**Take Assessment: Exercise 5**

窗体顶端

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
|  | Please answer the following question(s). If the assessment includes multiple-choice questions, click the "Submit Answers" button when you have completed those questions. |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1. |  | [Go to bottom of question.](https://www.icarnegie.com/takeassmcmd.php?course_section_id=9788469&assm_id=6860314#6860317#6860317) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Programming with Transactions**  **Testing Commits and Rollbacks**  Almost all commercial-strength DBMS software supports the ability to undo the effects of a transaction before those effects are committed to the database. This is particularly useful when an operation must be performed to completion or not performed at all. You may undo these effects on the data using rollbacks. This portion of the exercise will familiarize you with rollbacks and commits.   1. Run the [SQL script](https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/build_trans_db.sql) provided to build a small bank database consisting of two columns: an account id number and a balance. There are two accounts, a savings account and a checking account. The savings account has *id = 1* and the checking account has *id = 2*. 2. Begin a new transaction. 3. Select balance of the checking account and paste the output in a file named *rollback.txt*. 4. Delete both the savings and the checking accounts. Place the SQL query to perform this operation in *rollback.txt*. 5. Select all of the data in the account table and paste the output in *rollback.txt*. 6. Roll back the transaction. 7. Select the balance of the savings and checking accounts and paste the output in *rollback.txt*. 8. Begin a new transaction. 9. Delete the savings account. Place the SQL code to do this in *rollback.txt*. 10. Select all rows from the account table and paste the output in *rollback.txt*. 11. Commit the transaction. 12. Attempt to rollback the transaction that you have just committed. What values are stored in the tables? Please explain the effects of transaction commits and rollbacks you have observed above. Place your explanation in *rollback.txt*.   **Testing Isolation Levels**  PostgreSQL supports two transaction isolation levels. To become familiar with PostgreSQL isolation levels, perform the following tasks:   1. Run the [SQL script](https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/build_trans_db.sql) provided to build a small bank database consisting of two columns: an account id number and a balance. There are two accounts, a savings account and a checking account. The savings account has *id = 1* and the checking account has *id = 2*. 2. Begin two sessions of the PostgreSQL client in two separate console windows. 3. Begin new transactions in both windows. 4. In the first window, update the checking account to have a balance of 455.66. 5. In the first window, select all of the data from the account table and place the output into the file *isolation.txt*. Be sure to label the data so that it is clear this data is from the first table. 6. In the second window, select all of the data from the account table and place the output into the file *isolation.txt*. Be sure to label the data so that it is clear this is data from the second window. 7. Commit the transaction in the first window to update the account table. 8. In the second window, select all of the data from the account table and place the output into the file *isolation.txt.* Be sure to label the data so that it is clear this is data from the second window. What has changed about the data and why? Place the answer to this question in *isolation.txt*. 9. Commit the transaction in second window. 10. In both windows, begin new transactions. 11. Set the transaction isolation level of the transaction in the second window to serializable. Place the code to do this in *isolation.txt*. 12. In the first window, set the balance of checking account balance to 1400.00. 13. In the second window, set the balance of savings account balance to 1. 14. Select all data from the account table in the first and second window and place it into *isolation.txt*. Be sure to label the data clearly to denote what data came from the first and second windows. 15. Commit the data in the first window. 16. Select all of the data from the account table in the second window. Has the data for the checking account changed? Why or why not? Has the data for the savings account changed in the first window? Why or why not? Places the answers to these questions in *isolation.txt* 17. Commit the data in the second window. Display all of the data from the account table in both windows. Place the output of both windows into *isolation.txt*. Clearly label which data is from the first window and which data is from the second window. What do you notice now about the balances of the checking and savings accounts? Please explain. Place your explanation in *isolation.txt*.   **Blocking and Deadlocks**  In this portion of the exercise, you will cause two transactions to block and deadlock two transactions in PostgreSQL.   1. Run the [SQL script](https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/build_trans_db.sql) provided to build a small bank database consisting of two columns: an account id number and a balance. There are two accounts, a savings account and a checking account. The savings account has *id = 1* and the checking account has *id = 2*. 2. Begin two sessions of the PostgreSQL client in two separate console windows. 3. Begin new transactions in both windows. 4. Update the checking account's balance to 455.75 in the second window. 5. Update the checking account's balance to 1400.00 in the first window. Does the update occur? Why or why not? Place your answer in *deadlock.txt*. 6. Commit the transaction in the second window. What do you notice happening in the first window? Please explain. Place your answer in *deadlock.txt*. 7. Commit the transaction in the first window. 8. Begin a new transaction in each window. 9. Update the savings account balance to 2400.00 in the second window. 10. Update the checking account balance to 2000.00 in the first window. 11. Update the savings account balance to 1400.00 in the first window. 12. Update the checking account balance to 1000.50 in the second window. 13. What happens to the transactions? Why? Place your answer in *deadlock.txt*. 