδικασιών (procedures)

Εργαλεία

A. Μεταβλητές

1. Η **δημιουργία μεταβλητής** γίνεται με την ακόλουθη γενική μορφή εντολής. Η κάθε μεταβλητή καθώς και το περιεχόμενό της διατηρούνται για την συγκεκριμένη σύνοδο (session) του χρήστη.

```
SET @var_name = expr [, @var_name = expr] ...
```

2. Η **δημιουργία μεταβλητών** με: όνομα alpha και τιμή 10, όνομα beta και τιμή 20, γίνεται με την εντολή:

```
set @alpha=10, @beta=20;
```

3. Η εμφάνιση των περιεχομένων μιας μεταβλητής γίνεται με την εντολή:

```
select @alpha;           //εμφάνιση τιμής της μεταβλητής @alpha  
  
select @beta;            //εμφάνιση τιμής της μεταβλητής @beta  
  
select @chi:=@alpha+@beta; //εμφάνιση τιμής της μεταβλητής @chi που είναι το άθροισμα των  
                           //alpha και beta
```

Δραστηριότητες

Υλοποιήστε τις ακόλουθες δραστηριότητες. Η εντολή ή οι εντολές που απαιτούνται για την υλοποίηση του κάθε βήματος, καθώς και το αποτέλεσμα της εκτέλεσής της/τους θα πρέπει να ενταχθεί/-ούν σε ένα παραδοτέο αρχείο με τη μορφή κειμένου ή με τη μορφή στιγμιότυπου (screenshot). Το αρχείο ή τα αρχεία με τις απαντήσεις σας, θα πρέπει να συμπεριστούν σε ένα **xx_ZZZZZ_ΕΡΩΝΥΜΟ.zip**, όπου: (α) xx ο αριθμός του τμήματος στο οποίο ανήκετε (π.χ. για την ομάδα [02] ΔΕΥΤΕΡΑ 12:00-13:00, **xx = 02**) και (β) ZZZZZ ο Αριθμός Μητρώου σας. Το τελικό αυτό αρχείο θα υποβάλλεται στο χώρο του e-class -> ΒΑΣΕΙΣ ΔΕΔΟΜΕΝΩΝ II -> Εργασίες.

1. **Συνδεθείτε** στην MySQL του συστήματός σας με όποιον από τους προαναφερόμενους τρόπους επιθυμείτε.
2. Ελέγξτε αν υπάρχει ΒΔ με την ονομασία **my_accounts**.
3. Δημιουργήστε τη ΒΔ my_accounts, επιλέξτε την για χρήση και δημιουργήστε και πίνακα με όνομα Accounts με δομή και περιεχόμενα, όπως φαίνονται στις ακόλουθες εντολές. Δείξτε το αποτέλεσμα εμφανίζοντας (α) τη λίστα πινάκων της ΒΔ, (β) τα περιεχόμενα και (γ) τη δομή του πίνακα Accounts.

```
drop database if exists my_accounts;
create database my_accounts;
use my_accounts;
create table Accounts ( acctID integer not null primary key,
Balance integer not null);
insert into Accounts (acctID, Balance) values (101, 1000);
insert into Accounts (acctID, Balance) values (202, 2000);
insert into Accounts (acctID, Balance) values (303, 2500);
insert into Accounts (acctID, Balance) values (404, 3000);
```

4. Εμφανίστε τα περιεχόμενα του πίνακα Accounts, προσθέτοντας αύξουσα αρίθμηση στις εγγραφές του:

```
set @rownum=0;
select (@rownum:=@rownum+1) as No, acctID, Balance from Accounts order by acctID;
```

