Transactions

To execute a transaction with node-postgres you simply execute `BEGIN / COMMIT / ROLLBACK` queries yourself through a clie
node-postgres strives to be low level an un-opinionated it doesn't provide any higher level abstractions specifically around tr

> You **must** use the *same* client instance for all statements within a transaction. PostgreSQL isolates a transaction to individu
> This means if you initialize or use transactions with the `pool.query` method you **will** have problems. Do not use transacti
> `pool.query` .

Examples

A pooled client with callbacks

```
const { Pool } = require('client')
const pool = new Pool()

pool.connect((err, client, done) => {

  const shouldAbort = (err) => {
    if (err) {
      console.error('Error in transaction', err.stack)
      client.query('ROLLBACK', (err) => {
        if (err) {
          console.error('Error rolling back client', err.stack)
        }
        // release the client back to the pool
        done()
      })
    }
    return !!err
  }

  client.query('BEGIN', (err) => {
    if (shouldAbort(err)) return
    client.query('INSERT INTO users(name) VALUES($1) RETURNING id', ['brianc'], (err, res) => {
      if (shouldAbort(err)) return

      const insertPhotoText = 'INSERT INTO photos(user_id, photo_url) VALUES ($1, $2)'
      const insertPhotoValues = [res.rows[0].id, 's3.bucket.foo']
      client.query(insertPhotoText, insertPhotoValues, (err, res) => {
        if (shouldAbort(err)) return

        client.query('COMMIT', (err) => {
          if (err) {
            console.error('Error committing transaction', err.stack)
          }
          done()
        })
      })
    })
  })
})
```

I omitted any additional libraries from the example for clarity, but if you're using callbacks you'd typically be using a flow c library like async.

A pooled client with async/await

Things are considerably more straightforward if you're using async/await:

```
const { Pool } = require('pg')
const pool = new Pool()

(async () => {
  // note: we don't try/catch this because if connecting throws an exception
  // we don't need to dispose of the client (it will be undefined)
  const client = await pool.connect()

  try {
    await client.query('BEGIN')
    const { rows } = await client.query('INSERT INTO users(name) VALUES($1) RETURNING id', ['br

    const insertPhotoText = 'INSERT INTO photos(user_id, photo_url) VALUES ($1, $2)'
    const insertPhotoValues = [res.rows[0].id, 's3.bucket.foo']
    await client.query(insertPhotoText, insertPhotoValues)
    await client.query('COMMIT')
  } catch (e) {
    await client.query('ROLLBACK')
    throw e
  } finally {
    client.release()
  }
})().catch(e => console.error(e.stack))
```