welcome to production

Graeme Foster

“Spanish Software” Developer

5 years ago, I co-designed, and built a MONOLITH

We based in on the **buzzwords** and ideas of the time like *ORMs*, *DDD*, and *web services*.

Like proud parents, we released it into the wild in 2011

System Architecture

Fat Client (WPF)

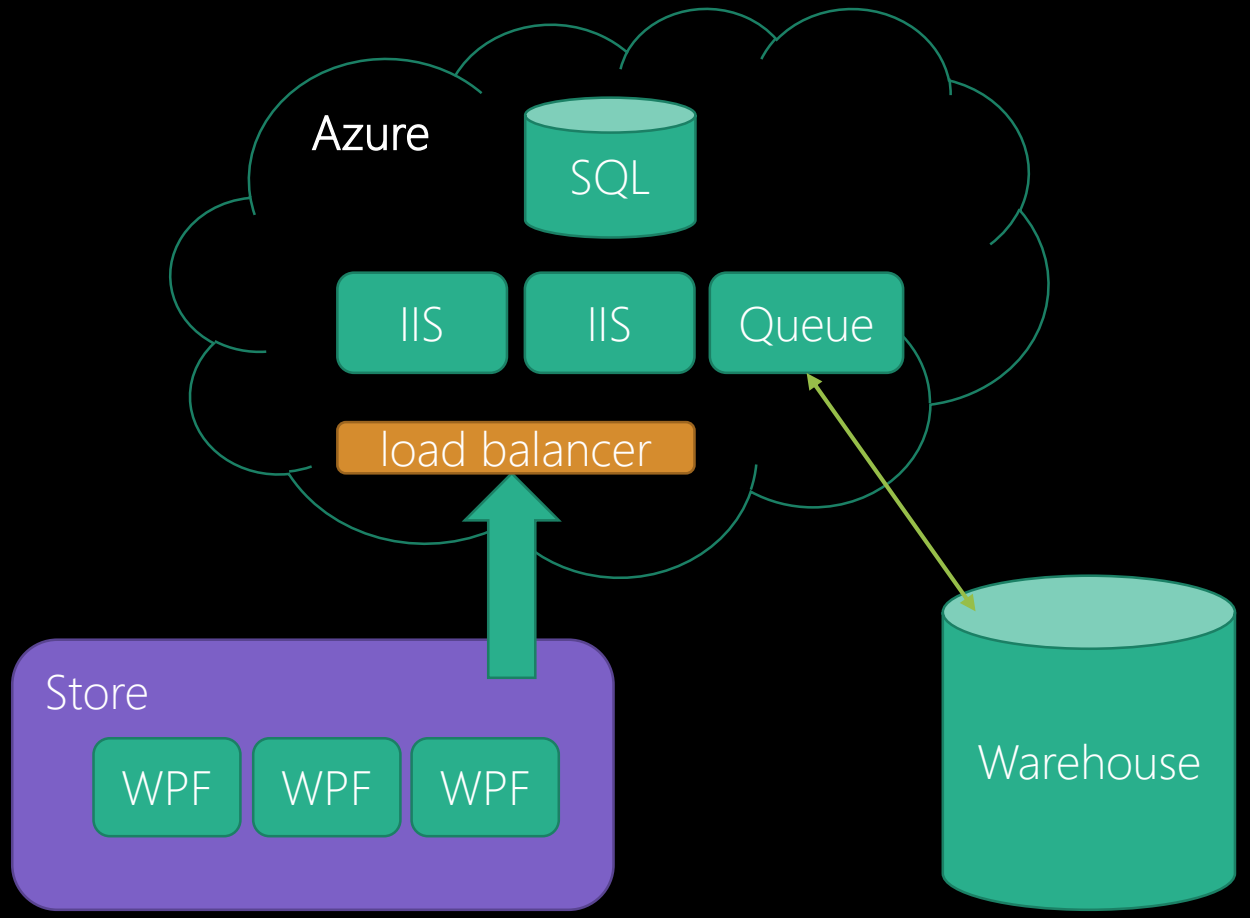
Web Services via WCF, aka Death-*

Object Oriented Domain

Some DDD concepts but not the important ones

ORM (NHibernate) & a SQL database

Warehouse populated via audit-trail

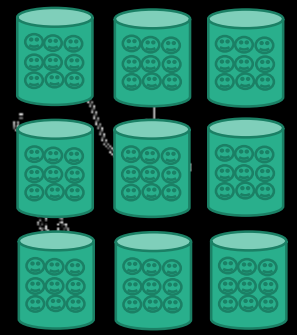


2011 - 2012

Single Database Per Store
Small number of users
Hosted locally

2012 - 2013

Move to Azure
(Singapore)



Issues begin to appear

Performance

Scaling

Concurrency

How will we diagnose issues?

