STADVDB MCO2

# A Three-Node Implementation of a MySQL Distributed Database System

**Dalisay, Pinawin, Salvador, Sy**

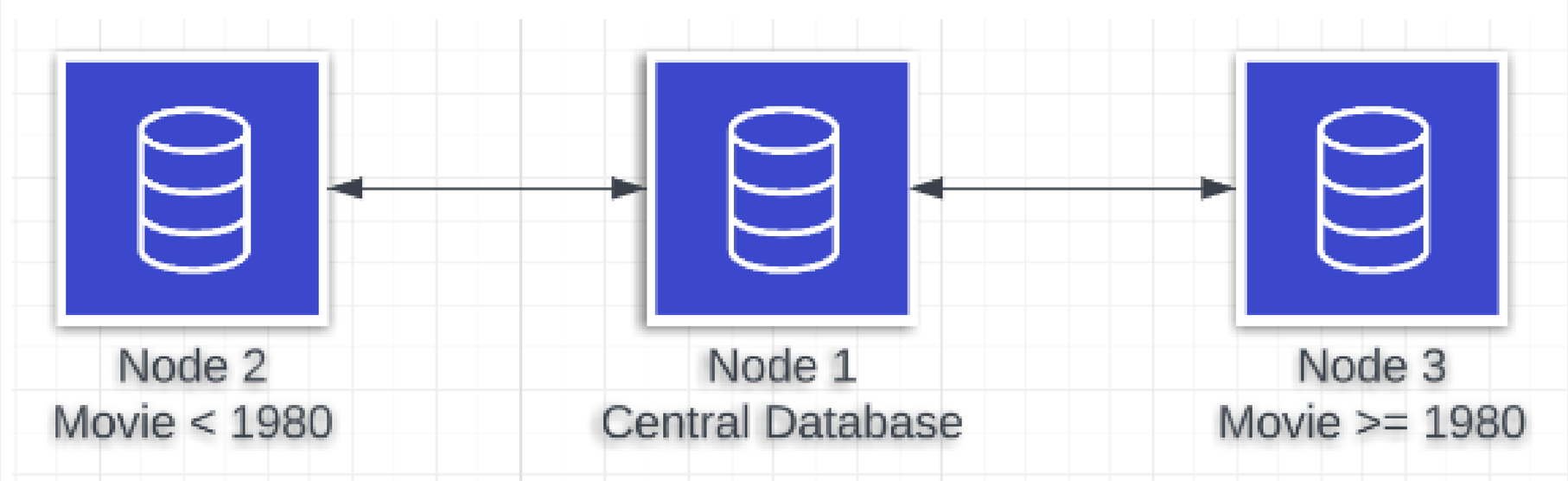
# Overview

Distributed Database Setup
Concurrency and Replication
Recovery Strategy
Discussion and Conclusion