5. Η αύξουσα αρίθμηση που εμφανίστηκε στην στήλη με τίτλο No του βήματος 4, θα πρέπει να υπάρχει και στον πίνακα Accounts; Αιτιολογήστε την απάντησή σας.
6. Προσθέστε τον πίνακα CUSTOMERS που εμφανίζεται στην [Εικόνα 1](#). Ορίστε τύπους δεδομένων των στηλών CUSTNO και CUST_NAME ως integer και varchar(30) αντίστοιχα. Δείξτε το αποτέλεσμα εμφανίζοντας (α) τη λίστα πινάκων της ΒΔ, (β) τα περιεχόμενα και (γ) τη δομή του πίνακα CUSTOMERS.
7. Στον πίνακα Accounts προσθέστε στήλη με όνομα Custno, τύπο δεδομένων integer και ορίστε την ως FK του πίνακα Accounts για τη σύνδεση των εγγραφών του με τις εγγραφές του πίνακα CUSTOMERS. Ενημερώστε τα περιεχόμενα της στήλης Custno, ώστε ο λογαριασμός με AcctID=202 να αντιστοιχεί στον κωδικό πελάτη 20 και όλοι οι υπόλοιποι λογαριασμοί να αντιστοιχούν στον κωδικό πελάτη 10. Δείξτε το αποτέλεσμα εμφανίζοντας (α) τα περιεχόμενα και (β) τη δομή του πίνακα Accounts.

8. Εκτελέστε και ερμηνεύστε τις ακόλουθες δηλώσεις SQL:

```
select CUSTNO, count(*), sum(Balance)
from Accounts
where CUSTNO not in (20)
group by CUSTNO;
```

```
//Παραλλαγή με χρήση μεταβλητής
set @CUST_NO=20;
```

```
select CUSTNO, count(*), sum(Balance)
from Accounts
where CUSTNO not in (@CUST_NO)
group by CUSTNO;
```

9. Εκτελέστε και ερμηνεύστε τις ακόλουθες δηλώσεις SQL:

```
select count(*), sum(Balance) from Accounts;
```

```
//Παραλλαγή με χρήση μεταβλητών
set @COUNT_acctID=0, @SUM_acctID=0, @AVG_acctID=0;
```

```
select count(*), sum(Balance), avg(Balance)
into @COUNT_acctID, @SUM_acctID, @AVG_acctID
from Accounts;
```

```
select @COUNT_acctID, @SUM_acctID, @AVG_acctID, @MY_AVG := @SUM_acctID/@COUNT_acctID;
```

- 10.** Στον πίνακα Accounts προσθέστε στήλη με όνομα Amount, τύπο δεδομένων integer και ορίστε την ως FK του πίνακα Accounts για τη σύνδεση των εγγραφών του με τις εγγραφές του πίνακα CUSTOMERS. Ενημερώστε τα περιεχόμενα της στήλης Custno, ώστε ο λογαριασμός με AcctID=202 να αντιστοιχεί στον κωδικό πελάτη 20 και όλοι οι υπόλοιποι λογαριασμοί να αντιστοιχούν στον κωδικό πελάτη 10. Δείξτε το αποτέλεσμα εμφανίζοντας (α) τα περιεχόμενα και (β) τη δομή του πίνακα Accounts.

11. Εκτελέστε και ερμηνεύστε τις ακόλουθες δηλώσεις SQL:

```
create trigger CALC_SUM
before insert on Accounts
for each row
set @SUM = @SUM + new.Amount;
```

```
// Δοκιμή της λειτουργίας του trigger
set @SUM = 0;
insert into Accounts values(100,1000.50),(101, 2000.50),(102, 1500.00);
select @SUM as 'TOTAL AMOUNT';
select * from Accounts;
```

- 12.** Ορίστε και χρησιμοποιήστε τη συνάρτηση factorial που υπολογίζει το $n! = 1 * 2 * \dots * n$:

```
drop function if exists factorial;
delimiter !
create function factorial(N int)
returns int
deterministic
begin
declare F int default 1;
while N > 0 do
set F = N * F;
set N = N - 1;
end while;
return F;
end !
delimiter ;
```

```
// Δοκιμές της λειτουργίας του trigger για N=4 και N=15
select factorial(4);

select factorial(15);
```

