**CST-236 Activity 5 Guide**

Contents

[A.C.I.D. Transactions 2](#_Toc530481049)

[Activity 5 Overall Submissions 4](#_Toc530481050)

A.C.I.D. Transactions

**Overview**

In this activity, students will learn how to support ACID database transactions in PHP.

**Execution**

1. Create the following support Tables in MySQL in a new schema called ica17:
   1. Create a CHECKING Table with the following columns:
      1. ID that is auto generated.
      2. BALANCE that is a float.
   2. Create a SAVING Table with the following columns:
      1. ID that is auto generated.
      2. BALANCE that is a float.
   3. Initialize the Check balance to a value of 1000 and initialize the Saving balance to a value of 2000 by running appropriate insert SQL statements.
2. Create a CheckAccount and SavingAccount Data Services:
   1. Create a class in Eclipse PHP called *CheckingAccountDataService*. Use a Database class from a prior activity to get a database connection.
   2. Add a public method called *getBalance()* that returns the BALANCE column values from the CHECKING table.
   3. Add a public method called *updateBalance*($balance) that updates the balance of the CHECKING table from the method argument. Return a 1 to indicate success.
   4. Create a class in Eclipse PHP called *SavingAccountDataService*. Use a Database class from a prior activity to get a database connection.
   5. Add a public method called getBalance() that returns the BALANCE column values from the SAVING table.
   6. Add a public method called updateBalance($balance) that updates the balance of the SAVING table from the method argument. Return a 1 to indicate success.
   7. Test your Data Service classes (with auto commit enabled) by writing a Test PHP script:
      1. Create an instance of a *CheckingAccountDataService*.
      2. Display the Checking Balance.
      3. Update the Checking Balance by 100.
      4. Display the Checking Balance.
      5. Create an instance of a *SavingAccountDataService*.
      6. Display the Saving Balance.
      7. Update the Saving Balance by 100.
      8. Display the Saving Balance.
      9. Validate that checking and saving account balances were updated properly.
3. Create a Bank Business Service class:
   1. Create a class in Eclipse PHP called *BankBusinessService*.
   2. Refactor Data Services:
      1. Remove any mysqli code that returns a DB connection from the implementation.
      2. Add a constructor to each Data Service that takes a DB connection argument (of type mysqli) and save this in a private class member variable. Use the class member variable connection in all query().
   3. Add a public method called *getCheckingBalance*() to the BankBusinessService class that calls the *CheckingAccountDataService getBalance()* method.
   4. Add a public method called *getSavingBalance*() to the *BankBusinessService* class that calls the *SavingAccountDataService getBalance()* method.
   5. Add a public method called *transaction*()to the *BankBusinessService* class.
   6. Implement the following workflow in the *transaction*() method:
      1. Get a connection from the database and save in a variable $conn.
      2. Turn off Autocommit using $conn->autocommit(FALSE).
      3. Begin a transaction using $conn->begin\_transaction().
      4. Call the *CheckingDataService* using the *getBalance* () method. Subtract 100 from the balance value returned from this method. Then update the balance by calling *CheckingDataService* using the *updateBalance*() method. Save the return value from the *updateBalance*() method.
      5. Call the *SavingDataService* using the *getBalance* () method. Add 100 from the balance value returned from this method. Then update the balance by calling *SavingDataService* using the *updateBalance*() method. Save the return value from the *updateBalance*() method.
      6. Commit the transaction using $conn->commit() if the return value is 1 from the *CheckingDataService updateBalance*() AND the return value is 1 from the *SavingDataService updateBalance*() else rollback the transaction by calling $conn->rollback().
      7. Write a Test PHP script to Test your Bank Service class transaction() method:
         1. Create an instance of a *BankBusinessService*.
         2. Display the Checking Balance.
         3. Display the Saving Balance.
         4. Make a Bank Transaction.
         5. Display the Checking Balance.
         6. Display the Saving Balance.
         7. Validate that checking and saving account balances were updated properly.
      8. Test a failed *SavingDataService updateBalance()* by hard coding the return value from this method from 1 to 0.
      9. Run the Test Script again. Test your *BankBusinessService* class *transaction*() method and validate that checking and saving account balances were both rolled back properly.

Activity 5 Overall Submissions

Submit the following to the learning management system:

A report (using Word), containing your name, course, assignment name, and date, as well as the following components:

1. An ER diagram of your database table design. Simply capture and paste a screenshot of the ER diagram created by MySQL workbench.
2. A drawn UML diagram for the classes involved in this transaction. Include CheckAccountDataService, SavingAccountDataService, BankBusinessService and the Client (the Test Script)). Show how the components are related to each other. You can draw the diagram in a design tool such as Draw.io or Lucid Charts or even using pen and paper.
3. Screenshots from the web browser that show that checking and saving account balances were updated properly. These are done in Step 2g and Step 3f.
4. A theory of operation (in at least 100 words) explaining how this design supports an n-layer architecture and ACID. Also provide the rationale for why this design is better than running with auto commit enabled in your Data Services.