

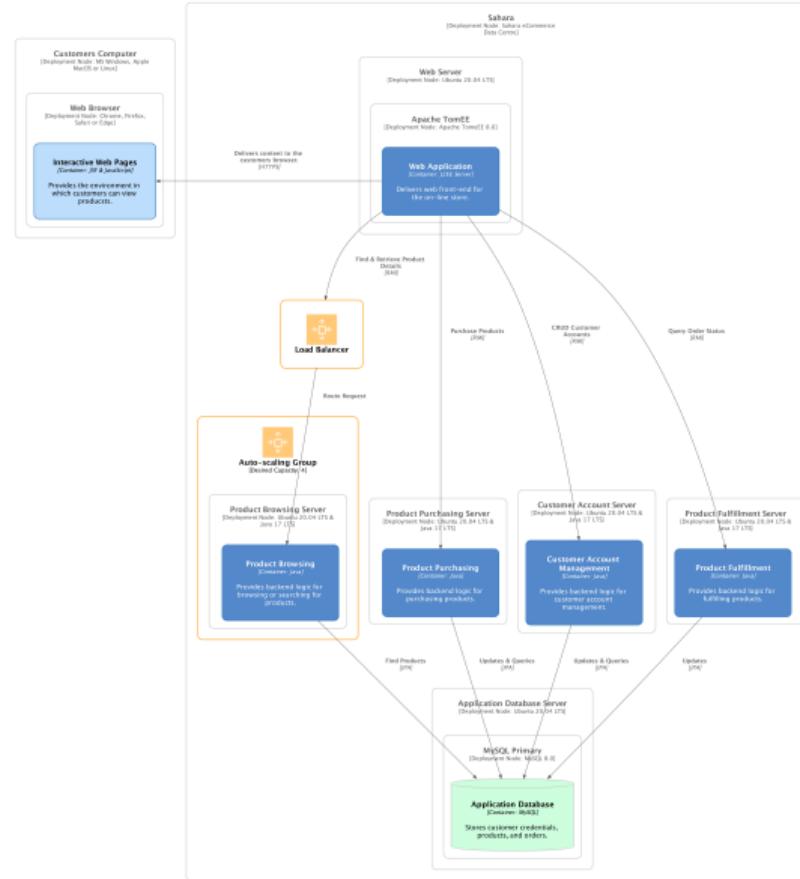
# Distributed Computing II

CSSE6400

Brae Webb

April 4, 2022

# Previously in CSSE6400...



Question

What is the *problem*?

# Database



Disclaimer

This is *not* a database course.



# Advanced Database Systems (INFS3200)

**Course level**

Undergraduate

**Faculty**

Engineering, Architecture &amp; Information Technology

**School**

Info Tech &amp; Elec Engineering

**Units**

2

**Duration**

One Semester

**Class contact**

2 Lecture hours, 1 Tutorial hour, 1 Practical or Laboratory hour

**Incompatible**

INFS7907

**Prerequisite**

INFS2200

**Assessment methods**

## Current course offerings

Course offerings	Location	Mode	Course Profile
Semester 1, 2022	St Lucia	Internal	<a href="#">COURSE PROFILE</a>
Semester 1, 2022	External	External	<a href="#">COURSE PROFILE</a>
Semester 2, 2022	External	External	PROFILE UNAVAILABLE
Semester 2, 2022	St Lucia	Internal	PROFILE UNAVAILABLE

Please Note: Course profiles marked as not available may still be in development.

## Course description

Distributed database design, query and transaction processing, data integration, data warehousing, data cleansing, management of spatial data, and data from large scale distributed devices.

## Archived offerings

Course offerings	Location	Mode	Course Profile
Semester 1, 2021	St Lucia	Flexible Delivery	<a href="#">COURSE PROFILE</a>
Semester 1, 2021	External	External	<a href="#">COURSE PROFILE</a>
Semester 2, 2021	External	External	<a href="#">COURSE PROFILE</a>
Semester 2, 2021	St Lucia	Internal	<a href="#">COURSE PROFILE</a>
Semester 1, 2020	St Lucia	Internal	<a href="#">COURSE PROFILE</a>

Question

How do we fix database scaling issues?