- 13.** Ορίστε και χρησιμοποιήστε την διαδικασία `my_procedure_Local_Variables` για υπολογισμούς με χρήση τοπικών μεταβλητών:

```
drop procedure my_procedure_Local_Variables;
delimiter $$
create procedure my_procedure_Local_Variables()
begin
set @X = 25;
set @Y = 10;
select @X, @Y, @X*@Y;
end $$
delimiter ;
// Κλήση της διαδικασίας
call my_procedure_Local_Variables();
```

- 14.** Ακολουθήστε τα ακόλουθα για τη δημιουργία μιας αποθηκευμένης διαδικασίας και χρήση των `commit/rollback`. Εξηγείστε τι κάνει η διαδικασία `myProc`.

```
// Δοκιμές χρήσης της συνάρτησης MOD
SET @p_no=3;
SELECT MOD(@p_no,2);
SET @p_no=8;
SELECT MOD(@p_no,2);

// Δημιουργία βάσης και πίνακα
DROP TABLE IF EXISTS myTrace;
CREATE TABLE myTrace ( t_no INT,
t_user CHAR(20),
t_date DATE,
t_time TIME,
t_proc VARCHAR(16), t_what VARCHAR(30));

// Δημιουργία αποθηκευμένης διαδικασίας myProc
DROP PROCEDURE IF EXISTS myProc;
DELIMITER !
CREATE PROCEDURE myProc (IN p_no INT, IN p_in VARCHAR(30),
OUT p_out VARCHAR(30))
LANGUAGE SQL
BEGIN
SET p_out=p_in;
INSERT INTO myTrace (t_no, t_user, t_date, t_time, t_proc, t_what)
VALUES (p_no, current_user, current_date, current_time, 'myProc', p_in);
IF (MOD(p_no,2)=0) THEN
COMMIT;
ELSE ROLLBACK;
END IF;
END !
DELIMITER ;

// Κλήση της διαδικασίας
SET AUTOCOMMIT=0;
CALL myProc(1, 'hello1', @out);
CALL myProc(2, 'hello2', @out);
CALL myProc(3, 'hello3', @out);
CALL myProc(4, 'hello4', @out);
CALL myProc(5, 'hello5', @out);
CALL myProc(6, 'hello6', @out);
CALL myProc(7, 'hello7', @out);
```

```
// Μελέτη αποτελέσματος  
SELECT * FROM myTrace;
```

15. Στον πίνακα Accounts (Εικόνα 1) η μεταφορά χρημάτων από ένα λογαριασμό σε έναν άλλο θα μπορούσε να υλοποιηθεί με δύο δηλώσεις UPDATE. Ακολουθεί παράδειγμα επίλυσης με χρήση συναλλαγής (transaction). Η συναλλαγή αυτή χαρακτηρίζεται ως αναξιόπιστη, καθώς δεν γίνεται έλεγχος σχετικά: (α) με την ύπαρξη του λογαριασμού στον οποίο μεταφέρονται τα χρήματα και (β) την επάρκεια του λογαριασμού από τον οποίο μεταφέρονται τα χρήματα.

```
// Δημιουργία πίνακα Accounts με 2 εγγραφές  
DROP TABLE IF EXISTS Accounts;  
CREATE TABLE Accounts ( acctID INTEGER NOT NULL PRIMARY KEY,  
    balance INTEGER NOT NULL,  
    CONSTRAINT unloanable_account CHECK (balance >= 0));  
INSERT INTO Accounts (acctID, balance) VALUES (101, 1000);  
INSERT INTO Accounts (acctID, balance) VALUES (202, 2000);  
COMMIT;  
SELECT * FROM accounts;  
  
// Συναλλαγή  
BEGIN TRANSACTION;  
UPDATE Accounts SET balance = balance - 100  
WHERE acctID = 101;  
UPDATE Accounts SET balance = balance + 100  
WHERE acctID = 202;  
COMMIT;  
  
// Αποτέλεσμα συναλλαγής  
SELECT * FROM accounts;
```

