DATA BASES II

DEPARTMENT OF INFORMATION AND COMPUTER ENGINEERING

5th LABORATORY EXERCISE

VARIABLES, FUNCTIONS AND PROCEDURES

WORK DETAILS

LABORATORY DEPARTMENT: [06] WEDNESDAY 13:00-14:00
LABORATORY RESPONSIBILITY : GAROFALAKI RANIA
DELIVERY DATE : 02/07/2024
SUBMISSION DEADLINE : 02/13/2024

STUDENT DETAILS

STUDENT PHOTO:

NAME: ATHANASIOU VASILEIOS EVANGELOS
STUDENT ID: 19390005
STUDENT SEMESTER: 9th
STUDENT STATUS : UNDERGRADUATE
PROGRAM OF STUDY : UNIWA

CONTENTS

DB personnel **6**

personnel.sql 6

personnel.png 8

DB my_accounts **8**

my_accounts.sql 8

my _ accounts . png 9

Activities **10**

1. Connect to your system's .....

1.2. Snapshot 10

2. Check if there is a DB named my _ accounts 10

2.1. Statement 10

2.2. Snapshot 11

3. Create the DB my_accounts, select it to use, and create a table named Accounts with structure and contents as shown in the following commands Show the result by displaying (a) the table list of the DB, (b) the contents, and (c) the structure of the Accounts table 11

3.1. Statement 11

3.2. Snapshot 12

4. Display the contents of the Accounts table , incrementing the entries by 13

4.1. Statement 13

4.2. Snapshot 13

5. The ascending numbering that appeared in the column entitled No of step 4, should also be present in the Accounts table ? Justify your answer. 13

5.1. Statement 13

5.2. Snapshot 14

5.3. Justification 14

CUSTOMERS table shown in Figure 1. Set data types of the CUSTNO and CUST _ NAME columns to integer and varchar (30) respectively. Show the result by displaying (a) the table list of the NW, (b) the contents, and (c) the structure of the CUSTOMERS table 14

6.1. Statement 14

6.2. Snapshot 15

7. In the Accounts table add a column named Custno, data type integer and set it as FK of the Accounts table to link its records with the records of the CUSTOMERS table . Update the contents of the Custno column so that the account with AcctID =202 corresponds to customer code 20 and all other accounts correspond to customer code 10. Show the result by displaying (a) the contents and (b) the structure of the Accounts table 16

7.1. Statement 16

7.2. Snapshot 17

8. Execute and interpret the following SQL statements 17

8.1. Statement & Interpretation 17

8. 2 . Snapshot 18

9. Execute and interpret the following SQL statements 18

9.1. Statement & Interpretation 18

9.2. Snapshot 19

factorial function that calculates n !=1*2*…* n 19

10.1. Statement 19

10.2. Snapshot 21

11. Define and use the procedure my _ procedure _ Local _ Variables for calculations using local variables 21

11.1. Statement 21

11.2. Snapshot 22

12. Do the following to create a stored procedure and use commit / rollback . Explain what the procedure myProc ..... 22 does

12.1. Statement 22

12.2. Snapshot 24

12.3. Justification 24

13. In the Accounts table (Figure 1) the transfer of money from one account to another could be implemented with two UPDATE statements . Here is an example of solving using transaction (transaction). This transaction is characterized as unreliable, as no check is made regarding: (a) the existence of the account to which the money is transferred and (b) the adequacy of the account from which the money is transferred. 25

13.1. Statement 25

13.2. Snapshot 26

14. Follows the solution of the problem in step 15 using the procedure BankTransfer 26

1 4 .1. Statement 26

14. 2 . Snapshot 29

15. A second solution to problem ..... 30 follows

15.1. Statement 30

1 5 .2. Snapshot 34

DB personnel

personnel.sql

```
drop database if exists personnel?

create database personnel?

use personnel?


create table

DEPT (

DEPTNO numeric( 2),

DNAME varchar( 24 ),

LOC char( 23)

);

insert into

DEPT

(DEPTNO, DNAME, LOC)

values

(50, 'SALES', 'ATHENS'),

(60, 'ACCOUNTING', 'ATHENS'),

(70, 'PAYROL', 'VOLOS');


create table

JOB(

JOBCODE numeric( 3),

JOB_DESCR varchar( 24 ),

SAL numeric( 10.2)

);
```

```
insert into
JOB
(JOBCODE, JOB_DESCR, SAL)
values
(100, 'SALESMAN', 2000),
(200, 'ANALYST', 2000),
(300, 'DBA', 3000);

create table
EMP(
EMPNO numeric( 4),
NAME varchar( 255 ),
JOBNO numeric( 3),
DEPTNO numeric( 2),
COMM numeric( 10,2)
);
insert into
EMP
(EMPNO, NAME, JOBNO, DEPTNO, COMM)
values
(10, 'CODD', 100, 50, NULL),
(20, 'NAVATHE', 200, 50, 450 ),
(30, 'ELMASRI', 300, 60, NULL),
(40, 'DATE', 100, 50, NULL);
```