- Log Files
- Application Profilers
- Network Profilers
- SQL profiling tools
- Intuition

Dear Diary…

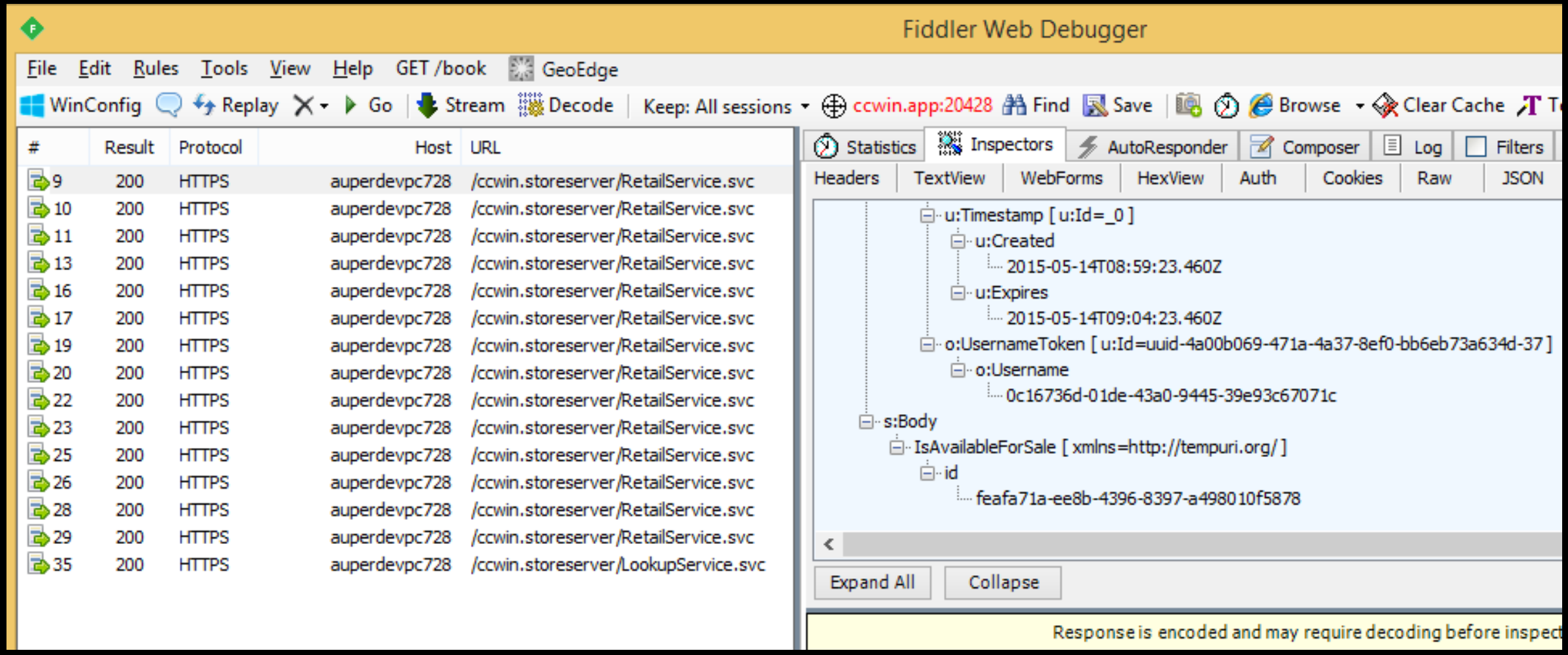
The following incidents are based on real world incidents that occurred to this system. Name have been altered to protect the innocent!!

Support

"users are struggling to complete large sales"

Server Log files *Fast operation*

As seen by Fiddler



The Chatter-Box Api

Caused by looped service calls in code

```
class SaleViewModel
{
        public void OnFinishSale(SaleRequest request)
        {
                DoItemPreCheck(request.Items);
        }

        private void DoItemPreCheck(Items[] items)
        {
                if (items.Any(!stockService.IsAvailableForSale))
                {
                        ShowCannotCompleteSaleMessage(....);
                }
        }
}
```

Why did it happen?