16. Ακολουθεί λύση του προβλήματος στο βήμα 15 με χρήση της procedure BankTransfer:

```
// Δημιουργία procedure BankTransfer  
DELIMITER //  
DROP PROCEDURE BankTransfer //  
CREATE PROCEDURE BankTransfer (IN fromAcct INT,  
    IN toAcct INT,  
    IN amount INT,  
    OUT msg VARCHAR(100)  
)  
  
P1: BEGIN  
    DECLARE rows INT ;  
    DECLARE newbalance INT;  
    SELECT COUNT(*) INTO rows FROM Accounts WHERE acctID = fromAcct;  
    UPDATE Accounts SET balance = balance - amount WHERE acctID = fromAcct;  
    SELECT balance INTO newbalance FROM Accounts WHERE acctID = fromAcct;  
    IF rows = 0 THEN  
        ROLLBACK;  
        SET msg = CONCAT('rolled back because of missing account ', fromAcct);  
    ELSEIF newbalance < 0 THEN  
        ROLLBACK;  
        SET msg = CONCAT('rolled back because of negative balance of account ', fromAcct);  
    ELSE  
        SELECT COUNT(*) INTO rows FROM Accounts WHERE acctID = toAcct;  
        UPDATE Accounts SET balance = balance + amount WHERE acctID = toAcct;  
        IF rows = 0 THEN  
            ROLLBACK;  
            SET msg = CONCAT('rolled back because of missing account ', toAcct);  
        ELSE  
            COMMIT;  
            SET msg = 'committed';  
        END IF;  
    END IF;  
END P1 //  
DELIMITER ;
```

```
// Δοκιμή μεταφοράς 100 από acctID=101 σε acctID=202
SET AUTOCOMMIT=0;
SET @out = ' ';
CALL BankTransfer (101, 202, 100, @out);
SELECT @OUT;
Select * from accounts;
COMMIT;

// Δοκιμή μεταφοράς 100 από acctID=101 σε acctID=201 (ανύπαρκτος)
SET autocommit=0;
SET @out = ' ';
CALL BankTransfer (100, 201, 100, @out);
SELECT @OUT;
Select * from accounts;
COMMIT;

// Δοκιμή μεταφοράς 100 από acctID=100 (ανύπαρκτος) σε acctID=201
SET autocommit=0;
SET @out = ' ';
CALL BankTransfer (100, 201, 100, @out);
SELECT @OUT;

// Δοκιμή μεταφοράς 1500 από acctID=101 (ανεπαρκής) σε acctID=201
SET AUTOCOMMIT=0;
SET @out = ' ';
CALL BankTransfer (101, 201, 1500, @out);
SELECT @OUT;
Select * from accounts;
COMMIT;
```

17. Ακολουθεί δεύτερη λύση του προβλήματος στο βήμα 15:

```
// Δημιουργία πίνακα Accounts
DROP TABLE IF EXISTS Accounts;
CREATE TABLE Accounts ( acctID INTEGER NOT NULL PRIMARY KEY,
                        balance INTEGER NOT NULL);
INSERT INTO Accounts (acctID, balance) VALUES (101, 1000);
INSERT INTO Accounts (acctID, balance) VALUES (202, 2000);
COMMIT;
SELECT * FROM accounts;

// Δημιουργία trigger Accounts_upd_trg για έλεγχο των updates
delimiter !
CREATE TRIGGER Accounts_upd_trg
BEFORE UPDATE ON Accounts
FOR EACH ROW
BEGIN
IF NEW.balance < 0 THEN
SIGNAL SQLSTATE '23513'
SET MESSAGE_TEXT = 'Negative balance not allowed';
END IF;
END; !
delimiter ;

// Δημιουργία trigger Accounts_ins_trg για έλεγχο των inserts
delimiter !
CREATE TRIGGER Accounts_ins_trg
BEFORE INSERT ON Accounts
FOR EACH ROW
BEGIN
IF NEW.balance < 0 THEN
SIGNAL SQLSTATE '23513'
SET MESSAGE_TEXT = 'Negative balance not allowed';
END IF;
END; !
delimiter ;
```

```
// Δημιουργία procedure BankTransfer
DELIMITER !
CREATE PROCEDURE BankTransfer (IN fromAcct INT,
IN toAcct INT,
IN amount INT,
OUT msg VARCHAR(100))
LANGUAGE SQL MODIFIES SQL DATA
P1: BEGIN
DECLARE acct INT;
DECLARE balance_v INT;
DECLARE EXIT HANDLER FOR NOT FOUND
BEGIN ROLLBACK;
SET msg = CONCAT('missing account ', CAST(acct AS CHAR));
END;
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN ROLLBACK;
SET msg = CONCAT('negative balance (?) in ', fromAcct);
END;
SET acct = fromAcct;
SELECT acctID INTO acct FROM Accounts WHERE acctID = fromAcct ;
UPDATE Accounts SET balance = balance - amount
WHERE acctID = fromAcct;
SET acct = toAcct;
SELECT acctID INTO acct FROM Accounts WHERE acctID = toAcct ;
UPDATE Accounts SET balance = balance + amount
WHERE acctID = toAcct;
SELECT balance INTO balance_v
FROM accounts
WHERE acctID = fromAcct;
IF balance_v < 0 THEN
ROLLBACK;
SET msg = CONCAT(' negative balance in ', fromAcct);
ELSE
COMMIT;
SET msg = 'committed';
END IF;
END P1 !
DELIMITER ;

CALL BankTransfer (101, 201, 100, @msg);
Select @msg;
CALL BankTransfer (100, 202, 100, @msg);
Select @msg;
CALL BankTransfer (101, 202, 100, @msg);
Select @msg;
CALL BankTransfer (101, 202, 2000, @msg);
Select @msg;
```