personnel.png

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

DB my_accounts

my_accounts.sql

```
DROP DATABASE IF EXISTS my_accounts ;

CREATE DATABASE my_accounts ;

USE my_accounts ;


CREATE TABLE Accounts ( acctID int not null primary key,

                                Balance int 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);

CREATE TABLE Customers ( custno int not null, cust_name varchar( 30 ), primary
key( custno ));


INSERT INTO Customers ( custno , cust_name ) VALUES (10, '101');

INSERT INTO Customers ( custno , cust_name ) VALUES (20, '202');
```

my _ accounts . png

Accounts

acctID	Balance
101	1000
202	2000
303	2500
404	3000

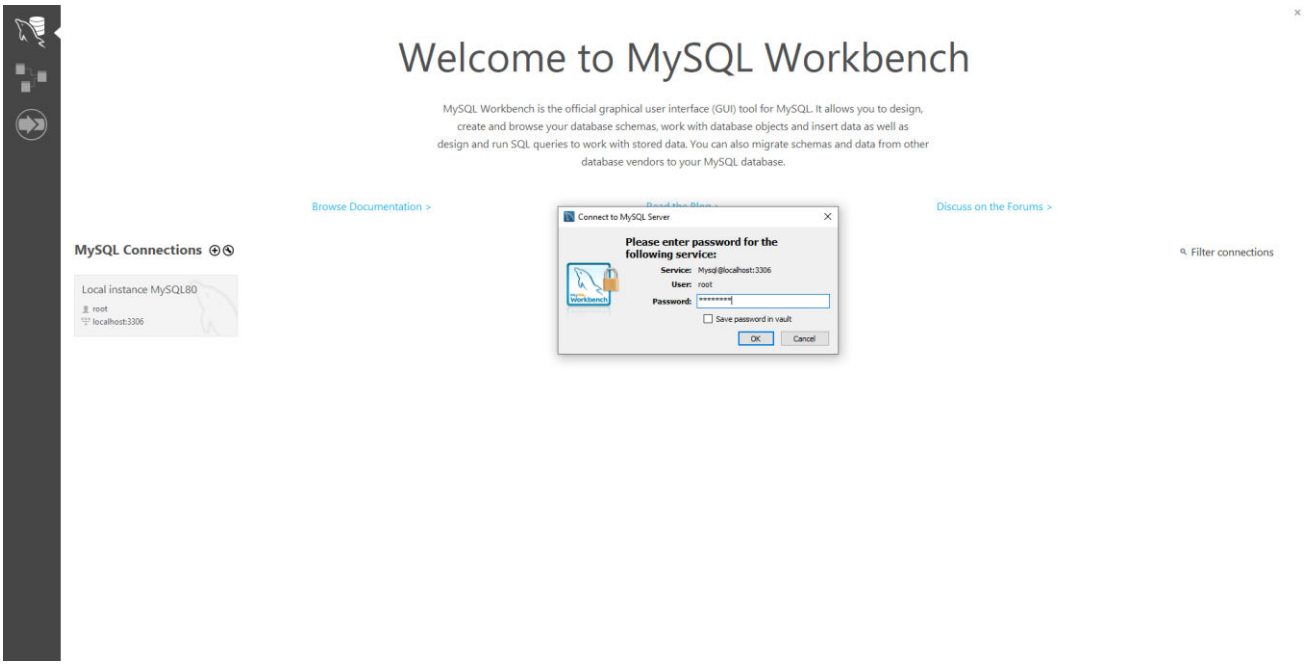
CUSTOMERS

CUSTNO	CUST_NAME
10	101
20	202

Activities

1. Connect to your system's MySQL using any of the above methods you wish

1.2. Snapshot



2. Check if there is a DB named my _ accounts

2.1. Statement

```
# Show all NWs

SHOW databases ;
```

2.2 . Snapshot

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

Database
delete_orders
information_schema
my_accounts
my_first_triggers_db
mysql
performance_schema
personnel
sys

3. Create the DB my_accounts, select it to use, and create a table named Accounts with structure and contents as shown in the following commands Show the result by displaying (a) the table list of the DB, (b) the contents, and (c) the structure of the Accounts table

