Transactions

Transactions allow you to support multiple users safely.

You need to use transactions when there are several users (or processes) reading and writing to the database at the same time.

This example transfers money from one account to another. You will see how:

- A simple program to move money from one account to another can lead to money being created or destroyed; thus undermining capitalism and ushering in a financial apocalypse.
- By using transactions we can ensure that each transfer either completely succeeds or completely fails.
- By using transactions we can detect failures.
- By retrying transactions we can ensure that all transactions are completed eventually.

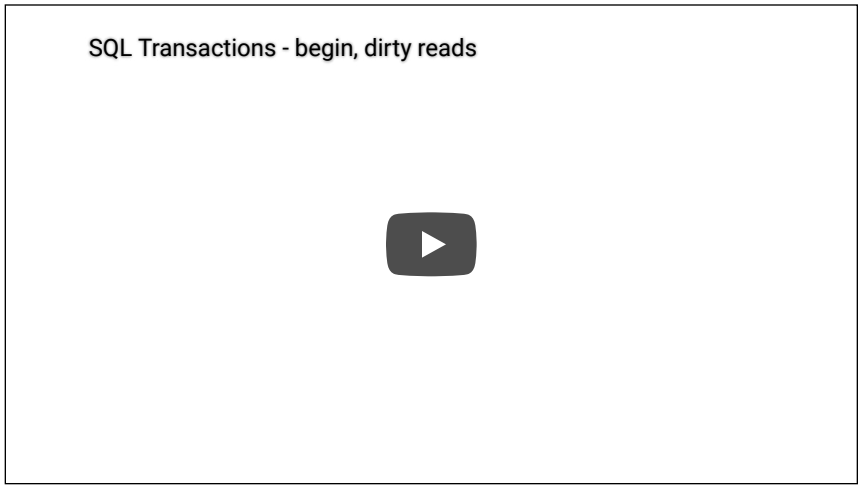
Contents

BEGIN

- Two sessions, with and without transactions
- The second session cannot see changes until they have been committed by the first session (to do otherwise would be a "dirty read").
- We create the bank of wealth - it has two customers, they both have £100

```
+--------+--------+
| cust   | amount |
+--------+--------+
| andrew |    100 |
| brian  |    100 |
+--------+--------+
```

```
DROP TABLE wealth;
CREATE TABLE wealth(
  cust VARCHAR(10) PRIMARY KEY,
  amount INT
);
INSERT INTO wealth VALUES ('andrew',100);
INSERT INTO wealth VALUES ('brian', 100);
```

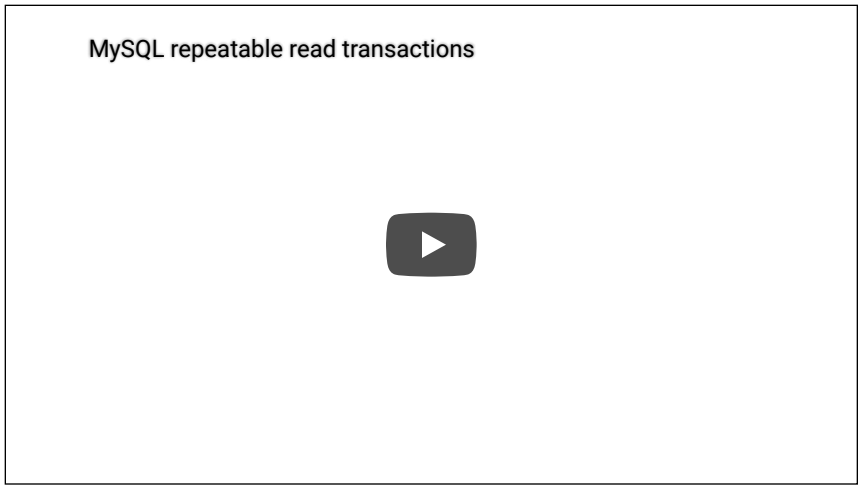


REPEATABLE READ

- When a session starts a transaction it has a consistent copy of the database that is isolated from activity in other sessions.
- If we are at "REPEATABLE READ" level then even committed transactions (from a different session) are not visible.

```
Pink session                  White Session
BEGIN;
                              SELECT amount FROM wealth WHERE cust='andrew';
                              --> 100
                              BEGIN;
```