ΒΔ personnel

Οι περιεχόμενοι πίνακες της ΒΔ personnel θα πρέπει να έχουν την ακόλουθη δομή και περιεχόμενα:

Στήλες	Τύπος δεδομένων
DEPT.DEPTNO, EMP.DEPTNO	numeric(2)
DNAME, JOB_DESCR	varchar(24)
LOC	char(23)
JOBCODE, JOBNO	numeric(3)
SAL, COMM	numeric(10,2)
EMPNO	numeric(4)
PROJECT.P_ID	int
PROJECT.P_NAME	varchar(255)

Πίνακας 1. Τύποι δεδομένων πινάκων EMP, JOB, DEPT

EMP

EMPNO	NAME	JOBNO	DEPTNO	COMM
10	CODD	100	50	
20	NAVATHE	200	50	450
30	ELMASRI	300	60	
40	DATE	100	50	

JOB

JOBCODE	JOB_DESCR	SAL
100	SALESMAN	2000
200	ANALYST	2000
300	DBA	3000

DEPT

DEPTNO	DNAME	LOC
50	SALES	ATHENS
60	ACCOUNTING	ATHENS
70	PAYROL	VOLOS

Accounts

acctID	Balance
101	1000
202	2000
303	2500
404	3000

CUSTOMERS

CUSTNO	CUST_NAME
10	101
20	202

Εικόνα 1. Δεδομένα πινάκων

Παράρτημα

Στο Παράρτημα περιλαμβάνονται εξειδικευμένα παραδείγματα χρήσης των triggers, functions και procedures. Ακολουθείστε τα για εμβάθυνση στις ανάλογες έννοιες. Οι διαδικασίες που θα ακολουθήσετε και τα αποτελέσματά τους **ΔΕΝ αποτελούν ζητούμενο προς υποβολή** για την εργαστηριακή άσκηση.