3.1. Statement

```
# Create the my_accounts DB and the Accounts table
DROP DATABASE IF EXISTS my_accounts ;
CREATE DATABASE my_accounts ;
USE my_accounts ;
CREATE TABLE Accounts ( acctID int not null primary key ,
                                Balance int not null );

# Tests
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);

# (a) List of NW tables
```

```
SHOW tables ;

# ( b ) Contents of Accounts table

SELECT * FROM Accounts;

# ( c ) Structure of table Accounts

DESCRIBE Accounts?
```

3.2. Snapshot

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

Tables_in_my_accounts
accounts

(α) × (β) (γ)

Result Grid | Filter Rows:

acctID	Balance
101	1000
202	2000
303	2500
404	3000
NULL	NULL

(α) (β) × (γ)

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
acctID	int	NO	PRI	NULL	
Balance	int	NO		NULL	

(α) (β) (γ) ×

4 . Display the contents of the Accounts table , incrementing its entries

4.1. Statement

```
# The contents of the Accounts table with ascending numbering of its entries

SET @ rownum = 0;

SELECT (@ rownum := @ rownum + 1) AS No , acctID , Balance FROM Accounts ORDER
BY acctID ;
```

4.2. Snapshot

Result Grid | Filter Rows:

No	acctID	Balance
1	101	1000
2	202	2000
3	303	2500
4	404	3000

Rownum

5. The ascending numbering that appeared in the column entitled No of step 4 , should also be present in the Accounts table ? Justify your answer.

5.1. Statement

```
# The contents of the Accounts table with ascending numbering of its entries

SET @ rownum = 0;

SELECT (@ rownum := @ rownum + 1) AS No , acctID , Balance FROM Accounts ORDER
BY acctID ;

DESCRIBE Accounts?
```

5.2 . Snapshot

Result Grid | Filter Rows: | Export:

No	acctID	Balance
1	101	1000
2	202	2000
3	303	2500
4	404	3000

Rownum × Accounts

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
acctID	int	NO	PRI	NULL	
Balance	int	NO		NULL	

Rownum Accounts ×

5.3. Justification

The ascending numbering that appeared in the column titled No in step 4 should not also be present in the Accounts table , as the SELECT command prints results based on some arguments and therefore does not store them somewhere. CREATE and ALTER command , on the other hand, add columns to tables.

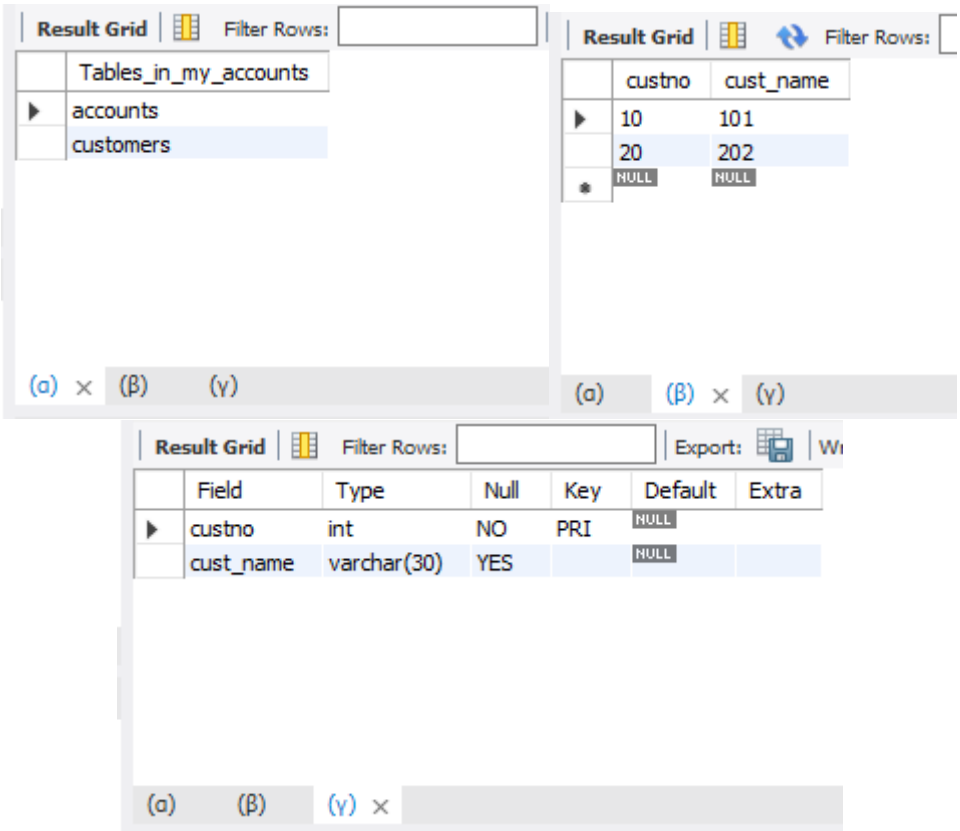
CUSTOMERS table shown in Figure 1 . Set data types of CUSTNO and CUST _ NAME columns to integer and varchar (30) respectively. Show the result by displaying (a) the table list of the NW, (b) the contents, and (c) the structure of the CUSTOMERS table .