- The ping time from a terminal to Singapore is about 350ms
- Calling a service multiple times serially, on the main UI thread, caused the application to lock
- Most of the time was due to network latency

Resolution

- We “own” the producers and consumers of the API.
- It’s simple for us to make the interface much less chatty.
- Don’t use chatty api’s over the network

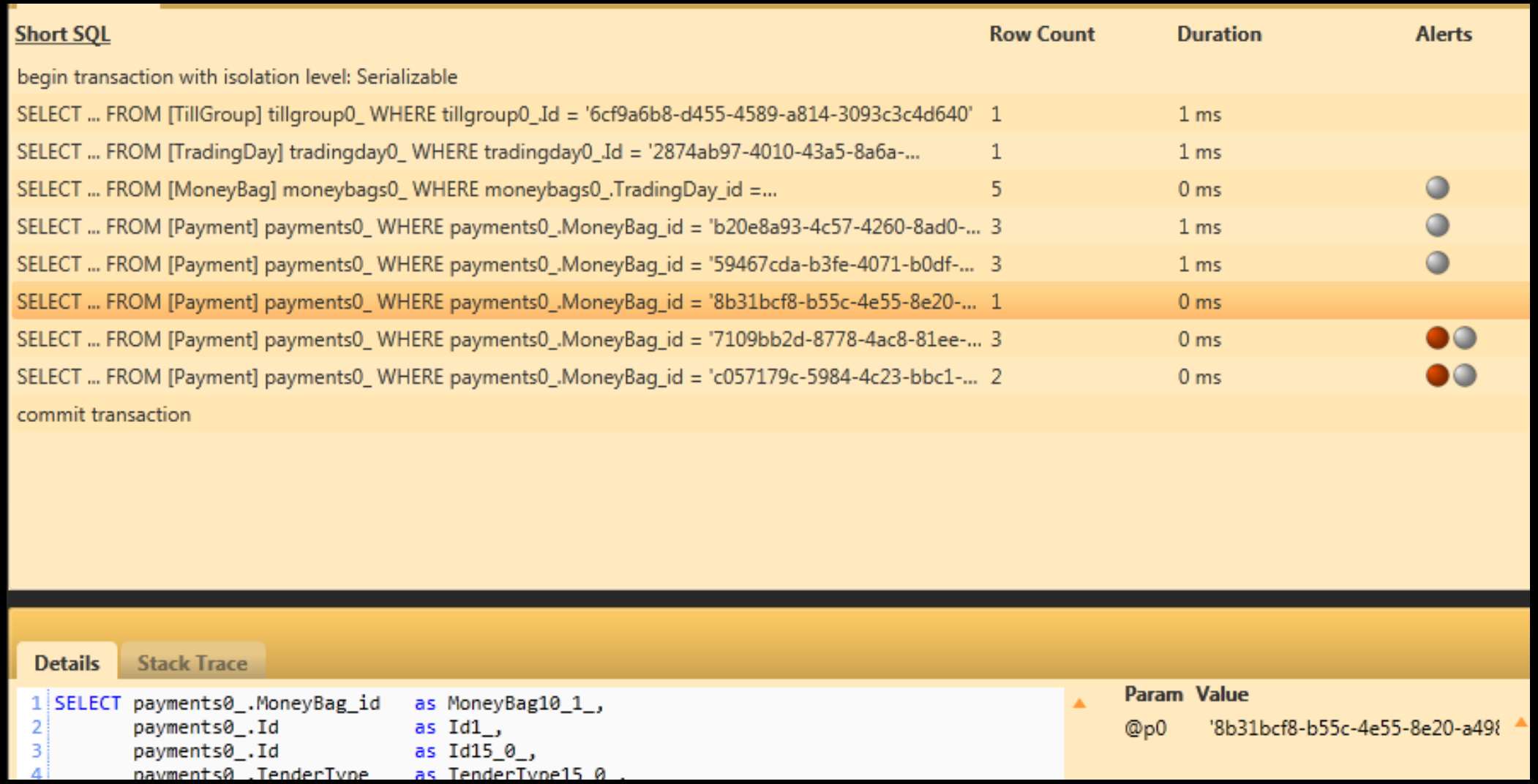
Support

"Balancing tills is taking a long time"