14. Try selecting the all of the data from the account table in each window. What do you notice? Place your answer in *deadlock.txt*.   **Application**  The final exercises will test your knowledge of transactions and their behavior in real world situations.   1. Suppose William and Julie share a savings and a checking account at a bank. The bank has many automatic teller machines. William banks from one ATM while Julie banks at another ATM. William wishes first to deposit four hundred dollars into the savings account and then to transfer three hundred dollars from the checking account to the savings account. Julie wishes first to deposit a check for five hundred dollars into the checking account and then withdraw one hundred dollars from the savings account.   The sequence of operations for a deposit is:   * + Select the current account balance for the customer from the database.   + Increase this account balance by the deposit amount.   + Update the customer's account balance in the database to the new amount.   The sequence of operations for a withdrawal is:   * + Select the current account balance for the customer from the database.   + Decrease this account balance by the withdrawal amount.   + Update the customer's account balance in the database to the new amount.   The sequence of operations for a transfer is:   * + Select the current account balance for the account from where the funds are transferred.   + Decrease the current balance by the amount transferred.   + Select the balance of the account to where the funds are being transferred.   + Increase the balance of the account where money is transferred.   + Update both account balances in the database.   Suppose the account activities occur as an interleaved execution of transactions updating the database. You will now analyze a number of sample scenarios involving these account activities. Each scenario fixes one or more problems with the previous scenario, but there still may be problems in the scenario. You must identify what problem was fixed as well as why the fix leads to a more correct execution. Note that in the scenario below, bulleted operations are those that occur in the DBMS, non-bulleted items are those that occur at the ATM machine. Also, you should assume that these scenarios are run on PostgreSQL meaning that you may use only those isolation levels supported by PostgreSQL. You may assume that no other transactions are occurring in the database except these transactions. This means that you do not need to consider the potential effects of other types of transactions on the data. Also note that the ordering of these operations must be consistent for each scenario (all of William's operations occur in the proper order for his operations and all of Julie's operations occur in the proper order for her set of operations, that is the time flows from top to bottom for each user's transactions). You cannot assume that all of William's transaction will occur before all of Julie's or vice-versa. You also cannot assume that the operations will always be interleaved. It is possible that all of Julie's operations will occur before all of Williams, all of William's will occur before all of Julie's, and any interleaving of the operations is also possible.   * + Scenario A  |  |  | | --- | --- | | William | Julie | | * + - Begin transaction (read committed isolation level) | * + - Begin transaction (read committed isolation level) | | ATM prompts user for operation. | ATM prompts user for operation. | | William Chooses Deposit into Savings. | Julie Chooses Deposit into Checking. | | ATM prompts user for amount. | ATM prompts user for amount. | | William enters 400. | Julie enters 500. | | * + - Select balance of savings | * + - Select balance of checking | | savings\_balance = savings\_balance + $400 | checking\_balance = checking\_balance + $500 | | * + - Update savings balance in database. | * + - Update checking balance in database. | | ATM displays confirmation of deposit. | ATM displays confirmation of deposit. | | ATM prompts user for operation. | ATM prompts user for operation. | | William chooses transfer from checking to savings. | Julie chooses withdrawal from savings. | | ATM prompts user for amount. | ATM prompts user for amount. | | William enters 300. | Julie enters 100. | | * + - Select the checking balance. | * + - Select the savings checking balance in database. | | checking\_balance = checking\_balance - $300 | savings\_balance = savings\_balance - $100 | | * + - Select the savings balance. | * + - Update savings balance in database. | | savings\_balance = savings\_balance + $300 | ATM displays confirmation of withdrawal. | | * + - Update savings balance in database. | ATM prompts user for operation. | | * + - Update checking balance in database. | Julie chooses no more operations. | | ATM displays confirmation of transfer. | * + - End Transaction | | ATM prompts user for operation. |  | | William chooses no more operations. |  | | * + - End Transaction |  |  * + We will help you with this first scenario. The following problems exist:     - Incorrect Isolation Level     - Confirmation Messages are not in the correct positions     - Incorrect Transaction Boundaries     - Pausing for user input within transaction boundaries   For each of these problems, you must state why they are a problem. We will provide with help on one of them and you must explain the rest.   The confirmation messages are placed such that a user can receive a confirmation message before the transaction ends. This is a problem because a transaction may roll back but the user believes his transaction has ended successfully. All confirmation messages must be displayed after a transaction has been committed.   * + Scenario B  |  |  | | --- | --- | | William | Julie | | ATM prompts user for operation. | ATM prompts user for operation. | | William Chooses Deposit into Savings. | Julie Chooses Deposit into Checking. | | * + - Begin transaction (read committed isolation level) | * + - Begin transaction (read committed isolation level) | | ATM prompts user for amount. | ATM prompts user for amount. | | William enters 400. | Julie enters 500. | | * + - Select balance of savings | * + - Select balance of checking | | savings\_balance = savings\_balance + $400 | checking\_balance = checking\_balance + $500 | | * + - Update savings balance in database. | * + - Update checking balance in database. | |  |  | | * + - End Transaction | * + - End Transaction | | ATM displays confirmation of deposit. | ATM displays confirmation of deposit. | | ATM prompts user for operation. | ATM prompts user for operation. | | William chooses transfer from checking to savings. | Julie chooses withdrawal from savings. | | * + - Begin transaction (read committed isolation level) | * + - Begin transaction (read committed isolation level) | | ATM prompts user for amount. | ATM prompts user for amount. | | William enters 300. | Julie enters 100. | | * + - Select the checking balance. | * + - Select the savings checking balance in database. | | checking\_balance = checking\_balance - $300 | savings\_balance = savings\_balance - $100 | | * + - Select the savings balance. | * + - Update savings balance in database. | | savings\_balance = savings\_balance + $300 | * + - End Transaction | | * + - Update savings balance in database. | ATM displays confirmation of withdrawal. | | * + - Update checking balance in database. | ATM prompts user for operation. | | * + - End Transaction | Julie chooses no more operations. | | ATM displays confirmation of transfer. |  | | ATM prompts user for operation. |  | | William chooses no more operations. |  |  * + Scenario C  |  |  | | --- | --- | | William | Julie | | ATM prompts user for operation. | ATM prompts user for operation. | | William Chooses Deposit into Savings. | Julie Chooses Deposit into Checking. | | * + - Begin transaction (serializable isolation level) | * + - Begin transaction (serializable isolation level) | | ATM prompts user for amount. | ATM prompts user for amount. | | William enters 400. | Julie enters 500. | | * + - Select balance of savings | * + - Select balance of checking | | savings\_balance = savings\_balance + $400 | checking\_balance = checking\_balance + $500 | | * + - Update savings balance in database. | * + - Update checking balance in database. | |  |  | | * + - End Transaction | * + - End Transaction | | ATM displays confirmation of deposit. | ATM displays confirmation of deposit. | | ATM prompts user for operation. | ATM prompts user for operation. | | William chooses transfer from checking to savings. | Julie chooses withdrawal from savings. | | * + - Begin transaction (serializable isolation level) | * + - Begin transaction (serializable isolation level) | | ATM prompts user for amount. | ATM prompts user for amount. | | William enters 300. | Julie enters 100. | | * + - Select the checking balance. | * + - Select the savings checking balance in database. | | checking\_balance = checking\_balance - $300 | savings\_balance = savings\_balance - $100 | | * + - Select the savings balance. | * + - Update savings balance in database. | | savings\_balance = savings\_balance + $300 | * + - End Transaction | | * + - Update savings balance in database. | ATM displays confirmation of withdrawal. | | * + - Update checking balance in database. | ATM prompts user for operation. | | * + - End Transaction | Julie chooses no more operations. | | ATM displays confirmation of transfer. |  | | ATM prompts user for operation. |  | | William chooses no more operations. |  |  * + You must write the final sequence of events that lead to a correct execution while allowing the maximum amount of concurrency. Be sure to state each step for both William and Julie. Also, state what actions are handled by the DBMS and what actions are handled by the ATM machine.  1. Suppose you are asked to write a Web-based database application where tickets may be purchased for a local movie theater by a large number of Web clients. You may assume that the movie theater customers usually wait until thirty minutes or less before a show begins, thus the system has many users accessing the site concurrently. The sequence of events for a typical customer:    * The system queries the database for a list of current movies and displays them.    * The user selects the movie he wants to see from a list.    * There are typically many showings of the same movie throughout the day and the user must select which showing he will attend. To this end, the system queries the database for all of the daily showings of the chose movie and displays them. The user must pick one of these times.    * The system queries the DBMS and returns how many seats are available for the selected show.    * If there are seats available, the system displays the count to the user.    * The user must enter how many tickets he wants to buy.    * Finally, the user confirms his order and completes payment.   The table structure for this scenario along with some sample data is given below. Create a new database in PostgreSQL named movie and run the [SQL script](https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/create_movie.sql) provided to build these tables.   |  |  | | --- | --- | | Movies | | | Movie Id (PK) | Movie Title | | 001 | *The Terminator* | | 002 | *James Bond* | | 003 | *The Matrix* |  |  |  |  |  | | --- | --- | --- | --- | | Movie Showings | | | | | Movie Id (PK,FK) | Movie Time(PK) | Total Seats | Available Standard Seats | | 001 | 1:00 PM | 40 | 25 | | 001 | 4:00 PM | 30 | 27 | | 002 | 12:30 PM | 50 | 32 | | 002 | 7:30 PM | 75 | 17 | | 003 | 9:30 PM | 100 | 10 |   You must modify the [Java console application provided](https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/MovieTransaction.java) that simulates this scenario using PostgreSQL meaning that you can only use those isolation levels supported by PostgreSQL. This application sets incorrect transaction boundaries and uses an incorrect isolation level. Your task is to correct these mistakes by moving the transaction boundaries to where they allow the best performance for all system users. Note that you may use more than one transaction in your solution. For each transaction that is used in your solution, you must include the following:   * + An explanation as to where the transaction begins and ends (make specific references to the code immediately before and after it).   + State why these boundaries allow for the best performance.   + State why the isolation level you have selected will lead to a consistent view of the data to all users.   To help yourself do your best on this assessment, consult this general list of [grading guidelines](https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/rubrics.html). |  |