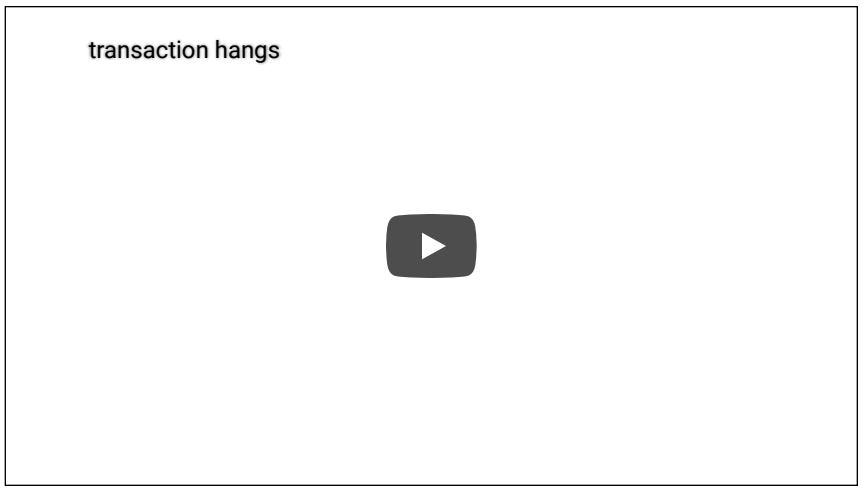
```
UPDATE wealth SET amount=200
 WHERE cust = 'andrew';
                                SELECT amount FROM wealth WHERE cust='andrew';
                                --> 100
COMMIT;
                                SELECT amount FROM wealth WHERE cust='andrew';
                                --> 100
                                COMMIT;
                                SELECT amount FROM wealth WHERE cust='andrew';
                                --> 200
```



Transaction left hanging

- Two transactions attempt to increase andrew's wealth
- One attempts to add £10 the other attempts to add £20
- After both transactions succeed andrew will be £30 richer.

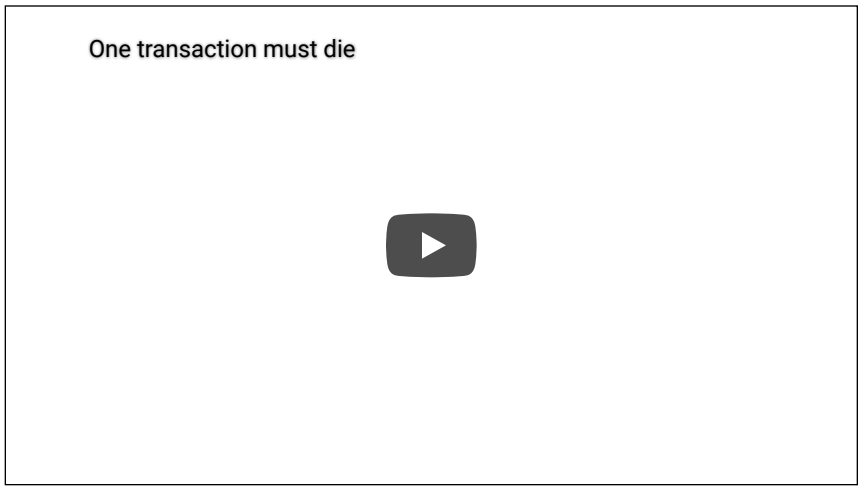
```
Pink session                    White Session
BEGIN;
                                BEGIN;
UPDATE wealth
   SET amount=amount+10
 WHERE cust = 'andrew';
                                UPDATE wealth
                                   SET amount=amount+20
                                 WHERE cust = 'andrew';
                                --This session hangs; its fate rests on the other session
                                --If the other session does rollback andrew will have £120
                                --If the other session does commits  andrew will have £130
COMMIT;
                                SELECT * FROM wealth WHERE cust='andrew';
                                +--------+--------+
                                | cust   | amount |
                                +--------+--------+
                                | andrew |    130 |
                                +--------+--------+
```