Question

# How do we fix database scaling issues?

Answer

- Replication

Question

# How do we fix database scaling issues?

Answer

- Replication
- Partitioning

Question

# How do we fix database scaling issues?

Answer

- Replication
- Partitioning
- Independent databases

Question

# How do we fix database scaling issues?

Answer

- *Replication*
- Partitioning
- Independent databases

Question

What is *replication*?

### Definition 1. Replication

Data copied across multiple different machines.



product_id	name	stock	price
1234	Nicholas Cage Reversible Pillow	10	\$10.00
4321	Lifelike Elephant Inflatable	5	\$50.00



product_id	name	stock	price
1234	Nicholas Cage Reversible Pillow	10	\$10.00
4321	Lifelike Elephant Inflatable	5	\$50.00

### Definition 2. Replica

Database node which stores a copy of the data.

Question

What are the advantages of *replication*?

Question

What are the advantages of *replication*?

Answer

- Scale out our database to cope with *load*.

Question

What are the advantages of *replication*?

Answer

- Scale out our database to cope with *load*.
- Provide *fault tolerance* from a single database instance failure.

Question

# What are the advantages of *replication*?

Answer

- Scale out our database to cope with *load*.
- Provide *fault tolerance* from a single database instance failure.
- Locate databases *closer to end-users*.

Question

**How do we replicate our data?**

First approach

## Leader-follower Replication



Leader-based Replication

**On write** Writes sent to leader, change is propagated via change stream.

## Leader-based Replication

**On write** Writes sent to leader, change is propagated via change stream.

**On read** Any replica can be queried.



Propogating changes

## *Synchronous vs. Asynchronous*

ProductBrowsing

Leader

Follower1

1 UPDATE products SET stock=4

2 UPDATE

3 OK

4 OK

ProductBrowsing

Leader

Follower1

ProductBrowsing

Leader

Follower1

1 UPDATE products SET stock=4

2 UPDATE

3 OK

4 OK

ProductBrowsing

Leader

Follower1



## Synchronous propagation

- Writes must propagate to *all followers* before being successful.

## Synchronous propagation

- Writes must propagate to *all followers* before being successful.
- *Any* replica goes down, *all* replicas are un-writable.

## Synchronous propagation

- Writes must propagate to *all followers* before being successful.
- *Any* replica goes down, *all* replicas are un-writable.
- Writes must *wait* for propagation to all replicas.

## Asynchronous propagation

- Writes *don't* have to *wait* for propagation.

## Asynchronous propagation

- Writes *don't* have to *wait* for propagation.
- If the leader goes down before propagating, the *write is lost*.

## Asynchronous propagation

- Writes *don't* have to *wait* for propagation.
- If the leader goes down before propagating, the *write is lost*.
- Replicas can have out-dated or *stale* data.

### Definition 3. Replication Lag

The time taken for replicas to update *stale* data.





Eventually, all replicas must become consistent

The system is *eventually consistent*

Eventual Consistency

**Problems?**



**Brae Webb**  
[@braewebb](https://twitter.com/braewebb)



**Brae Webb**  
**@braewebb**

Name: Brae

Cancel

Save



**Brae Webb**  
@braewebb

Name:

Cancel

Save



**Brae Webb**  
@braewebb



#### Definition 4. Read-your-writes Consistency

Users always see the updates that *they have made*.



**Brae Webb**  
[@braewebb](https://twitter.com/braewebb)

My fist post



**Brae Webb**  
*@braewebb*

My fist post



**Brae Webb**  
*@braewebb*

My first post



**Brae Webb**  
[@braewebb](https://twitter.com/braewebb)

My fist post



**Brae Webb**  
[@braewebb](https://twitter.com/braewebb)

My first post



**Brae Webb**  
[@braewebb](https://twitter.com/braewebb)

My fist post



### Definition 5. Monotonic Reads

Once a user reads an updated value, they don't later see the old value.

## Summary

- Leader-follower databases allow *reads to scale* more effectively.
- Asynchronous propagation weakens consistency to *eventually consistent*.
- Leader-follower databases still have a *leader write bottle-neck*.

Second approach

## Multi-leader Replication



Why multi-leader?

