Life Beyond Relational Database

Capital Match Team

2016-03-10

Agenda

- Introduction

Agenda

- Introduction
- Event-Sourcing Model

Agenda

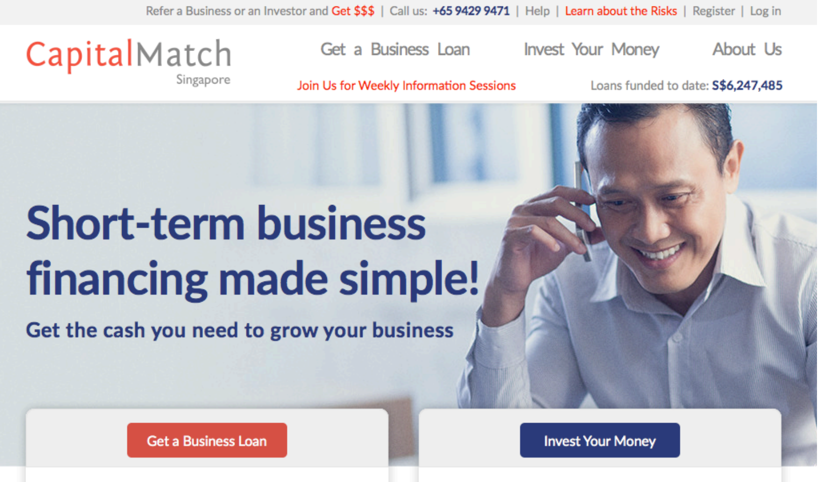
- Introduction
- Event-Sourcing Model
- Implementation & Usage

Agenda

- Introduction
- Event-Sourcing Model
- Implementation & Usage
- Future works

Introduction

Who are we?



Refer a Business or an Investor and Get $$$ | Call us: +65 9429 9471 | Help | Learn about the Risks | Register | Log in
CapitalMatch
Singapore
Get a Business Loan
Invest Your Money
About Us
Join Us for Weekly Information Sessions
Loans funded to date: S$6,247,485
Get a Business Loan
Invest Your Money

Short-term business financing made simple!

Get the cash you need to grow your business

Figure 1:

Who are we?

- Capital Match is the leading plaform in Singapore for peer-to-peer lending to SMEs

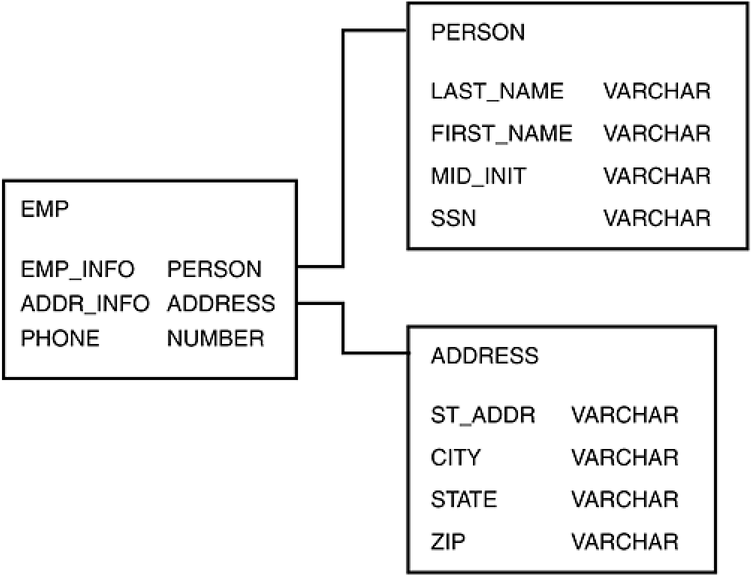
Who are we?

- Capital Match is the leading plaform in Singapore for peer-to-peer lending to SMEs
- Backend system developed in Haskell, frontend in Clojurescript/Om since 2014

Who are we?