|  |  |  |
| --- | --- | --- |
|  | [Go to top of question.](https://www.icarnegie.com/takeassmcmd.php?course_section_id=9788469&assm_id=6860314#top_6860317#top_6860317) |  |

# 练习九 事务编程

**第一部分：测试数据库事务提交和回滚**

几乎所有的商业DBMS软件都会支持在事务提交前的回滚，这对于保证事务原子特性非常有用。你可以使用rollback命令撤销掉部分操作。此部分练习将会让你熟悉rollback和commit命令。

1. 运行[SQL script](mhtml:mk:@MSITStore:D:\job\ssd7\SSD7带练习版本.chm::/DOssd7/Exercise/9.mht!https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/build_trans_db.sql)建立一个小的银行数据库，account关系包含两列，用户id和账户余额。有两行数据，定期储蓄账户（id为1）和活期储蓄账户（id为2）。
2. 开始一个新事务。
3. 查询活期储蓄账户余额，将查询结果拷贝到rollback.txt文档中。
4. 删除定期和活期两个储蓄账户，将SQL语句拷贝到rollback.txt文档中。
5. 查询两个账户的信息，将结果拷贝入rollback.txt文档中。
6. 回滚事务。
7. 查询两个账户的余额，将结果拷贝入rollback.txt文档中。
8. 开始一个新事务。
9. 删除定期储蓄账户，将SQL语句拷贝入rollback.txt文档中。
10. 查询表account的所有行，将结果拷贝入rollback.txt文档中。
11. 提交事务。
12. 尝试回滚刚刚提交的事务。查询表中存储的值。解释说明事务提交和回滚产生的不同影响。将结果写入rollback.txt文档中。