1. Δημιουργήστε trigger που συνδέεται με πίνακα λογαριασμού και προσθέτει τις τιμές που εισάγονται (δηλώσεις INSERT) σαν ποσά σε λογαριασμούς:

```
// Δημιουργία db my_accounts με πίνακα accounts
DROP DATABASE if exists my_accounts;
CREATE DATABASE my_accounts;
USE my_accounts;
DROP TABLE IF EXISTS accounts;
CREATE TABLE accounts (acct_num INT, amount DECIMAL(12,2));
CREATE TABLE total_bal(total_sum INT);
Insert into total_bal VALUES(0);

// Δημιουργία trigger calc_sum για έλεγχο των inserts
DROP TRIGGER IF EXISTS calc_sum;
DELIMITER #
CREATE TRIGGER calc_sum
BEFORE INSERT ON accounts
FOR EACH ROW
BEGIN
    DECLARE new_sum INT;
    SELECT total_sum
    INTO new_sum
    FROM total_bal;
    SET new_sum=new_sum+ NEW.amount;
    Update total_bal
    SET total_sum= new_sum;
END;
#
DELIMITER ;

// Δοκιμές
INSERT INTO accounts VALUES(100,1000.50),(101, 2000.50),(102, 1500.00);
SELECT * FROM accounts;
SELECT * FROM total_bal;
INSERT INTO accounts VALUES(103,1000.50),(104, 2000.50),(105, 1500.00);
SELECT * FROM accounts;
SELECT * FROM total_bal;
```

2. Εναλλακτική λύση:

```
// Δημιουργία db my_accounts με πίνακα accounts
DROP DATABASE if exists my_accounts;
CREATE DATABASE my_accounts;
USE my_accounts;
DROP TABLE IF EXISTS accounts;
CREATE TABLE accounts (acct_num INT, amount DECIMAL(12,2));
CREATE TABLE total_bal(total_sum INT);
Insert into total_bal VALUES(0);

// Δημιουργία trigger calc_sum για έλεγχο των inserts
DROP TRIGGER IF EXISTS calc_sum;
DELIMITER #
CREATE TRIGGER calc_sum
BEFORE INSERT ON accounts
FOR EACH ROW
BEGIN
    Update total_bal
```

```

    SET total_sum= total_sum + NEW.amount;
END;
#
DELIMITER ;

// Δοκιμές
INSERT INTO accounts VALUES(100,1000.50),(101, 2000.50),(102, 1500.00);
SELECT * FROM accounts;
SELECT * FROM total_bal;
INSERT INTO accounts VALUES(103,1000.50),(104, 2000.50),(105, 1500.00);
SELECT * FROM accounts;
SELECT * FROM total_bal;

```

3. Δημιουργήστε trigger delete_orders και procedure delete_orders_orderlines για τη διαγραφή παραγγελίας. Με την procedure add_order_line θα ελέγχεται η εισαγωγή γραμμών για παραγγελία που δεν υπάρχει:

```

// Δημιουργία db delete_orders με πίνακες orders και orderlines
drop database if exists delete_orders;
create database delete_orders;
use delete_orders;
drop table if exists orders;
create table orders (orderno int auto_increment not null, custno int not null, odate
datetime not null, primary key(orderno));
drop table if exists orderlines;
create table orderlines (orderno int not null, stockno int not null, qty int not null,
primary key (orderno,stockno));
DESCRIBE orders;
DESCRIBE orderlines;

SET AUTOCOMMIT=0;
insert into orders (custno,odate) values (1,current_timestamp);
insert into orderlines (orderno,stockno,qty) values (1,10,1),(1,50,2);
commit;
select * from orders;
select * from orderlines;

// Δημιουργία trigger delete_orders για έλεγχο των deletes
drop trigger IF EXISTS delete_orders;
CREATE TRIGGER delete_orders
AFTER DELETE ON orders
FOR EACH ROW
DELETE FROM orderlines WHERE orderno= OLD.orderno;
/* testing */
delete from orders where orderno=1;
select * from orders;
select * from orderlines;
rollback;
select * from orders;
select * from orderlines;

// Δημιουργία procedure delete_orders_orderlines για έλεγχο των deletes στον πίνακα
// orderlines
DROP PROCEDURE delete_orders_orderlines;
DELIMITER $
CREATE PROCEDURE delete_orders_orderlines(IN del_order INT)
BEGIN
    DELETE FROM orderlines WHERE orderno=del_order;
    DELETE FROM orders WHERE orderno=del_order;
END;
$
Delimiter ;

