

Lab Session 10: Transactions\

In this lab, we will conduct a series of experiments using transactions. The main goal is to demonstrate the ACID properties of transactions, i.e. they are **atomic**, they guarantee that the database is always **consistent**, they are **isolated** from each another, and their effects are **durable** once the transaction commits.

As usual, we will use our bank database as an example. To be able to run multiple transactions at the same time, we will need to open multiple connections to the database. For this purpose, in this lab we will use an SSH client such as **PuTTY** (on Windows) or the command-line utility **ssh** (on Mac OS or Linux).

Part I: Concurrent transactions

1. Open a SSH connection to **sigma.ist.utl.pt**
 - On Windows, use **PuTTY** (see Figure 1 below)
 - On Mac OS or Linux, run the following command in a terminal:
ssh istxxxxxx@sigma.ist.utl.pt
where istxxxxxx is your username.
To connect to sigma.ist.utl.pt, you will have to provide you Fénix password.

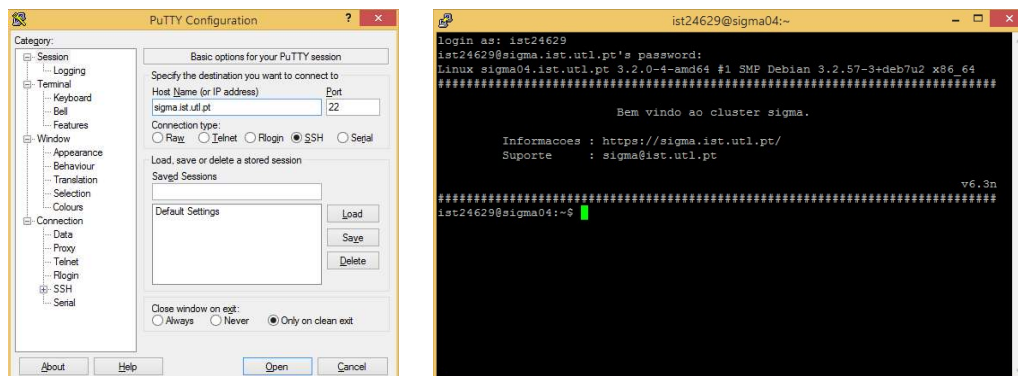


Figure 1. Opening an SSH connection with PuTTY

2. If you still remember your MySQL password, go to the next step. Otherwise, if you have forgotten your MySQL password, execute: **mysql_reset**
3. Connect to the MySQL server with the command:
mysql -h db.ist.utl.pt -u istxxxxxx -p

where istxxxxxx is your username.

When prompted, enter your MySQL password (the password given by **mysql_reset**).

4. Once you are connected to MySQL, execute the command:
source bank.sql
to re-create the database and start from a clean state.
5. Execute the following SQL query:
SELECT * FROM account NATURAL JOIN depositor;
in order to list the existing accounts and their respective customers.
6. Customer 'Johnson' has two accounts (A-101 and A-201) and we want to transfer 200 € from account A-101 to account A-201. Execute the following query:
SELECT * FROM account NATURAL JOIN depositor WHERE customer_name = 'Johnson';
in order to see how much money 'Johnson' has in both accounts.
7. Now we will start the bank transfer. Execute the following command to begin a new transaction:
START TRANSACTION;
8. Execute the following command in order to take 200 € from account 'A-101':
UPDATE account SET balance = balance - 200 WHERE account_number = 'A-101';
9. Now execute the following command in order to put 200 € in account 'A-201':
UPDATE account SET balance = balance + 200 WHERE account_number = 'A-201';
10. Execute again the following query in order to check the balance of both accounts:
SELECT * FROM account NATURAL JOIN depositor WHERE customer_name = 'Johnson';
11. For the moment, leave the previous transaction as it is (do not close the window) and open a second connection to the database. You can do this as follows:
 - Launch **PuTTY** again (or **ssh** in a new terminal) and open a second SSH connection to **sigma.ist.utl.pt**. Use your Fénix credentials.
 - Connect to the MySQL server with the command
mysql -h db.ist.utl.pt -u istxxxxxx -p
Use the password given by **mysql_reset**.
 - Once connected to MySQL, execute the command **use istxxxxxx** to select your database.
12. In the second connection, execute the following query:

```
SELECT * FROM account NATURAL JOIN depositor WHERE customer_name = 'Johnson';
```

13. Why is the transfer of 200 € not showing up?

14. Go back to the first connection and execute:

```
COMMIT;
```

15. Now go to the second connection and repeat the query:

```
SELECT * FROM account NATURAL JOIN depositor WHERE customer_name = 'Johnson';
```

16. Now the second connection sees that the bank transfer has been done. This is the new state of the database. Keep both connections open for the next steps.

| |
|--------------------------------------|
| Part II: Blocked transactions |
|--------------------------------------|

17. In the first connection, begin a new transaction:

```
START TRANSACTION;
```

18. In the second connection, begin a new transaction:

```
START TRANSACTION;
```

19. In the first transaction, take 50 € from account A-101:

```
UPDATE account SET balance = balance - 50 WHERE account_number = 'A-101';
```

20. In the second transaction, take 25 € from account A-101:

```
UPDATE account SET balance = balance - 25 WHERE account_number = 'A-101';
```

21. The second transaction freezes. The system has blocked this operation. Why?

22. In the first transaction, execute **COMMIT**;

23. Check what happens in the second transaction. The system now allows the second transaction to continue.

24. In the second transaction, execute **COMMIT**;

25. Then execute the following query in both sessions:

```
SELECT balance FROM account WHERE account_number = 'A-101';
```