- If you have multiple leaders, you can write to any, allowing *writes to scale*.

## Why multi-leader?

- If you have multiple leaders, you can write to any, allowing *writes to scale*.
- A leader going down doesn't prevent writes, giving *better fault-tolerance*.

Question

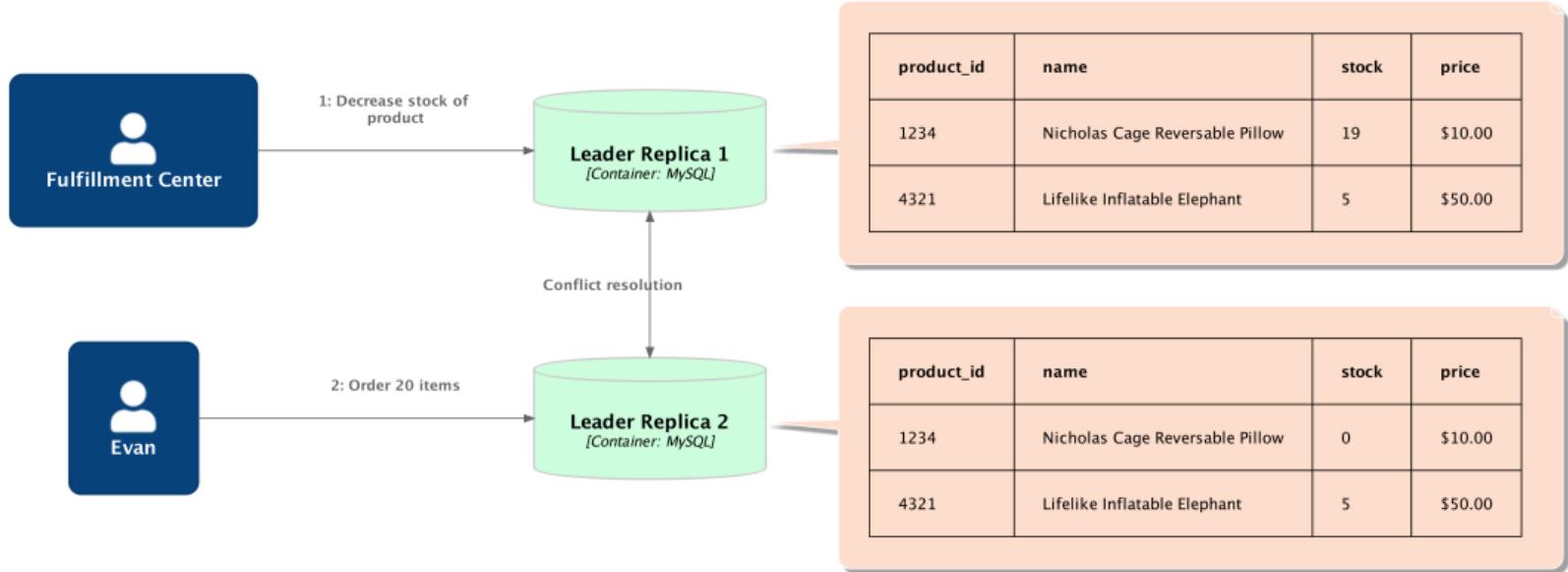
What might go wrong?

Question

What might go wrong?