One transaction must die

- We try the same pair of transactions as above but this time we do a read first.

```
Pink session                    White Session
BEGIN;
                                BEGIN;
                                SELECT * FROM wealth WHERE cust='andrew';
                                +--------+--------+
                                | cust   | amount |
                                +--------+--------+
                                | andrew |    130 |
                                +--------+--------+
UPDATE wealth
   SET amount=amount+10
 WHERE cust = 'andrew';
--Hangs
                                UPDATE wealth
                                   SET amount=amount+20
                                 WHERE cust = 'andrew';
--Fails                         --Succeeds
                                COMMIT;
                                SELECT * FROM wealth WHERE cust='andrew';
                                +--------+--------+
                                | cust   | amount |
                                +--------+--------+
                                | andrew |    150 |
                                +--------+--------+
```



Update Madness

We will try a naive update to see how badly things can go wrong. We will then try to fix it so that the errors are more manageable.

We create the wealth table and add two rows:

```
DROP TABLE wealth;
CREATE TABLE wealth(
  cust VARCHAR(10) PRIMARY KEY,
  amount INT
);
INSERT INTO wealth VALUES ('andrew',100);
INSERT INTO wealth VALUES ('brian', 100);
```

We create a php program `trnt.php` to transfer money from andrew to brian or the other way around.

```
<?php
$payer = $argv[1];
$payee = $argv[2];
$value = $argv[3];
$dbh = new PDO('mysql:host=localhost;dbname=scott','scott','tiger');
$dbh->setAttribute(PDO::ATTR_EMULATE_PREPARES,false);
$sth = $dbh->prepare("SELECT amount FROM wealth WHERE cust=?");
$sth->execute(array($payer));
$a = $sth->fetchAll()[0][0];
$sth->execute(array($payee));
$b = $sth->fetchAll()[0][0];
$sth = $dbh->prepare("UPDATE wealth SET amount=? WHERE cust=?");
$sth->execute(array($a-$value,$payer));
$sth->execute(array($b+$value,$payee));
echo "$payer has ".($a-$value).", $payee has ".($b+$value)."\n";
```

This can be run from the command line - you give three parameters - payer, payee and amount.

This Linux bash script command gives one pound from andrew to brian:

```
php trnt.php andrew brian 1
```

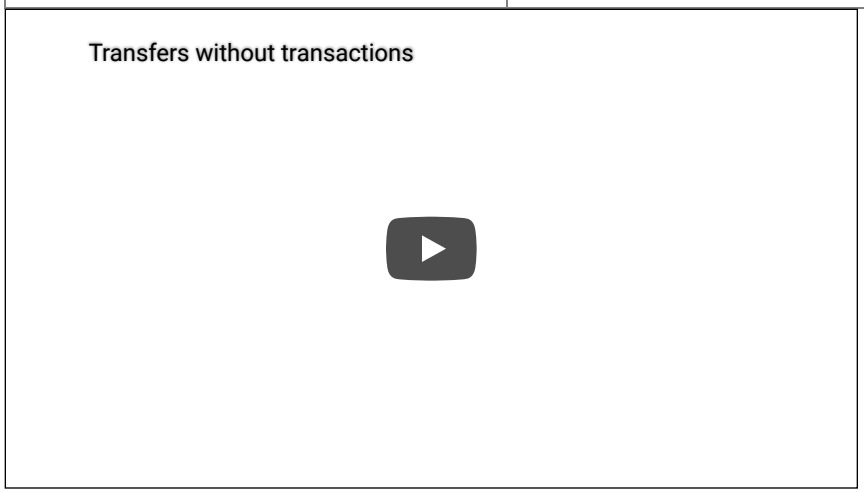
This is the Linux bash script to run the command 100 times:

```
for i in `seq 100`; do php trnt.php andrew brian 1; done
```

You will find that if you run these two commands in different putty sessions at the same time then serious errors can occur:

```
for i in `seq 100`; do php trnt.php andrew brian 1; done
```

```
for i in `seq 100`; do php trnt.php andrew brian -1; done
```



Update Consistency - detecting failure

This version of the program includes transactions.