Network Profiler — *Single service call from terminal*

Application Profiler — *Slow*

As seen by the profiler



Short SQL	Row Count	Duration	Alerts
begin transaction with isolation level: Serializable			
SELECT ... FROM [TillGroup] tillgroup0_ WHERE tillgroup0_.Id = '6cf9a6b8-d455-4589-a814-3093c3c4d640'	1	1 ms	
SELECT ... FROM [TradingDay] tradingday0_ WHERE tradingday0_.Id = '2874ab97-4010-43a5-8a6a-...	1	1 ms	
SELECT ... FROM [MoneyBag] moneybags0_ WHERE moneybags0_.TradingDay_id =...	5	0 ms	
SELECT ... FROM [Payment] payments0_ WHERE payments0_.MoneyBag_id = 'b20e8a93-4c57-4260-8ad0-...	3	1 ms	
SELECT ... FROM [Payment] payments0_ WHERE payments0_.MoneyBag_id = '59467cda-b3fe-4071-b0df-...	3	1 ms	
SELECT ... FROM [Payment] payments0_ WHERE payments0_.MoneyBag_id = '8b31bcf8-b55c-4e55-8e20-...	1	0 ms	
SELECT ... FROM [Payment] payments0_ WHERE payments0_.MoneyBag_id = '7109bb2d-8778-4ac8-81ee-...	3	0 ms	
SELECT ... FROM [Payment] payments0_ WHERE payments0_.MoneyBag_id = 'c057179c-5984-4c23-bbc1-...	2	0 ms	
commit transaction			

Details | Stack Trace

```
1 SELECT payments0_.MoneyBag_id   as MoneyBag10_1_,
2        payments0_.Id            as Id1_,
3        payments0_.Id            as Id15_0_,
4        payments0_.TenderType    as TenderType15_0_,
```

Param	Value
@p0	'8b31bcf8-b55c-4e55-8e20-a49

The family of ‘N+1’s

Caused by looping over sub-collections

```
class TillGroup
{
        public decimal Balance {
                get { return moneyBags.Sum(t => t.Balance); }
        }
}

...

class MoneyBag
{
        … Balance { get { return payments.Sum(t => t.Amount); } }
}
```

Resolution

- We hoisted the balance property onto the Till, and pre-calculated.

- Getting the balance of a till-group was now a simple read from the database

```
public void AddTransaction(Transaction tran)
{
      transactions.Add(tran);
      Balance += tran.MoneyBags.Sum(t => t.Amount);
}

public decimal Balance { get; protected set; }
```

Support

"Till balancing doesn't always work"

Application Profiler *Fast*

Found in the log file

Timestamp	Thread	Message	**Balance**
2011-07-12-09:32:04.126	(1)	retail sale begin	**100**
2011-07-12-09:32:04.336	(2)	retail sale begin	**100**
2011-07-12-09:32:04.421	(1)	retail sale end	**130**
2011-07-12-09:32:05.073	(2)	retail sale end	**120**

Poor concurrency choice

Caused by our Sql Isolation Level

```
public void AddTransaction(Transaction tran)
{
      transactions.Add(tran);
      Balance += tran.MoneyBags.Sum(t => t.Amount);
}

public decimal Balance { get; protected set; }
```

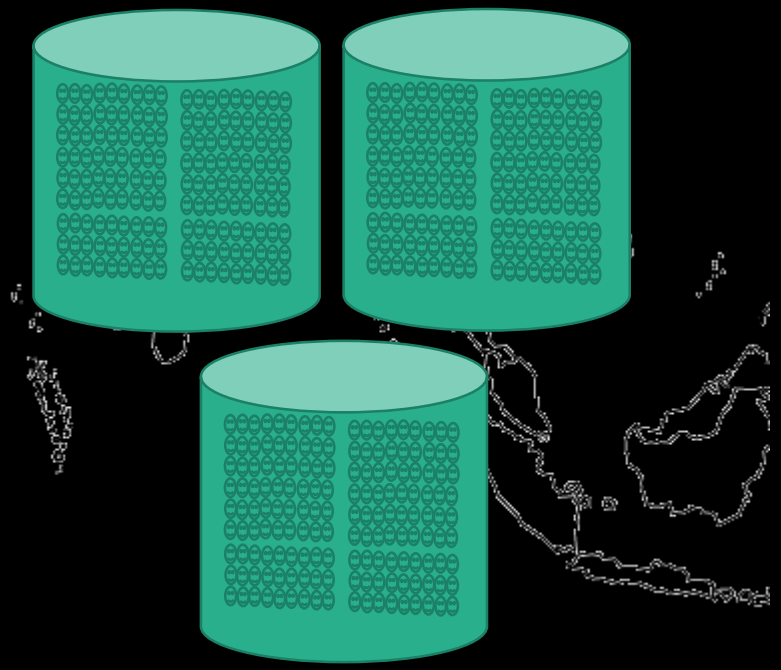
```
session.BeginTransaction(IsolationLevel.ReadCommitted);
```