**第二部分 测试事务隔离等级**

PostgreSQL支持两种事务隔离等级。完成以下练习，熟悉PostgreSql支持的隔离等级。

* 1. 运行[SQL script](mhtml:mk:@MSITStore:D:\job\ssd7\SSD7带练习版本.chm::/DOssd7/Exercise/9.mht!https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/build_trans_db.sql)建立一个小的银行数据库，account关系包含两列，用户id和账户余额。有两行数据，定期储蓄账户（id为1）和活期储蓄账户（id为2）。
  2. 开始两个PostgreSQL窗口会话。
  3. 两个窗口都开始一个新事务。
  4. 第一个窗口，将活期储蓄账户余额更改为455.66。
  5. 第一个窗口，查询account关系所有行，结果拷贝入isolation.txt文档中，用序号标识出数据来自于窗口一。
  6. 第二个窗口，查询account关系所有行，结果拷贝入isolation.txt文档中，用序号标识出数据来自于窗口二。
  7. 第一个窗口，提交事务。
  8. 第二个窗口，查询account表所有行，结果拷贝入isolation.txt文档中，用序号标识出数据来自于窗口二。窗口一的数据更新是否生效？为什么？将解释部分写入isolation.txt文档中。
  9. 第二个窗口，提交事务。
  10. 两个窗口都开始一个新事务。
  11. 第二个窗口设置事务隔离等级为串行化，将SQL语句写入isolation.txt文档中。
  12. 第一个窗口，设置活期储蓄账户余额为1400。
  13. 第二个窗口，设置定期储蓄账户余额为1。
  14. 在两个窗口内查询account关系的所有行。用序号标识出数据来源于窗口一或二。
  15. 第一个窗口，提交事务。
  16. 第二个窗口，查询account关系所有行。活期储蓄账户值是否更新？为什么？第一个窗口的定期储蓄账户值是否更新？为什么？将解释语句写入isolation.txt文档中。
  17. 第二个窗口提交事务。在两个窗口查询数据。将结果拷贝入isolation.txt文档中，用序号标识出数据来源，并解释产生为何会产生此结果。将答案拷贝入isolation.txt文档中。

**第三部分 阻塞和死锁**

1.运行[SQL script](mhtml:mk:@MSITStore:D:\job\ssd7\SSD7带练习版本.chm::/DOssd7/Exercise/9.mht!https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/build_trans_db.sql)建立一个小的银行数据库，account关系包含两列，用户id和账户余额。有两行数据，定期储蓄账户（id为1）和活期储蓄账户（id为2）。

2.开始两个PostgreSQL窗口会话。

3.两个窗口都开始一个新事务。

4.第二个窗口中将活期储蓄账户余额更新为455.75.

5.第一个窗口中将活期储蓄账户余额更新为1400，更新能否成功，解释说明，将答案写入deadlock.txt文档中。

6.第二个窗口提交事务，观察第一个窗口变化，解释说明，答案写入deadlock.txt文档中。

7.第一个窗口提交事务。

8.两个窗口开始一个新事务。

9.第二个窗口中将定期储蓄账户余额更新为2400.

10.第一个窗口中将活期储蓄账户余额更新为2000.

11.第一个窗口中将定期储蓄账户余额更新为1400.

12.第二个窗口中将活期储蓄账户余额更新为1000.50.

13.观察结果，试着解释原因。结果写入deadlock.txt文档中。

14.在每个窗口中查询表的数据，你观察到了什么？试做说明，将答案写入deadlock.txt文档中。

# 附录四 应用程序

最后一部分练习将检测你在实际条件下对事务的应用。

1.假设William和Julie公用一个银行账户。银行有许多自动取款机，两人同时在两台取款机上进行作业。William希望先在定期账户中存入4000美金，然后将活期账户中的3000美金转账到定期账户中。Julie希望先在活期账户中存入5000美金，然后从定期账户中取走1000美金。

**存款操作顺序为：**

1. 查询当前账户余额。
2. 将存款金额增加到账户余额中。
3. 更新新的账户余额到数据库中。

**取款操作顺序为：**

1. 查询当前账户余额。
2. 将取款金额从余额中扣除。
3. 更新新的账户余额到数据库中。

转账操作顺序为：

1. 查询转出账户余额。
2. 将转账金额从转出余额中扣除。
3. 查询转入账户余额。
4. 将转账金额增加到转入账户中。
5. 将两个新的账户余额更新到数据库中。