6.1. Statement

```
# Create the Customers table in my_accounts DB

USE my_accounts ;

CREATE TABLE Customers ( custno int not null , cust_name varchar( 30 ) , primary
key ( custno ));
```

```
# Tests
INSERT INTO Customers ( custno , cust_name ) VALUES (10, '101');
INSERT INTO Customers ( custno , cust_name ) VALUES (20, '202');

# (a) List of NW tables
SHOW tables ;
# ( b ) Contents of Customers table
SELECT * FROM Customers;
# ( c ) Structure of Customers table
DESCRIBE Customers?
```

6.2. Snapshot



7 . In the Accounts table add a column named Custno, data type integer and set it as FK of the Accounts table to link its records with the records of the CUSTOMERS table . Update the contents of the Custno column so that the account with AcctID =202 corresponds to customer code 20 and all other accounts correspond to customer code 10. Show the result by displaying (a) the contents and (b) the structure of the Accounts table .

7.1. Statement

```
# Add foreign key custno to Accounts table
USE my_accounts ;
ALTER TABLE Accounts ADD custno int ;
ALTER TABLE Accounts ADD foreign key ( custno ) references Customers( custno );

# Tests
UPDATE Accounts SET custno = 20 WHERE acctID = 202;
UPDATE Accounts SET custno = 10 WHERE acctID <> 202;

# ( a ) Contents of Accounts table
SELECT * FROM Accounts;
# ( b ) Structure of Accounts table
DESCRIBE Accounts?
```

7.2 . Snapshot

Result Grid | Filter Rows:

acctID	Balance	custno
101	1000	10
202	2000	20
303	2500	10
404	3000	10
NULL	NULL	NULL

(α) × (β)

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
acctID	int	NO	PRI	NULL	
Balance	int	NO		NULL	
custno	int	YES	MUL	NULL	

(α) (β) ×

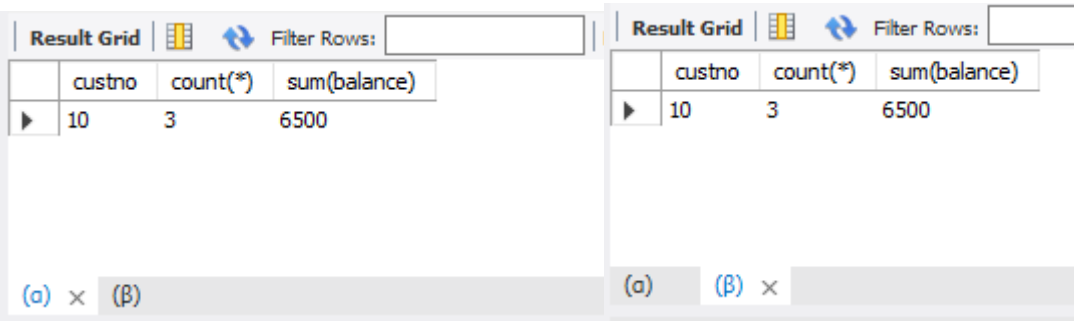
8. Execute and interpret the following SQL statements

8.1. Statement & Interpretation

```
# Using DB my_accounts
USE my_accounts;

# (a) Display the id, count and sum of the customer balances
# that do not have an id equal to 20. Display the results per customer
SELECT custno, count(*), sum(balance)
      FROM Accounts
      WHERE custno NOT IN (20)
      GROUP BY custno ;

SET @CUST_NO = 20;

# (b) Display the id, count and sum of the customer balances
# that do not have an id equal to the contents of the CUST_NO variable.
# Display the results per customer
SELECT custno, count(*), sum(balance)
```

```
FROM Accounts
WHERE custno NOT IN (@CUST_NO)
GROUP BY customer?
```

8. 2 . Snapshot



Result Grid | Filter Rows:

	custno	count(*)	sum(balance)
▶	10	3	6500

(α) × (β)