Resolution

- We didn't nail the concurrency model early on
  - We took the database default of 'Read Committed'
- Optimistic concurrency models are well supported on ORM's like Hibernate but moving to one could be a costly exercise for us.
  - But they have a hidden side effect which disables sql batching
- But luckily, we have single stores per database, and a small number of users…

```
session.BeginTransaction(IsolationLevel.Serializable);
```

2013 -



"Hive" databases (up to 50 stores)

Large number of users
Hosted in Singapore

Database approaching 40GB in size
>400 concurrent users

Sometimes it all goes wrong!!

Support

"Victoria can't trade. The entire system is down"

Application Profiler *Lit up like a Christmas tree*

Suspicious log file entries

```
Timestamp Thread Message
2011-07-12-09:32:04.126 (1) finish stocktake
2011-07-12-09:32:05.336 (2) update stock
2011-07-12-09:32:05.567 (3) new retail sale
2011-07-12-09:32:06.600 (4) find stock
2011-07-12-09:33:05.336 (2) update stock timeout
2011-07-12-09:33:05.400 (8) finish stocktake
2011-07-12-09:33:05.567 (3) new retail sale timeout
2011-07-12-09:33:06.600 (4) find stock timeout
```

What does “finish stocktake do”?

```
SELECT ... FROM [User] user0_ WHERE user0_.Id = '5aac41b9-6141-4e26-8c72-050e482025ee'
SELECT ... FROM [Stocktake] stocktake0_ WHERE stocktake0_.Id = '3fa4f68e-07fe-42fa-80c4-360f0315fe97'
SELECT ... FROM [CashConvertersStore] cashconver0_ WHERE cashconver0_.StoreId = 'e7eca6a1-1248-444e-965b-756416cb39d
SELECT stocktakei0_.Stocktake_id as Stocktake4_...
SELECT ... FROM StoreItemActivity allactivit0_ WHERE allactivit0_.StoreItemBase_id in (Select storei...
SELECT ... FROM [ContractBase] contractba0_ left outer join [Cu... WHERE contractba0_Id in ('7e778baa-13fc-44bc-b757-a49801
SELECT ... FROM StoreItemStone stones0_ WHERE stones0_.ItemMetadata_id in (Select storeitemb1...
SELECT ... FROM [ItemWriteoff] writtenoff0_ inner join Activity... WHERE writtenoff0_.Stocktake_id = '3fa4f68e-07fe-42fa-80c4-36
SELECT ... FROM ItemCategory this_ left outer join ItemCategory... WHERE this_.FullName in ('Gardening Equipment\GOLDEN CA
SELECT this_.Id as Id113_6_, this_.Action as Ac...
INSERT INTO ActivityBase ...
INSERT INTO [ItemWriteoff] ...
INSERT INTO ActivityBase ...
INSERT INTO [ItemWriteoff] ...
INSERT INTO ActivityBase ...
INSERT INTO [ItemWriteoff] ...
```

Self induced denial of service

What the !#!##!?

- The Stock Take operation *greedily* locked rows causing other operations to block
- The ORM was not able to *batch* effectively and took a long time to flush
- Multi-tenanting had rendered some of our *sql indexes* useless
- The terminal *timed out*, and the user kept *retrying*
- The server-side did *not* time out

Potential fix #1

- The hibernate family of ORM have flags to improve batching
- Batching enables more sql to be sent to the database *per call*
- Can be a real improvement when latency to the database is high
- But batch operations will still take some time

Potential fix #2

- Move work to background processes to spread the load!
- For many use-cases your user's don't care if things don't happen instantly.

Quick resolution

Sometimes you need to get the network moving again…

```
if (time is between 8am and 7pm) {
      throw new DomainException(
            "Sorry. Please try again outside of business hours");
}
```

Conclusions

- Avoid chatty distributed components
- Keep your indexes tuned
- Think about concurrency
- Watch that ORM!!!
  - Fetch strategies, horrid joins, fetching too much, fetching too little!

Thank you for listening

Graeme Foster
@graefoster
gograemefoster@blogspot.com