- Capital Match is the leading plaform in Singapore for peer-to-peer lending to SMEs
- Backend system developed in Haskell, frontend in Clojurescript/Om since 2014
- Core Development team of $3 + 1$: Amar, Arnaud, Guo Liang, Zhou Yu

Relational Model



EMP

EMP_INFO	PERSON
ADDR_INFO	ADDRESS
PHONE	NUMBER

PERSON

LAST_NAME	VARCHAR
FIRST_NAME	VARCHAR
MID_INIT	VARCHAR
SSN	VARCHAR

ADDRESS

ST_ADDR	VARCHAR
CITY	VARCHAR
STATE	VARCHAR
ZIP	VARCHAR

Figure 2:

What's good with Relational Model?

- Really great for querying ⟶ *SQL Rocks!*

What's good with Relational Model?

- Really great for querying $\longrightarrow$ *SQL Rocks!*
- Conceptually simple to understand: *Everything is a Table*

What's good with Relational Model?

- Really great for querying $\longrightarrow$ *SQL Rocks!*
- Conceptually simple to understand: *Everything is a Table*
- Ubiquitous

What's wrong with Relational Model?

- Writes/updates are complex

What's wrong with Relational Model?

- Writes/updates are complex
- *Impedance Mismatch*: Lot of data is more tree-ish or graph-ish

What's wrong with Relational Model?

- Writes/updates are complex
- *Impedance Mismatch*: Lot of data is more tree-ish or graph-ish
- One single Database for everything $\longrightarrow$ *SPOF*

What's wrong with Relational Model?

- Writes/updates are complex
- *Impedance Mismatch*: Lot of data is more tree-ish or graph-ish
- One single Database for everything ⟶ *SPOF*
- **Mutable State**

Figure 3:

Event Sourcing

State vs. Transitions

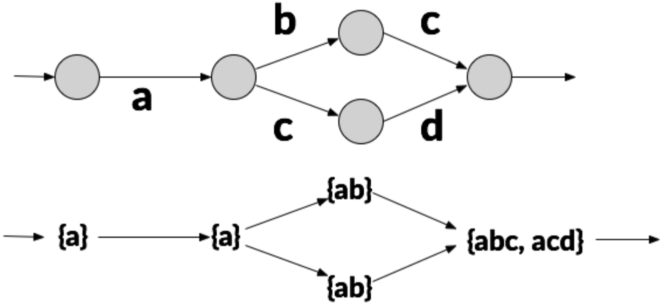


Figure 4:

State vs. Transitions

- RDBMS stores the **state** of the model at some point in time...

State vs. Transitions

- RDBMS stores the **state** of the model at some point in time...
- ... But we are also interested in the **transitions** ...

State vs. Transitions

- RDBMS stores the **state** of the model at some point in time...
- ... But we are also interested in the **transitions** ...
- ... And state[1] can always be reconstructed from a *sequence of transitions*.

[1] Assuming state is deterministic of course

The Event Sourcing Model

> *Event Sourcing ensures that all changes to application state are stored as a sequence of events. Not just can we query these events, we can also use the event log to reconstruct past states, and as a foundation to automatically adjust the state to cope with retroactive changes.*
>
> *Martin Fowler*

Events makes it easier to...

- Audit current state and what lead to it

Events makes it easier to...

- Audit current state and what lead to it
- Implement generic undo/redo mechanism[2]

[2] May require invertible events

Events makes it easier to...

- Audit current state and what lead to it
- Implement generic undo/redo mechanism[2]
- Run simulations with different hypothesis over live data

[2] May require invertible events

Events makes it easier to...

- Audit current state and what lead to it
- Implement generic undo/redo mechanism[2]
- Run simulations with different hypothesis over live data
- Cope with data format migrations

[2] May require invertible events

Events makes it easier to...

- Audit current state and what lead to it
- Implement generic undo/redo mechanism[2]
- Run simulations with different hypothesis over live data
- Cope with data format migrations
- Handle potentially conflicting changes[3]

[2] May require invertible events

[3] That's the way RDBMS handle transactional isolation: Record a *log* of all operations on data then reconcile when transactions are committed

Events Drive Business

- Events are what makes a model dynamic: What affects it, how it reacts to outside world...