Result Grid | Filter Rows:

	custno	count(*)	sum(balance)
▶	10	3	6500

(α) (β) ×

9. Execute and interpret the following SQL statements

9.1. Statement & Interpretation

```
# Using DB my _ accounts
USE my_accounts ;

# (a) Display the count and sum of customer balances
SELECT count( *), sum(balance) FROM Accounts;

SET @COUNT_acctID = 0 , @SUM_acctID = 0, @AVG_acctID = 0;

# Store the customer count in the COUNT_acctID variable
# Store the sum of customer balances in the SUM_acctID variable
# Store the average of customer balances in the AVG_acctID variable
SELECT count( *), sum(balance), avg (balance)
      INTO @ COUNT_acctId , @ SUM_acctID , @ AVG_acctID
    FROM Accounts?
```

```
# (b) Display the contents of the COUNT_acctID variable,

# the contents of the SUM_acctID variable,

# the contents of the AVG_acctID variable,

# the contents of the MY_AVG variable, which contains the result

# of division SUM_acctID / COUNT_acctID

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

9.2. Snapshot

Result Grid | Filter Rows:

count(*)	sum(balance)
4	8500

(α) × (β)

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

@COUNT_acctID	@SUM_acctID	@AVG_acctID	@MY_AVG := @SUM_acctID/@COUNT_acctID
4	8500	2125.000000000	2125.000000000

(α) (β) ×

factorial function that calculates n !=1*2*…* n

10.1. Statement

```
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 ;

# (a) Display the result of 4!
SELECT factorial(4);

# (b) Display the result of 15!
SELECT factorial( 15 );
# Error Code: 1264. Out of range value for column 'F' at row 1
# The result of factoring 15! exceeds the maximum value
# which can be stored in type int
```

1 0.2. Snapshot

Result Grid | Filter Rows:

factorial(4)
24

(a) ×

Output

Action Output

#	Time	Action	Message
1	17:19:40	DROP FUNCTION IF EXISTS factorial	0 row(s) affected
2	17:19:40	CREATE FUNCTION factorial(N int) RETURNS int DETERMINISTIC BEGIN DECLARE F int DEFAULT 1; WHILE N > 0 DO SET F = N * F; SET...	0 row(s) affected
3	17:19:40	# (α) Εμφάνισε το αποτέλεσμα του 4! SELECT factorial(4) LIMIT 0, 1000	1 row(s) returned
4	17:19:40	SELECT factorial(15) LIMIT 0, 1000	Error Code: 1264. Out of range value for column 'F' at row 1

11. Define and use the procedure my _ procedure _ Local _ Variables for calculations using local variables

11.1. Statement

```
DROP PROCEDURE IF EXISTS my_procedure_Local_Variables ;

DELIMITER $$

CREATE PROCEDURE my_procedure_Local_Variables ( IN x int , IN y int )

BEGIN

    SET @X = x;

SET @Y = y;

    SELECT @X, @Y, @X*@Y;

END $$

DELIMITER ;


# ( a ) Calculation of 25*10

CALL my_procedure_Local_Variables ( 25 , 10 ) ;

# ( b ) Calculation 50*10

CALL my_procedure_Local_Variables ( 50 , 10 ) ;

# ( c ) Calculation 3*5
```

```
CALL my_procedure_Local_Variables ( 3 , 5 ) ;
```

11.2. Snapshot

Result Grid | Filter Rows:

	@X	@Y	@X*@Y
▶	25	10	250

(α) × (β) (γ)

Result Grid | Filter Rows:

	@X	@Y	@X*@Y
▶	50	10	500

(α) (β) × (γ)

Result Grid | Filter Rows:

	@X	@Y	@X*@Y
▶	3	5	15

(α) (β) (γ) ×

12. Do the following to create a stored procedure and use commit / rollback . Explain what the procedure myProc does

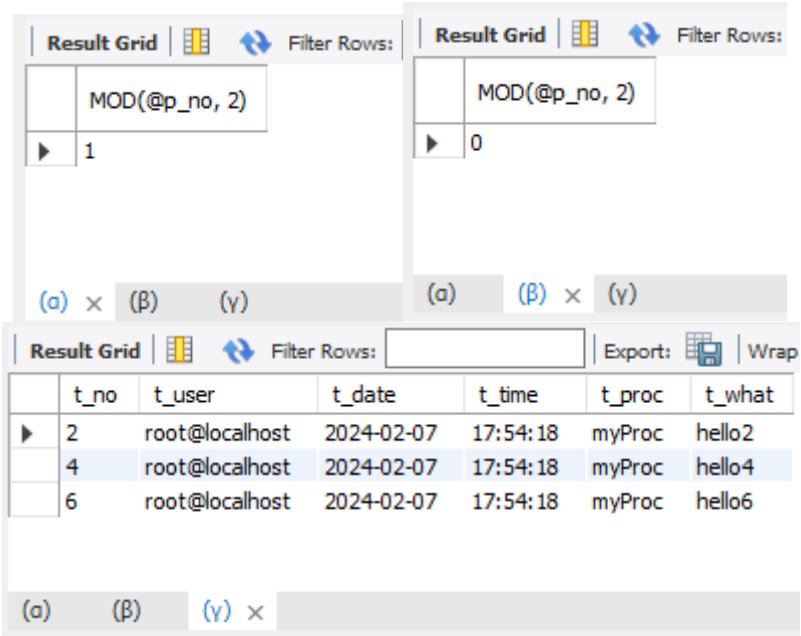
12.1. Statement

