Without Transaction

- **Atomicity**
  Insert an abort in the program after incrementing the amount in the receiver’s account.

  Use Demo – atomicity.c

  **Problem:**
  After the program is executed the receiver’s account will be incremented. However, the sender’s account will not be decremented.

- **Serializability**
  Insert a break point and run the program in Debug mode till the break point. Now run a query to update the sender’s balance via phpPgAdmin. Then run the rest of the program

  Use Demo - base program.c

  if (balance >= amount) {
      -- Break Point
      increment_recv_account();
      decrement_sender_account();
  }

  And execute the below in phpPgAdmin:
  UPDATE account SET balance = balance – 100
  WHERE name LIKE ‘%Alice%’

  **Problem:**
  Alice’s account will have a negative balance since our program checked the available balance before the other query updated it.

By default (without BEGIN), PostgreSQL executes transactions in "autocommit" mode, that is,
- Each statement is executed in its own transaction and
- A commit is implicitly performed at the end of the statement (if execution was successful, otherwise a rollback is done).
BEGIN and END transaction

- **Atomicity - problem solved.**

  Use Demo - BEGIN and END.c
  Insert an abort in the program after incrementing the amount in the receiver’s account.

  **Result:** The receiver’s account is not updated if the program aborts or the transaction fails.

- **Introduce COMMIT or ROLLBACK**

  Use Demo - COMMIT and ROLLBACK.c
  Instead of an ABORT you can decide to rollback the changes based on a condition.

  **Result:** If the sender’s account balance has changed after reading in the program, rollback the changes.

BEGIN initiates a transaction block, that is, all statements after a BEGIN command will be executed in a single transaction until an explicit COMMIT or ROLLBACK is given.

- **Serializable – problem not yet solved.**

  Insert a break point and run the program in Debug mode till the break point. Now run a query to update the sender’s balance via phpPgAdmin. Then run the rest of the program

  Use Demo - Demo - BEGIN and END.c

  ```
  if (balance >= amount) {   -- Break Point
    increment_recv_account();
    decrement_sender_account();
  }
  ```

  And execute the below in phpPgAdmin:

  ```
  UPDATE account SET balance = balance – 100
  WHERE name LIKE ‘%Alice%’
  ```

  **Problem:**
  We can still update the receiver’s account via phpPgAdmin while our program is running. Our changes are within a transaction. However, we are letting other transactions to update the same tuples.
ISOLATION LEVEL and LOCKING

- **Introduce ISOLATION LEVEL SERIALIZABLE**
  Serializable – If a transaction T is running at level serializable, then the execution of T must appear as if all other transactions run either entirely before or entirely after T.

  Use Demo – SERIALIZABLE.c

  Insert a break point and run the program in Debug mode till the break point. Now run a query to update the sender’s balance via phpPgAdmin. Then run the rest of the program. The changes are rolled back and you get a Warning message.

- **Introduce LOCKING**

  To acquire an exclusive row-level lock on a row without actually modifying the row, select the row with SELECT FOR UPDATE.

  Use Demo – Locking.c

  Insert a break point and run the program in Debug mode till the break point. Now run a query to update the sender’s balance via phpPgAdmin. Then run the rest of the program.

  **Result:** The transaction in phpPgAdmin does not execute till the program release a lock.

References:
- [http://www.postgresql.org/docs/8.0/static/sql-begin.html](http://www.postgresql.org/docs/8.0/static/sql-begin.html)
- [http://www.postgresql.org/docs/8.0/static/sql-start-transaction.html](http://www.postgresql.org/docs/8.0/static/sql-start-transaction.html)