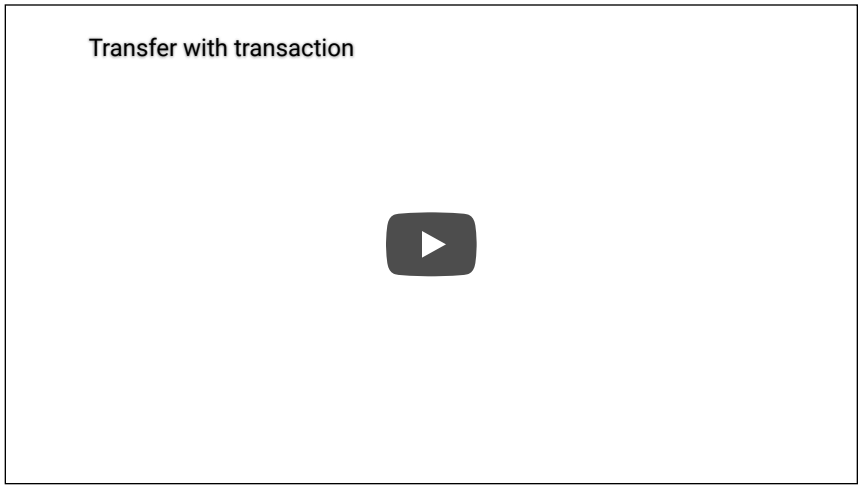
- Before running any SQL read or write statements we BEGIN a transaction.
- After all of the SQL statements have been run we issue a COMMIT
- Every SQL statement that runs is checked - if a failure is detected we show an error message and abandon the program.

The php program `trwt.php` to transfer money now includes transactions and failure detection.

```
<?php
$payer = $argv[1];
$payee = $argv[2];
$value = $argv[3];
$dbh = new PDO('mysql:host=localhost;dbname=scott','scott','tiger');
$dbh->setAttribute(PDO::ATTR_EMULATE_PREPARES,false);
$dbh->query("SET TRANSACTION ISOLATION LEVEL SERIALIZABLE");
$dbh->beginTransaction();
$sth = $dbh->prepare("SELECT amount FROM wealth WHERE cust=?");
$comm = "php ".join(' ',$argv)."\n";
if (!$sth->execute(array($payer))){
  die("#SELECT failed\n$comm");
}
$a = $sth->fetchAll()[0][0];
if (!$sth->execute(array($payee))){
  die("#SELECT failed\n$comm");
}
$b = $sth->fetchAll()[0][0];
$sth = $dbh->prepare("UPDATE wealth SET amount=? WHERE cust=?");
if (!$sth->execute(array($a-$value,$payer))){
  $dbh->rollback();
  die("#UPDATE failed\n$comm");
}
if (!$sth->execute(array($b+$value,$payee))){
  $dbh->rollback();
  die("#UPDATE failed\n$comm");
}
$dbh->commit();
echo "#$payer has ".($a-$value).", $payee has ".($b+$value)."\n";
```

It is still possible for a transaction to fail - but this time the transaction either completes or fails entirely. It is no longer possible for money to go missing from the system altogether.

When a transaction does fail we are notified and we can take steps to cope with that - typically we can try again.