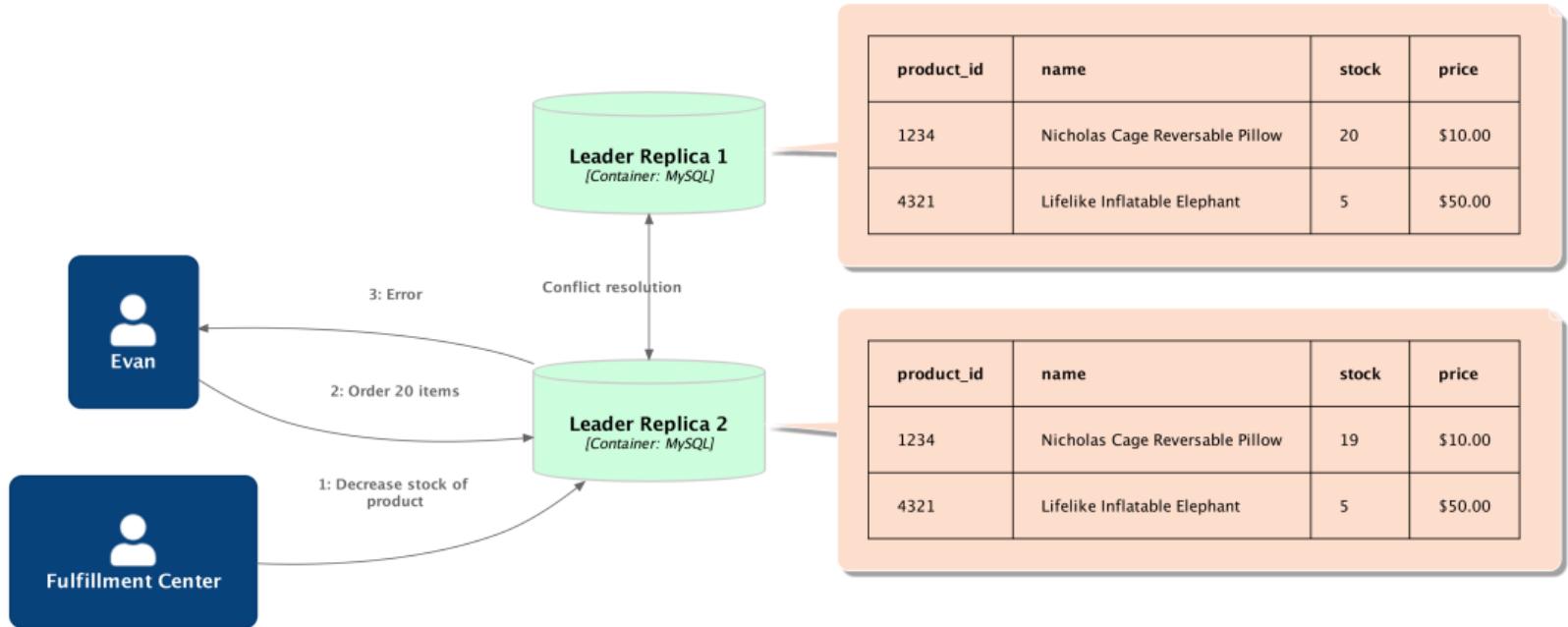
Answer

Write conflicts



Where possible

Avoid write conflicts



Where impossible

Convergence

## Convergence Strategies

- Assign each *write* a unique ID.

## Convergence Strategies

- Assign each *write* a unique ID.
- Assign each *leader replica* a unique ID.

## Convergence Strategies

- Assign each *write* a unique ID.
- Assign each *leader replica* a unique ID.
- Custom resolution logic.



## Resolving Conflicts

**On Write** When a conflict is first noticed, take proactive resolution action.

**On Read** When a conflict is next read, ask for a resolution.

Third Approach

## Leaderless Replication



How do they work?

Each read/write is sent to *multiple* replicas.





How are changes propagated?

- Read Repair

How are changes propagated?

- Read Repair
- Anti-entropy Process

Question

**How do we know it's consistent?**



Question

**How do we know it's consistent?**

Question

**How do we know it's consistent?**

Answer

**Quorum Reads and Writes**

## Quorum Consistency

$$w + r > n$$

$n$  total replicas

$w$  amount of replicas to *write* to

$r$  amount of replicas to *read* from



*n* total replicas

*w* amount of replicas to *write* to

*r* amount of replicas to *read* from

## Summary

- *Replication* copies data to multiple replicas.

## Summary

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simplest.

## Summary

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simplest.
- Replication introduces *eventual consistency*.

## Summary

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simplest.
- Replication introduces *eventual consistency*.
- *Multi-leader* replication scales writes as well as reads but introduces *write conflicts*.

## Summary

- *Replication* copies data to multiple replicas.
- *Leader-based* replication is most common and simplest.
- Replication introduces *eventual consistency*.
- *Multi-leader* replication scales writes as well as reads but introduces *write conflicts*.
- *Leaderless* replication is another approach which keeps the problems of multi-leader.