Events Drive Business

- Events are what makes a model dynamic: What affects it, how it reacts to outside world...
- Provide foundation for Domain Driven Design techniques ⟶ Better business models, Ubiquitous language

Events Drive Business

- Events are what makes a model dynamic: What affects it, how it reacts to outside world...
- Provide foundation for Domain Driven Design techniques ⟶ Better business models, Ubiquitous language
- Lead to Event Storming technique for “requirements” elicitation and business domain modelling[4]

[4] I never know how many `l`s modelling takes...

In Practice

Overview

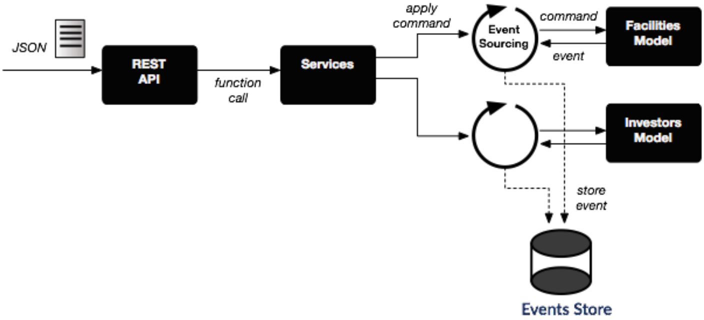


Figure 5:

Pure Business Models

- Each model delimits a *Bounded Context*: It is responsible for a single cohesive part of the domain

Pure Business Models

- Each model delimits a *Bounded Context*: It is responsible for a single cohesive part of the domain
- Models are **pure** immutable data structures

Pure Business Models

- Each model delimits a *Bounded Context*: It is responsible for a single cohesive part of the domain
- Models are **pure** immutable data structures
- Distinguish *Commands* from *Events*

Pure Business Models (2)

- Commands compute Event from State

```
act :: Command -> Model -> Event
```

Pure Business Models (2)

- Commands compute Event from State

  ```
  act :: Command -> Model -> Event
  ```

- Events modify model

  ```
  apply :: Event -> Model -> Model
  ```

Effectful Services

> *Services are used to orchestrate interaction between one or more business models and the outside world*

- Services are functions operating *across several contexts*

Effectful Services

> *Services are used to orchestrate interaction between one or more business models and the outside world*

- Services are functions operating *across several contexts*
- They can be synchronous or asynchronous (we use mostly synchronous)[5]

[5] Synchronicity is a property of the business domain, e.g. depends on what client expects from the service and whether or not he wants to "pay" for synchronous confirmation

Effectful Services

Services are used to orchestrate interaction between one or more business models and the outside world

- Services are functions operating *across several contexts*
- They can be synchronous or asynchronous (we use mostly synchronous)[5]
- There are no *distributed transactions*: Service has to cope with failures from each context

[5] Synchronicity is a property of the business domain, e.g. depends on what client expects from the service and whether or not he wants to “pay” for synchronous confirmation

Effectful Services (2)

- We have a monad to express effects and sequencing on each context: `WebStateM`

```
newtype WebStateM g l m a = WebStateM { runWebM :: TVar g -
```

Effectful Services (2)

- We have a monad to express effects and sequencing on each context: `WebStateM`

  ```
  newtype WebStateM g l m a = WebStateM { runWebM :: TVar g -
  ```

- g is a "global" Model which can be accessed concurrently ⟶ protected in STM

Effectful Services (2)

- We have a monad to express effects and sequencing on each context: `WebStateM`

  ```
  newtype WebStateM g l m a = WebStateM { runWebM :: TVar g -
  ```

- g is a "global" Model which can be accessed concurrently ⟶ protected in STM
- `l` is local data, contextual to a single service execution

Effectful Services (2)

- We have a monad to express effects and sequencing on each context: `WebStateM`

  ```
  newtype WebStateM g l m a = WebStateM { runWebM :: TVar g -
  ```

- g is a “global” Model which can be accessed concurrently ⟶ protected in STM
- `l` is local data, contextual to a single service execution
- `m` is underlying monad, usually `IO`