```
# Tests using the MOD function
SET @p_no = 3;
# (a) Display the result of operation 3 mod 2
SELECT MOD( @ p_no , 2);
SET @ p_no = 8;
# (b) Display the result of operation 8 mod 2
SELECT MOD(@p_no, 2);


# Create base and table
DROP DATABASE IF EXISTS trace;
CREATE DATABASE trace;
USE trace?
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));

# Create myProc stored procedure
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 ;

# Call her process
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 );


# (c) Display the contents of the myTrace array

SELECT * FROM myTrace;
```

12.2. Snapshot



Result Grid | Filter Rows:

	MOD(@p_no, 2)
▶	1

(α) × (β) (γ)

Result Grid | Filter Rows:

	MOD(@p_no, 2)
▶	0

(α) (β) × (γ)

Result Grid | Filter Rows: | Export: | Wrap

	t_no	t_user	t_date	t_time	t_proc	t_what
▶	2	root@localhost	2024-02-07	17:54:18	myProc	hello2
	4	root@localhost	2024-02-07	17:54:18	myProc	hello4
	6	root@localhost	2024-02-07	17:54:18	myProc	hello6

(α) (β) (γ) ×

12.3. Justification

The procedure myProc takes as input the id (p _ no) and the message (p _ in) of the trace (entry of the myTrace table of the NW trace) and produces as output the message of the trace (p _ out). After, assign the trace message on the output to a variable (SET p _ out = p _ in), the procedure performs an INSERT into the myTrace table with data as follows:

- The id of the trace (p _ no) in the field t _ no
- The current user (current _ user) in the t _ user field

- The current date (current _ date) in the t _ date field
- The current time (current _ time) in the t _ time field
- The name of the procedure (' myProc ') in the field t _ proc
- The trace message (p _ in) in the field t _ what

It then checks for the ids (p _ no) of traces that have been successfully written to the myTrace table . More specifically, the MOD routine performs the operation p _ no mod 2 and if the result is equal to 0, that is, it is an even trace id , then the COMMIT operation is executed , otherwise the ROLLBACK operation is executed . Thoroughly, if it is an even trace id then the change made to the myTrace table is committed (COMMIT), otherwise if it is an odd trace id the change made to the myTrace table is undone.

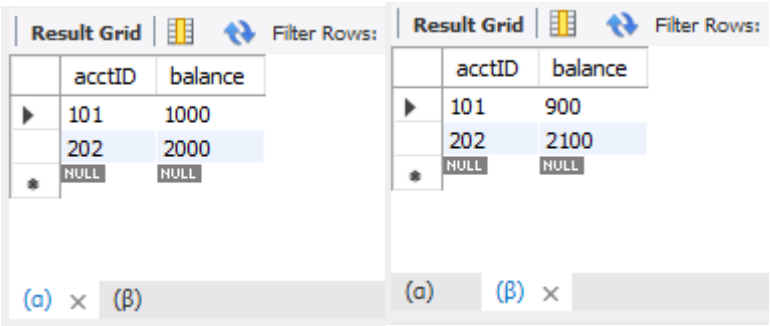
13. In the Accounts table (Figure 1) the transfer of money from one account to another could be implemented with two UPDATE statements . Here is an example of solving using transaction (transaction). This transaction is characterized as unreliable, as no check is made regarding: (a) the existence of the account to which the money is transferred and (b) the adequacy of the account from which the money is transferred.

13.1. Statement

```
Accounts table with 2 entries

USE my_accounts ;

DROP TABLE IF EXISTS Accounts?

CREATE TABLE Accounts ( acctID int not null primary key ,

      balance int not null ,

CONSTRAINT unloanable_account CHECK (balance >= 0));


# Tests

INSERT INTO Accounts ( acctID , balance) VALUES (101, 1000);

INSERT INTO Accounts ( acctID , balance) VALUES (202, 2000);

COMMIT ;

# (a) Display the contents of the Accounts table before the transaction

SELECT * FROM Accounts;
```

```
# Transaction

BEGIN ;

UPDATE Accounts SET balance = balance - 100

WHERE acctID = 101;

UPDATE Accounts SET balance = balance + 100

WHERE acctID = 202;

COMMIT ;