假设用户操作和数据库事务执行可以交叉操作。接下来将提供几个操作实例，每个实例相对前一个实例都解决了一个或多个问题，但是却仍然存在问题。你需要指出解决了那些问题，并解释为何如此修改。注意，在实例中，带点的操作为数据库操作，不带点的为ATM操作。假设数据库系统使用PostgreSQL，你只能使用PostgreSQL支持的隔离等级。假设除了这些操作没有其他事务操作，因此你没必要考虑其他啊事务对于数据的潜在影响，同时每个例子中的操作都是一致协调的。你不能假设William所有的事务在Julie之前或之后，也不能假设两个人的操作一定是交叉的。Julie的所有操作在William之前或之后或交叉进行都是可能的。

1. Scenario A

|  |  |
| --- | --- |
| William | Julie |
| * + - Begin transaction (read committed isolation level) | * + - Begin transaction (read committed isolation level) |
| ATM prompts user for operation. | ATM prompts user for operation. |
| William Chooses Deposit into Savings. | Julie Chooses Deposit into Checking. |
| ATM prompts user for amount. | ATM prompts user for amount. |
| William enters 400. | Julie enters 500. |
| * + - Select balance of savings | * + - Select balance of checking |
| savings\_balance = savings\_balance + $400 | checking\_balance = checking\_balance + $500 |
| * + - Update savings balance in database. | * + - Update checking balance in database. |
| ATM displays confirmation of deposit. | ATM displays confirmation of deposit. |
| ATM prompts user for operation. | ATM prompts user for operation. |
| William chooses transfer from checking to savings. | Julie chooses withdrawal from savings. |
| ATM prompts user for amount. | ATM prompts user for amount. |
| William enters 300. | Julie enters 100. |
| * + - Select the checking balance. | * + - Select the savings checking balance in database. |
| checking\_balance = checking\_balance - $300 | savings\_balance = savings\_balance - $100 |
| * + - Select the savings balance. | * + - Update savings balance in database. |
| savings\_balance = savings\_balance + $300 | ATM displays confirmation of withdrawal. |
| * + - Update savings balance in database. | ATM prompts user for operation. |
| * + - Update checking balance in database. | Julie chooses no more operations. |
| ATM displays confirmation of transfer. | * + - End Transaction |
| ATM prompts user for operation. |  |
| William chooses no more operations. |  |
| * + - End Transaction |  |

序列1中存在以下问题：

1. 不正确的隔离等级
2. 确认信息位置不正确
3. 不正确的事务边界
4. 在事务边界内等待用户输入

对于以上每个问题，解释为何不正确，题目将提供一个例子，请完成剩余部分。

确认信息位置不正确，如用户可能在事务没有结束前就收到了操作完成的确认信息。当事务发生回滚后，事务将会被撤销，但是用户以为已经执行成功，会存在问题。所以，所有的确认信息必须在事务成功提交后进行显示。

1. Scenario B

|  |  |
| --- | --- |
| William | Julie |
| ATM prompts user for operation. | ATM prompts user for operation. |
| William Chooses Deposit into Savings. | Julie Chooses Deposit into Checking. |
| * + 1. Begin transaction (read committed isolation level) | * + 1. Begin transaction (read committed isolation level) |
| ATM prompts user for amount. | ATM prompts user for amount. |
| William enters 400. | Julie enters 500. |
| * + 1. Select balance of savings | * + 1. Select balance of checking |
| savings\_balance = savings\_balance + $400 | checking\_balance = checking\_balance + $500 |
| * + 1. Update savings balance in database. | * + 1. Update checking balance in database. |
|  |  |
| * + 1. End Transaction | * + 1. End Transaction |
| ATM displays confirmation of deposit. | ATM displays confirmation of deposit. |
| ATM prompts user for operation. | ATM prompts user for operation. |
| William chooses transfer from checking to savings. | Julie chooses withdrawal from savings. |
| * + 1. Begin transaction (read committed isolation level) | * + 1. Begin transaction (read committed isolation level) |
| ATM prompts user for amount. | ATM prompts user for amount. |
| William enters 300. | Julie enters 100. |
| * + 1. Select the checking balance. | * + 1. Select the savings checking balance in database. |
| checking\_balance = checking\_balance - $300 | savings\_balance = savings\_balance - $100 |
| * + 1. Select the savings balance. | * + 1. Update savings balance in database. |
| savings\_balance = savings\_balance + $300 | * + 1. End Transaction |
| * + 1. Update savings balance in database. | ATM displays confirmation of withdrawal. |
| * + 1. Update checking balance in database. | ATM prompts user for operation. |
| * + 1. End Transaction | Julie chooses no more operations. |
| ATM displays confirmation of transfer. |  |
| ATM prompts user for operation. |  |
| William chooses no more operations. |  |