Update Consistency - retry on failure

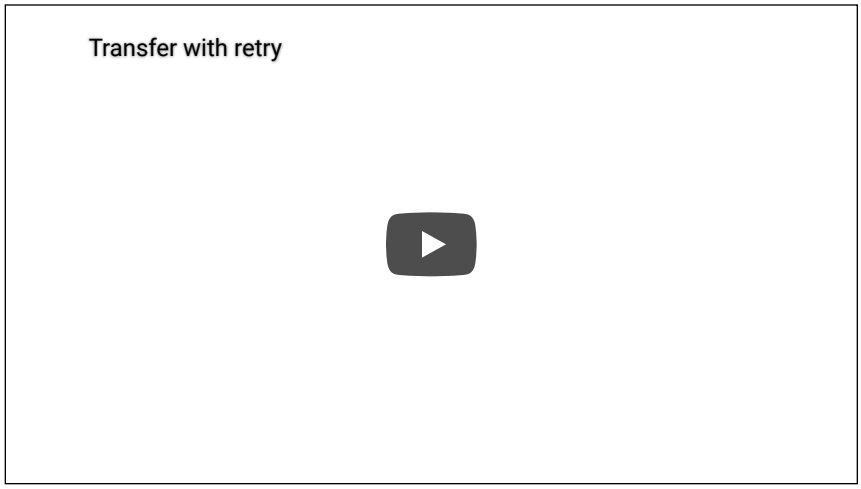
The final version of the program includes a retry on failure

- As before we enclose all the SQL execute attempts with a transaction; BEGIN at the top COMMIT at the end
- If any of the four execute statements fails then we simply reissue the command - we try again.
- So long as failures are rare this is a safe thing to do.
  - Our previous test showed one or two failures per hundred transactions
  - This is a terrifyingly high number of failures - in a practical application we would hope that failures were much more unusual.
  - Even with a 2% failure rate the retry will not hugely increase the total run time.

The php program `trwr.php` to transfer money now includes transactions and retry in failure.

```
<?php
$payer = $argv[1];
$payee = $argv[2];
$value = $argv[3];
$dbh = new PDO('mysql:host=localhost;dbname=scott','scott','tiger');
$dbh->query("SET TRANSACTION ISOLATION LEVEL SERIALIZABLE");
$dbh->beginTransaction();
$sth = $dbh->prepare("SELECT amount FROM wealth WHERE cust=?");
$comm = "php ".join(' ',$argv)."\n";
if (!$sth->execute(array($payer))){
  $dbh->rollback();
  `$comm`;
  die("#SELECT failed\n");
}
$a = $sth->fetchAll()[0][0];
if (!$sth->execute(array($payee))){
  $dbh->rollback();
  `$comm`;
  die("#SELECT failed\n");
}
$b = $sth->fetchAll()[0][0];
$sth = $dbh->prepare("UPDATE wealth SET amount=? WHERE cust=?");
if (!$sth->execute(array($a-$value,$payer))){
  $dbh->rollback();
  `$comm`;
  die("#UPDATE failed\n");
}
if (!$sth->execute(array($b+$value,$payee))){
  $dbh->rollback();
  `$comm`;
  die("#UPDATE failed\n");
}
$dbh->commit();
echo "#$payer has ".($a-$value).", $payee has ".($b+$value)."\n";
```

This time when a failure occurs we simply try again - nothing can possibly go wrong!



Challenge

You can solve this problem without explicitly starting a transaction.

- You need to know that a single SQL statement will always complete atomically - it will either succeed or fail completely.
- A single UPDATE command can change more than one row, for example the following will increase the wealth of two people:

```
UPDATE wealth SET amount = amount+1
 WHERE cust IN ('andrew','brian');
```

- You can use a CASE statement in an UPDATE. This statement will give 'andrew' £1,000,000 (it will also set everyone else's account to NULL; what's not to like?).

```
UPDATE wealth
   SET amount = CASE WHEN cust='andrew' THEN 1000000 END;
```

Can you adapt the php code to transfer $value from $payer to $payee in one single SQL statement? There is still the possibility that this transaction will fail but you are unlikely to witness such an event.

Have a go at using transactions to fix a problem yourself: Transactions Airline

---

Retrieved from "http://napier.sqlzoo.net/w/index.php?title=Transactions&oldid=39670"

This page was last edited on 16 January 2020, at 15:30.