# (b) Display the contents of the Accounts table after the transaction

SELECT * FROM Accounts;
```

13.2. Snapshot



Result Grid | Filter Rows:

acctID	balance
101	1000
202	2000
NULL	NULL

(α) × (β)

Result Grid | Filter Rows:

acctID	balance
101	900
202	2100
NULL	NULL

(α) (β) ×

14. Follows the solution of the problem in step 15 using the procedure BankTransfer

14.1. Statement

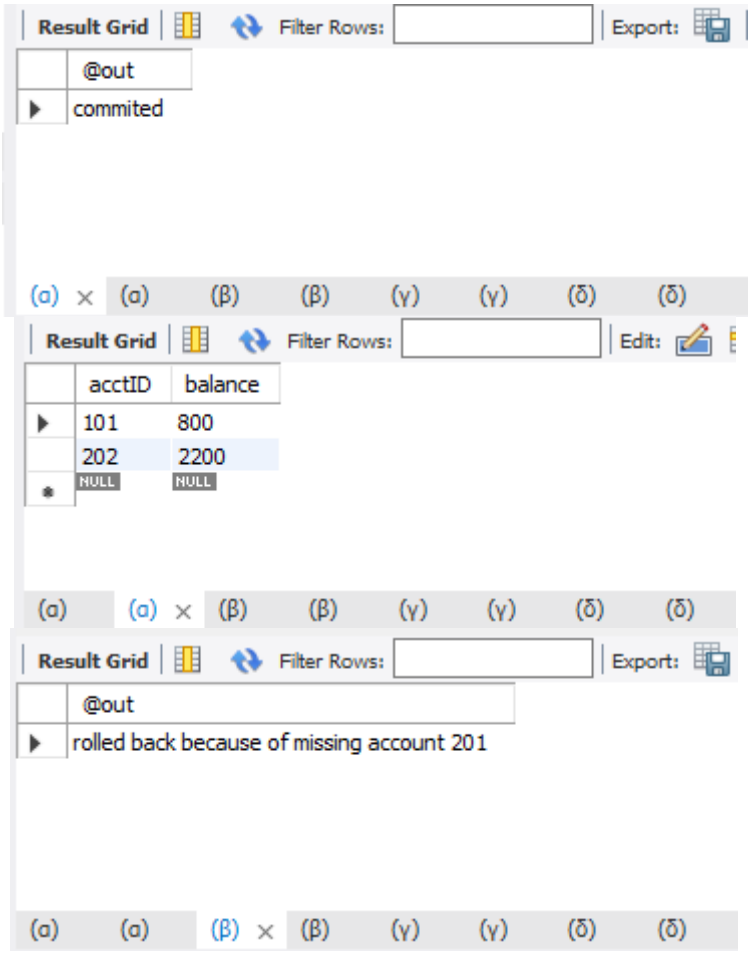
```
# Create procedure BankTrasfer

USE my_accounts ;

DELIMITER //

DROP PROCEDURE IF EXISTS BankTransfer //

CREATE PROCEDURE BankTransfer ( IN fromAcct INT ,
```

```
                                IN toAcct   INT ,
                                IN amount INT ,
                                OUT msg     VARCHAR ( 100 )
)
P1: BEGIN
DECLARE row_s INT ;
DECLARE newbalance INT ;
    SELECT count( *) INTO row_s FROM Accounts WHERE acctID = fromAcct ;
    UPDATE Accounts SET balance = balance - amount WHERE acctID = fromAcct ;
SELECT balance INTO new balance FROM Accounts WHERE acctID = fromAcct ;
IF row_s = 0 THEN
            ROLLBACK ?
            SET msg = concat ( 'rolled back due to missing account ', fromAcct
);
ELSEIF newbalance < 0 THEN
            ROLLBACK ?
            SET msg = concat ( 'rolled back because of negative balance of
account ', fromAcct );
   ELSE
      SELECT count( *) INTO row_s FROM Accounts WHERE acctID = toAcct ;
            UPDATE Accounts SET balance = balance + amount WHERE acctID = toAcct
;
            IF row_s = 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 ;

# (a) Test transfer 100 from acctID = 101 to acctID = 202
SET AUTOCOMMIT = 0;
SET @out = ' ';
CALL BankTransfer (101, 202, 100, @out);
SELECT @out;
SELECT * FROM Accounts;
COMMIT ;

# ( b ) Test transfer 100 from acctID = 101 in acctID = 201 ( does not exist )
SET AUTOCOMMIT = 0;
SET @out = ' ';
CALL BankTransfer (101, 201, 100, @out);
SELECT @out;
SELECT * FROM Accounts;
COMMIT ;

# ( c ) Test transfer 100 from acctID = 100 ( nil ) at acctID = 201
SET AUTOCOMMIT = 0;
SET @out = ' ';
CALL BankTransfer (100, 201, 100, @out);
SELECT @out;
SELECT * FROM Accounts;
COMMIT ;
```

```
# ( d ) Test transport 1500 from acctID = 101 ( insufficient ) at acctID = 201

