

# Eventstore Sample Course



# Overview

- Doing a Thing
- Doing that other thing
- What it all means

# Definitions of Terms

A stream is blah blah blah blah

Immutable, flexible schema, type flag defines contract with application using the stream



# A numbered list

1. In your IDE, you will find a code stub that has your write your first event to a stream in EventstoreDB.
2. Title of the stream:
3. Use your name as the stream title

# Lab Instructions

1. Open up your Lab environment
2. Run the test notebook to see that your node environment is functioning
3. Start your docker container running eventstore
4. Open up the stub notebook and follow the instructions
5. Run the validate notebook to verify you have succeeded

# Let's Review

You need to respecify the background if you change it, otherwise it repeats

# Some Benefits of Event Sourcing

- Audit

Event-source stores data as a series of immutable events over time, providing a strong audit capability out-the-box.

- Root Cause Analysis

Business events are tied back to their original chains of causation, allowing visibility into entire workflows.



# Benefits of Event Sourcing continued...

- Time Travel

All state changes are kept, so it is possible to move entire system state backwards in time for debugging and "what-if" analysis.

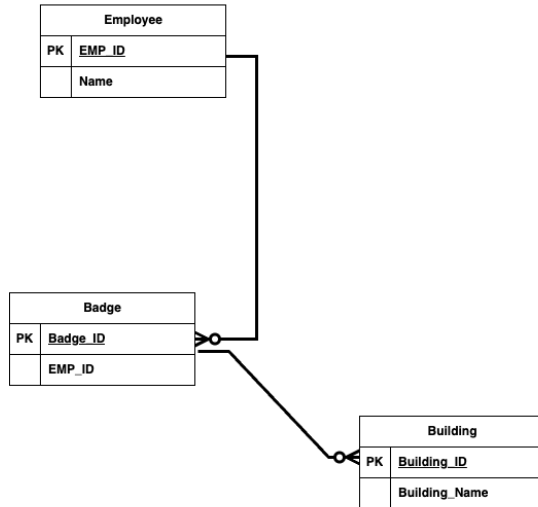
- Projections

Event streams can (re)played through different fold operations to look at existing data in new and interesting ways.



# Including Images

Here is an included image



# Lab Instructions

1. Open up the folder titled your second stream and follow the instructions
2. If you have issues you can run the test\_node.js file
3. When you are done you can run the validate lab script to see if you have succeeded

We are back to old style



The background is a dense, hand-drawn diagram in white lines on a dark blue background, illustrating Event Sourcing architecture. It shows multiple instances of the system. Key components include: 

- Command**: Represented by rectangles, showing actions like 'Check out', 'Get order overview', 'Mutate the state', 'Load the aggregate', and 'Update Read Model'.
- Query**: Represented by rectangles, showing requests like 'Get order overview'.
- Query handler**: Represented by ovals, processing queries.
- Command handler**: Represented by ovals, processing commands.
- Domain model**: Represented by ovals, the core logic of the application.
- Database**: Represented by cylinders, storing data.
- Event Store**: Represented by cylinders, storing a sequence of events.
- Projection**: Represented by ovals, deriving read models from events.
- Query database**: Represented by cylinders, storing query results.
- Aggregates**: Represented by rectangles with internal boxes, showing state like 'Orders', 'Shipping', and 'Payment'.
- Users**: Represented by stick figures, interacting with the system.

Arrows indicate the flow of data and control between these components across different system instances.

Hello world

Red text centered (only in the current slide page)

Back to normal



Red text centered (only in the current slide page)

# Back to Regular