26. Check that the result is the same (225 €). This is the new state of the database.

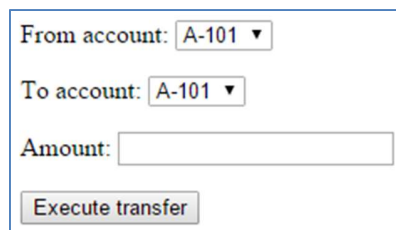
Part III: Transactions with PDO

The following PHP/HTML code creates a form to specify a bank transfer of a given amount between two existing accounts. Take a moment to understand what this code is doing:

```
<?php
    $host = "db.ist.utl.pt";
    $user = "istxxxxxx";
    $pass = "xxxxxxx";
    $dsn = "mysql:host=$host;dbname=$user";
    $connection = new PDO($dsn, $user, $pass);
    $sql = "SELECT account_number FROM account ORDER BY account_number";
    $result = $connection->query($sql);
    $rows = $result->fetchAll();
    $connection = null;
?>
<html>
<body>
<form action="exctransfer.php" method="post">
    <p>From account:
        <select name="account_from">
            <?php
                foreach($rows as $row)
                {
                    $account_number = $row['account_number'];
                    echo("<option
value=\"\$account_number\">\$account_number</option>");
                }
            ?>
        </select>
    </p>
    <p>To account:
        <select name="account_to">
            <?php
                foreach($rows as $row)
                {
                    $account_number = $row['account_number'];
                    echo("<option
value=\"\$account_number\">\$account_number</option>");
                }
            ?>
        </select>
    </p>
    <p>Amount:
        <input type="text" name="amount"/>
    </p>
    <p><input type="submit" value="Execute transfer"/></p>
</form>
</body>
</html>
```

(Note: For brevity, this code does not do any error checking. This is not recommended.)

27. Save the code above in a file called **formtransfer.php** on your local machine.
28. Replace the values of **\$user** and **\$password** with your login information for MySQL (your username from Fénix, and the password from **mysql_reset**)
29. Using an SFTP client, move or copy the file **formtransfer.php** into your "web" folder on sigma.ist.utl.pt.
30. Open the Web browser on your local machine and navigate to:
<http://web.tecnico.ulisboa.pt/istxxxxxx/formtransfer.php>
where **istxxxxxx** is to be replaced by your username.
31. You should see the following form appearing in your browser:



The screenshot shows a web form with a blue border. It contains three input fields and one button. The first field is labeled 'From account:' and has a dropdown menu with 'A-101' selected. The second field is labeled 'To account:' and also has a dropdown menu with 'A-101' selected. The third field is labeled 'Amount:' and is a text input box. Below these fields is a button labeled 'Execute transfer'.

The following PHP/HTML code executes the bank transfer. Take a moment to understand what this code is doing:

```
<html>
<body>
<?php
    $account_from = $_REQUEST['account_from'];
    $account_to = $_REQUEST['account_to'];
    $amount = (float)$_REQUEST['amount'];
    if ($amount > 0.0)
    {
        $host = "db.ist.utl.pt";
        $user = "istxxxxxx";
        $pass = "xxxxxxx";
        $dsn = "mysql:host=$host;dbname=$user";
        $connection = new PDO($dsn, $user, $pass);

        $connection->beginTransaction();

        $sql = "UPDATE account SET balance=balance-$amount WHERE
account_number='$account_from'";
        $connection->exec($sql);

        $sql = "SELECT balance FROM account WHERE account_number='$account_from'";
        $result = $connection->query($sql);
        $row = $result->fetch();
        $balance = $row['balance'];

        if ($balance < 0)
        {
            echo("<p>Not enough balance!</p>");
            $connection->rollback();
        }
        else
        {
            $sql = "UPDATE account SET balance=balance+$amount WHERE
account_number='$account_to'";
            $connection->exec($sql);
            $connection->commit();
            echo("<p>Transfer complete!</p>");
        }

        $connection = null;
    }
    else
    {
        echo("<p>Please specify a positive amount.</p>");
    }
?>
</body>
</html>
```

(Note: For brevity, this code does not do any error checking. This is not recommended.)

32. Save the code above in a file called **exctransfer.php** on your local machine.

33. Replace the values of **\$user** and **\$password** with your login information for MySQL (your username from Fénix, and the password from **mysql_reset**)
34. Using an SFTP client, move or copy the file **exectransfer.php** into your "web" folder on sigma.ist.utl.pt.
35. Open the Web browser on your local machine and navigate to the transfer form:
<http://web.tecnico.ulisboa.pt/istxxxxxx/formtransfer.php>
where **istxxxxxx** is to be replaced by your username.
36. Transfer 200 € from account A-101 to A-102.
When you press **Execute transfer**, you should see "Transfer complete".
The transaction has **committed**. (Where did this happen in the code?)
37. On a MySQL command line, check the new balance of these accounts:
SELECT * FROM account;
Account A-101 should have 25 € only.
38. Go back to the transfer form:
<http://web.tecnico.ulisboa.pt/istxxxxxx/formtransfer.php>
39. Transfer 100 € from account A-101 to A-102.
When you press **Execute transfer**, you should see "Not enough balance".
The transaction has been **rolled back**. (Where did this happen in the code?)
40. Suppose that instead of specifying a positive amount to be transferred, you specify a negative amount. Try to execute a transfer of -200 € from account A-101 to A-102.
What happens?
41. In the transfer form, the amount is specified using a text field. Do you think it is possible to carry out an SQL injection attack through this field? Justify your answer based on an analysis of the code.