```

```
// Δοκιμές
Set autocommit=0;
select * from orders;
select * from orderlines;
CALL delete_orders_orderlines(1);
select * from orders;
select * from orderlines;
rollback;
select * from orders;
select * from orderlines;

// Δημιουργία procedure add_order_line για έλεγχο ύπαρξης παραγγελίας
DELIMITER $
CREATE PROCEDURE add_order_line(IN o_no int, IN s_no int, IN qty int)
BEGIN
DECLARE count_var INT;
SELECT COUNT(orderno)
INTO count_var
FROM orders
WHERE orderno=o_no;
IF (count_var <>1) THEN
SIGNAL SQLSTATE '45000'
SET MESSAGE_TEXT='Order number is not found in orders table';
END IF;
END;
$
DELIMITER ;

// Δοκιμές
SELECT * FROM Orders;
SELECT * FROM Orderlines;
CALL add_order_line(2,10,1);
ERROR 1644(45000): Order number is not found in orders table

CALL add_order_line(2,50,2);
ERROR 1644(45000): Order number is not found in orders table

SELECT * FROM Orders;
SELECT * FROM Orderlines;
```

4. Οι δηλώσεις commit και rollback δεν επιτρέπονται σε αποθηκευμένες συναρτήσεις. Ακολουθεί παράδειγμα:

```
// Δημιουργία function myFun με commit και rollback
DROP FUNCTION IF EXISTS myFun;
DELIMITER !
CREATE FUNCTION myFun (p_no INT, p_in VARCHAR(30))
RETURNS VARCHAR(30)
LANGUAGE SQL
BEGIN
INSERT INTO myTrace (t_no, t_user, t_date, t_time, t_proc, t_what)
VALUES (p_no, current_user, current_date, current_time, 'myProc', p_in);
IF (MOD(p_no,2)=0) THEN
COMMIT;
ELSE ROLLBACK;
END IF;
RETURN p_in;
END !
DELIMITER ;

Η συνάρτηση δεν δημιουργείται και επιστρέφει σφάλμα:
ERROR 1422(HY000): Explicit or implicit commit is not allowed in stored function or trigger
```

5. Παράδειγμα διαχείρισης διαίρεσης με 0:

```
// Δημιουργία procedure Divide_by_zero για έλεγχο ύπαρξης διαίρεσης με το 0
DELIMITER $$
CREATE PROCEDURE Divide_by_zero (IN numerator INT, IN denominator INT,
    OUT results INT)
BEGIN
    DECLARE div_by_zero CONDITION FOR SQLSTATE '22012';
    DECLARE CONTINUE HANDLER FOR div_by_zero RESIGNAL SET MESSAGE_TEXT = 'Division by zero';
    IF denominator = 0 THEN
        SIGNAL div_by_zero;
    ELSE
        SET results := numerator/denominator;
    END IF;
END;
$$
DELIMITER ;
// Δοκιμή
CALL Divide_by_zero(100, 0, @results);
ERROR 1644(22012): Division by zero
```

6. Παράδειγμα διαχείρισης διαίρεσης με 0:

```
// Δημιουργία db my_first_triggers_db με πίνακα new_dept
USE my_first_triggers_db;
drop table if exists new_dept;
create table new_dept(deptno int not null, dname char(30),
    PRIMARY KEY(deptno));

// Δημιουργία trigger trg_signals_raising_error για έλεγχο των εισαγωγών εγγραφών στον
// πίνακα new_dept με αρνητικό κωδικό τμήματος
drop trigger if exists trg_signals_raising_error;
delimiter //
create trigger trg_signals_raising_error
before insert on new_dept
for each row
begin
    declare msg varchar(255);
    if new.deptno < 0 then
        set msg = concat('Trying to insert a negative value in deptno: ', cast(new.deptno as char));
        signal sqlstate '45000' set message_text = msg;
    end if;
end
//
delimiter ;

// Δοκιμές
insert into new_dept values (1, 'SALES'), (-1, 'ACCOUNTING'), (2, 'RESEARCH');
Μήνυμα λάθους και καθολική απόρριψη εισαγωγής

insert into new_dept values (1, 'SALES');
select * from new_dept;

insert into new_dept values (-1, 'ACCOUNTING');
Μήνυμα λάθους και απόρριψη εισαγωγής
```