Events Storage

```
data StoredEvent s = StoredEvent { eventVersion :: EventVersi
                           , eventType   :: EventType s
                                 , eventDate    :: Date
                               , eventUser    :: UserId
                          , eventRequest :: Encoded Hex
                           , eventSHA1   :: Encoded Hex
                            , event       :: ByteString
                                  }
```

Events Storage (2)

- We use a simple Append-only file store, writing serialized events (mostly JSON) packed with metadata

Events Storage (2)

- We use a simple Append-only file store, writing serialized events (mostly JSON) packed with metadata
- Each event has a (monotonically increasing) version which is used for proper deserialization

Events Storage (2)

- We use a simple Append-only file store, writing serialized events (mostly JSON) packed with metadata
- Each event has a (monotonically increasing) version which is used for proper deserialization
- Events carry useful information for troubleshooting and auditing: User who initiated the request, request id itself, SHA1 representing version of appplication

Events Storage (2)

- We use a simple Append-only file store, writing serialized events (mostly JSON) packed with metadata
- Each event has a (monotonically increasing) version which is used for proper deserialization
- Events carry useful information for troubleshooting and auditing: User who initiated the request, request id itself, SHA1 representing version of appplication
- Events Store serializes concurrent writes

Software

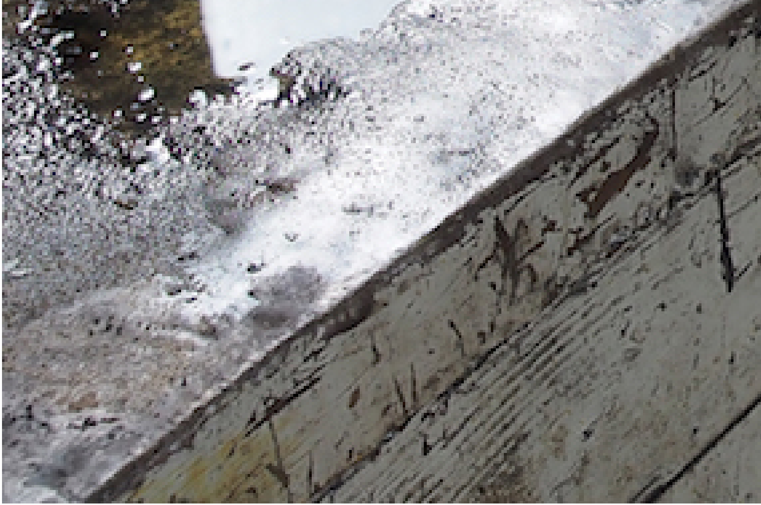


Figure 6: In Practice

Demo

- Anatomy of a complete business model

Demo

- Anatomy of a complete business model
    - Web layer w/ servant

Demo

- Anatomy of a complete business model
  - Web layer w/ servant
  - Service layer (w/ Free monads...)

Demo

- Anatomy of a complete business model
  - Web layer w/ servant
  - Service layer (w/ Free monads...)
  - Business model

Demo

- Anatomy of a complete business model
  - Web layer w/ servant
  - Service layer (w/ Free monads...)
  - Business model
  - Migration code

Demo

- Anatomy of a complete business model
  - Web layer w/ servant
  - Service layer (w/ Free monads...)
  - Business model
  - Migration code
  - Standalone service

Demo

- Anatomy of a complete business model
    - Web layer w/ servant
    - Service layer (w/ Free monads...)
    - Business model
    - Migration code
    - Standalone service
- Using Haskell scripts for operational queries and updates

Future Works

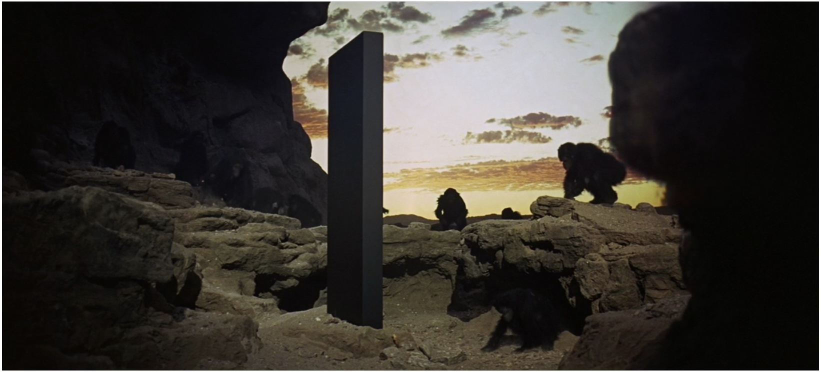


Figure 7:

Implement Better CQRS

- Separate *Read Model* from *Write Model*

Implement Better CQRS

- Separate *Read Model* from *Write Model*
- *Write Model*: Append-only linear data store per context, very fast, minimize locking/write time

Implement Better CQRS

- Separate *Read Model* from *Write Model*
- *Write Model*: Append-only linear data store per context, very fast, minimize locking/write time
- *Read model*: Optimized for specific querying, may be relational if needed in order to make it more user-friendly

Make models resilient

- Resilience of models $\longrightarrow$ *Replication*

Make models resilient

- Resilience of models ⟶ *Replication*
- Use Raft to maintain strong consistency of models: several implementations in Haskell

Make models resilient

- Resilience of models ⟶ *Replication*
- Use Raft to maintain strong consistency of models: several implementations in Haskell
- Started implementation of practical cluster based on Raft, called raptr

Make models secure

- Turn event stream into a *source of truth* ⟶ Blockchain[6] and beyond...

[6] Blockchain is all rage in the FinTech ecosystem those days, although early implementation like Bitcoins or Dogecoins failed to deliver all their promises.

Make models secure

- Turn event stream into a *source of truth* ⟶ Blockchain[6] and beyond...
- Juno: Smart contracts over Raft cluster

[6] Blockchain is all rage in the FinTech ecosystem those days, although early implementation like Bitcoins or Dogecoins failed to deliver all their promises.

Make models secure

- Turn event stream into a *source of truth* ⟶ Blockchain[6] and beyond...
- Juno: Smart contracts over Raft cluster
- Uses cryptographically signed events to ensure history cannot be tampered with

[6] Blockchain is all rage in the FinTech ecosystem those days, although early implementation like Bitcoins or Dogecoins failed to deliver all their promises.

Make models secure

- Turn event stream into a *source of truth* ⟶ Blockchain[6] and beyond...
- Juno: Smart contracts over Raft cluster
- Uses cryptographically signed events to ensure history cannot be tampered with
- Turns journal into a “legally binding ledger”?

[6] Blockchain is all rage in the FinTech ecosystem those days, although early implementation like Bitcoins or Dogecoins failed to deliver all their promises.

Questions?

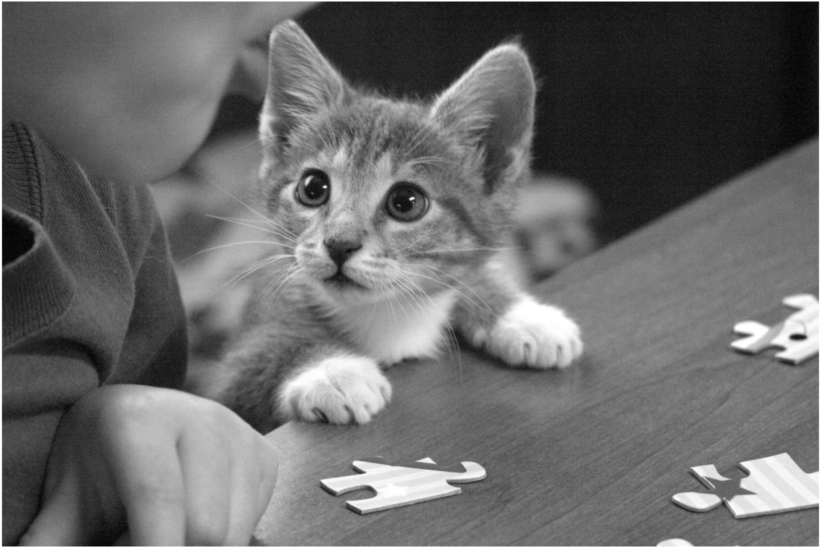


Figure 8:

Credits