* 1. Scenario C

|  |  |
| --- | --- |
| William | Julie |
| ATM prompts user for operation. | ATM prompts user for operation. |
| William Chooses Deposit into Savings. | Julie Chooses Deposit into Checking. |
| * + 1. Begin transaction (serializable isolation level) | * + 1. Begin transaction (serializable isolation level) |
| ATM prompts user for amount. | ATM prompts user for amount. |
| William enters 400. | Julie enters 500. |
| * + 1. Select balance of savings | * + 1. Select balance of checking |
| savings\_balance = savings\_balance + $400 | checking\_balance = checking\_balance + $500 |
| * + 1. Update savings balance in database. | * + 1. Update checking balance in database. |
|  |  |
| * + 1. End Transaction | * + 1. End Transaction |
| ATM displays confirmation of deposit. | ATM displays confirmation of deposit. |
| ATM prompts user for operation. | ATM prompts user for operation. |
| William chooses transfer from checking to savings. | Julie chooses withdrawal from savings. |
| * + 1. Begin transaction (serializable isolation level) | * + 1. Begin transaction (serializable isolation level) |
| ATM prompts user for amount. | ATM prompts user for amount. |
| William enters 300. | Julie enters 100. |
| * + 1. Select the checking balance. | * + 1. Select the savings checking balance in database. |
| checking\_balance = checking\_balance - $300 | savings\_balance = savings\_balance - $100 |
| * + 1. Select the savings balance. | * + 1. Update savings balance in database. |
| savings\_balance = savings\_balance + $300 | * + 1. End Transaction |
| * + 1. Update savings balance in database. | ATM displays confirmation of withdrawal. |
| * + 1. Update checking balance in database. | ATM prompts user for operation. |
| * + 1. End Transaction | Julie chooses no more operations. |
| ATM displays confirmation of transfer. |  |
| ATM prompts user for operation. |  |
| William chooses no more operations. |  |

你最终需要写出正确的顺序，以满足最大的并发操作。写出William和Julie的每个操作步骤。同时写出DBMS和ATM的操作。

2.假设你需要为电影院编写一个在线售票系统，用户量较大。假设用户只能在电影开始前30分钟才可以买票，因此会有大量的用户进行并发操作，一个用户的操作顺序如下：

1）系统查询并显示所有电影。

2）用户选择喜欢观看的电影。

3）系统查询并显示用户选择电影的所有场次，用户选择相应的场次。

4）系统查询剩余票数。

5）如果有剩余票，系统显示剩余票数。

6）用户输入购票张数。

7）用户确认订单并付款。

以下给定了一些测试数据，在PostgreSQL中创建一个 新的数据库movie，运行[SQL script](https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/create_movie.sql)创建以下表格：

|  |  |
| --- | --- |
| Movies | |
| Movie Id (PK) | Movie Title |
| 001 | *The Terminator* |
| 002 | *James Bond* |
| 003 | *The Matrix* |

|  |  |  |  |
| --- | --- | --- | --- |
| Movie Showings | | | |
| Movie Id (PK,FK) | Movie Time(PK) | Total Seats | Available Standard Seats |
| 001 | 1:00 PM | 40 | 25 |
| 001 | 4:00 PM | 30 | 27 |
| 002 | 12:30 PM | 50 | 32 |
| 002 | 7:30 PM | 75 | 17 |
| 003 | 9:30 PM | 100 | 10 |

你需要修改[Java console application provided](https://www.icarnegie.com/content/SSD/SSD7/1.5.1/normal/pg-performance/pg-transactionmanagement/qn-pr-transaction/qn-pr-transaction/handout/MovieTransaction.java)提供的程序，测试环境为PostgreSQL，只能使用其支持的隔离等级。应用程序使用了不正确的事务边界和隔离等级。你的任务是在最大话满足用户的前提下设置正确的事务边界。注意，你可以在程序中使用多个事务。对于每个事务，你需要包括：

1）解释事务何时开始，何时结束。

2）解释为何如此设置事务边界。

3）解释为何你的隔离等级能够确保所有用户看到一致的数据视图。