Question

How do we fix database scaling issues?

Question

# How do we fix database scaling issues?

Answer

- *Replication*
- Partitioning
- Independent databases

Question

# How do we fix database scaling issues?

Answer

- Replication
- *Partitioning*
- Independent databases

### Definition 6. Partitioning

Split the data of a system onto multiple nodes, these nodes are *partitions*.



product_id	name	stock	price
4321	Lifelike Elephant Inflatable	5	\$50.00



product_id	name	stock	price
1234	Nicholas Cage Reversible Pillow	10	\$10.00

Question

How should we decide which data is stored where?



student_id	name	...
s0746283	Bobby Tables	...
...	...	...



student_id	name	...
s1637285	Brae Webb	...
...	...	...

Question

What is the problem with this?

Question

What is the problem with this?

Answer

Over time some partitions become inactive,  
while others receive almost all load.

Question

How should we decide which data is stored where?

Question

How should we decide which data is stored where?

Answer

Maximize spread of requests, avoiding  
*skewing*.

Question

**Have we seen this before?**

Question

Have we seen this before?

Answer

Hashing?

Question

What is the problem with this?

Question

What is the problem with this?

Answer

Range queries are inefficient, i.e. get all students between s44444444 and s4565656

Question

How do we route queries?

### Query-insensitive Load Balancer

**Randomly route to any node, responsibility of the node to re-route to the correct node.**



### Query-sensitive Load Balancer

A load balancer which understands which queries should be forwarded to which node.



## Client-aware Queries

Place the responsibility on clients to choose the correct node.



## Summary

- *Partitioning* splits data across multiple nodes.

## Summary

- *Partitioning* splits data across multiple nodes.
- A *consistent method* to chose which node is required.

## Summary

- *Partitioning* splits data across multiple nodes.
- A *consistent method* to chose which node is required.
- Partitioning by *primary key* can create *skewing*.

## Summary

- *Partitioning* splits data across multiple nodes.
- A *consistent method* to chose which node is required.
- Partitioning by *primary key* can create *skewing*.
- Partitioning by *hash* makes range queries less efficient.

## Summary

- *Partitioning* splits data across multiple nodes.
- A *consistent method* to chose which node is required.
- Partitioning by *primary key* can create *skewing*.
- Partitioning by *hash* makes range queries less efficient.
- Three approaches to *routing requests*.

Disclaimer

We have ignored the hard parts of replication.

Question

How do we fix database scaling issues?

Question

# How do we fix database scaling issues?

Answer

- Replication
- *Partitioning*
- Independent databases

Question

# How do we fix database scaling issues?

Answer

- Replication
- Partitioning
- *Independent databases*

Distributed state creates a lot of *complexity*

When programmers are faced with complexity

They create *abstractions*

One key database abstraction

# Transactions

### **Definition 7. Transaction**

A group of operations performed as if they were one.

ACID

A tomic

C onsistent

I solated

D urable

The pushback

NoSQL and microservice architectures  
pushed back against transactions.

For more on transactions



Sept 25-26, 2015  
thestrangeloop.com



## Summary

- Replications

## Summary

- **Replications**

- Leader-based, multi-leader, and leaderless

## Summary

- **Replications**

- Leader-based, multi-leader, and leaderless
- Eventual consistency

## Summary

### • Replications

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

## Summary

- **Replications**

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- **Partitioning**

## Summary

- **Replications**

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- **Partitioning**

- Consistent method to pick nodes for data

## Summary

- **Replications**

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- **Partitioning**

- Consistent method to pick nodes for data
- Avoiding skewing

## Summary

- **Replications**

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- **Partitioning**

- Consistent method to pick nodes for data
- Avoiding skewing

- **Transactions**

## Summary

- **Replications**

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- **Partitioning**

- Consistent method to pick nodes for data
- Avoiding skewing

- **Transactions**

- ACID properties

## Summary

- **Replications**

- Leader-based, multi-leader, and leaderless
- Eventual consistency
- Write conflicts

- **Partitioning**

- Consistent method to pick nodes for data
- Avoiding skewing

- **Transactions**

- ACID properties
- Pushback causing headaches