SET AUTOCOMMIT = 0;

SET @out = ' ';

CALL BankTransfer (101, 201, 1500, @out);

SELECT @out;

SELECT * FROM Accounts;

COMMIT ;
```

14. 2 . Snapshot



Result Grid | Filter Rows: | Edit:

acctID	balance
101	800
202	2200
NULL	NULL

(α) (α) (β) (β) × (γ) (γ) (δ) (δ)

Result Grid | Filter Rows: | Export:

@out
rolled back because of missing account 100

(α) (α) (β) (β) (γ) × (γ) (δ) (δ)

Result Grid | Filter Rows: | Edit:

acctID	balance
101	800
202	2200
NULL	NULL

(α) (α) (β) (β) (γ) (γ) × (δ) (δ)

15. A second solution to the problem follows

15.1. Statement

```
# Create Accounts table
USE my_accounts ;
DROP TABLE IF EXISTS Accounts?
CREATE TABLE Accounts ( acctID int not null primary key ,
      balance int not null );
INSERT INTO Accounts ( acctID , balance) VALUES (101, 1000);
INSERT INTO Accounts ( acctID , balance) VALUES (202, 2000);
COMMIT ;
# (a) Display the contents of the Accounts table
```

```
SELECT * FROM Accounts;


# Create a trigger Accounts_upd_trg for control of 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 ;


# Create a trigger Accounts_ins_trg for control of the 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 ;
```

```
# Create procedure BankTransfer

DELIMITER !

DROP PROCEDURE IF EXISTS BankTransfer ?

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 ;

# (b) Test transfer 100 from acctID = 101 to acctID = 201 (non-existent)
CALL BankTransfer (101, 201, 100, @ msg );
SELECT @ msg ;
# (c) Test transfer 100 from acctID = 100 (nonexistent) to acctID = 202
CALL BankTransfer (100, 202, 100, @ msg );
SELECT @ msg ;
# (d) Test transfer 100 from acctID = 101 to acctID = 202
CALL BankTransfer (101, 202, 100, @ msg );
SELECT @ msg ;
# (e) Transfer test 2000 from acctID = 101 (insufficient) to acctID = 202
CALL BankTransfer (101, 202, 2000, @ msg );
SELECT @ msg ;
```

1 5 .2. Snapshot

Result Grid | Filter Rows:

acctID	balance
101	1000
202	2000
NULL	NULL

(α) × (β) (γ) (δ) (ε)

Result Grid | Filter Rows:

@msg
missing account 201

(α) (β) × (γ) (δ) (ε)

Result Grid | Filter Rows:

@msg
missing account 100

(α) (β) (γ) × (δ) (ε)

Result Grid | Filter Rows:

@msg
commited

(α) (β) (γ) (δ) × (ε)

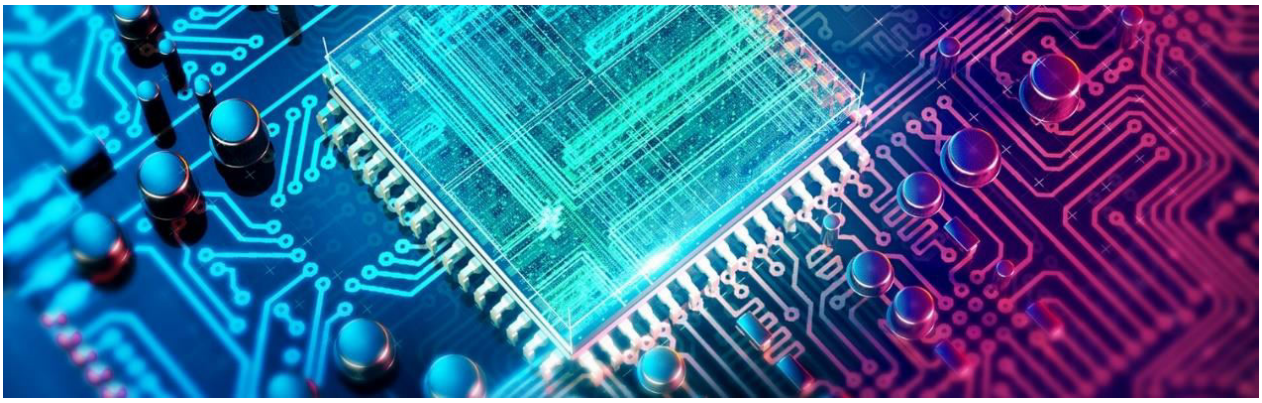
Result Grid | Filter Rows:

@msg
negative balance (?) in 101

(α) (β) (γ) (δ) (ε) ×

DATA BASES II

Thank you for your attention.



DATA BASES II