# Distributed Database Setup

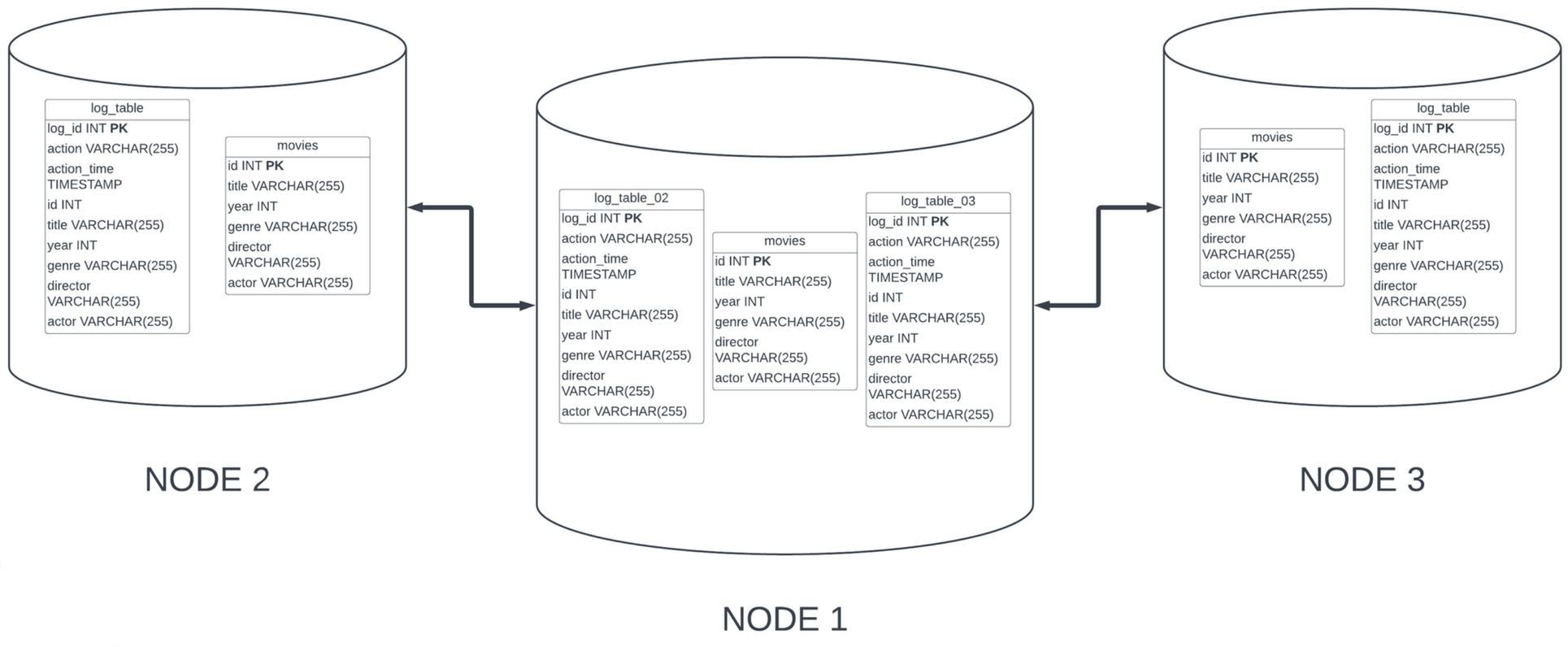
| | |
|---|---|
| **cloud and web platforms used** | NodeJS \| ExpressJS \| CCS Cloud |
| **data source** | IMDB ijs dataset |
| **data extraction** | Apache NiFi |
| **node setup** | 3 Nodes<br>Partial Replication<br>Horizontal Fragmentation<br>Multi-master setup |

# Distributed Database Setup



# Concurrency & Replication

# Log Tables


log_table
log_id INT PK
action VARCHAR(255)
action_time TIMESTAMP
id INT
title VARCHAR(255)
year INT
genre VARCHAR(255)
director VARCHAR(255)
actor VARCHAR(255)
movies
id INT PK
title VARCHAR(255)
year INT
genre VARCHAR(255)
director VARCHAR(255)
actor VARCHAR(255)
NODE 2
log_table_02
log_id INT PK
action VARCHAR(255)
action_time TIMESTAMP
id INT
title VARCHAR(255)
year INT
genre VARCHAR(255)
director VARCHAR(255)
actor VARCHAR(255)
movies
id INT PK
title VARCHAR(255)
year INT
genre VARCHAR(255)
director VARCHAR(255)
actor VARCHAR(255)
log_table_03
log_id INT PK
action VARCHAR(255)
action_time TIMESTAMP
id INT
title VARCHAR(255)
year INT
genre VARCHAR(255)
director VARCHAR(255)
actor VARCHAR(255)
NODE 1
movies
id INT PK
title VARCHAR(255)
year INT
genre VARCHAR(255)
director VARCHAR(255)
actor VARCHAR(255)
log_table
log_id INT PK
action VARCHAR(255)
action_time TIMESTAMP
id INT
title VARCHAR(255)
year INT
genre VARCHAR(255)
director VARCHAR(255)
actor VARCHAR(255)
NODE 3

# Triggers

```
DELIMITER //
CREATE TRIGGER insert_movie AFTER INSERT
ON movies FOR EACH ROW
BEGIN
    IF NEW.year < 1980 THEN
        INSERT INTO log_table_02 SET
        `action` = 'INSERT',
        `action_time` = NOW(),
        id = NEW.id,
        title = NEW.title,
        `year` = NEW.`year`,
        genre = NEW.genre,
        director = NEW.director,
        actor = NEW.actor;
    ELSE
        INSERT INTO log_table_03 SET
        `action` = 'INSERT',
        `action_time` = NOW(),
        id = NEW.id,
        title = NEW.title,
        `year` = NEW.`year`,
        genre = NEW.genre,
        director = NEW.director,
        actor = NEW.actor;
    END IF;
END//
DELIMITER ;
```

# Synchronizer

## sync_fragment

Synchronizes a fragment node (Node 2 or 3) to Node 1. Executes any queries that change the movies table.

## sync_central

Synchronizes Node 1 to the fragment nodes (Nodes 2 and 3). Executes any queries that change the movies table.

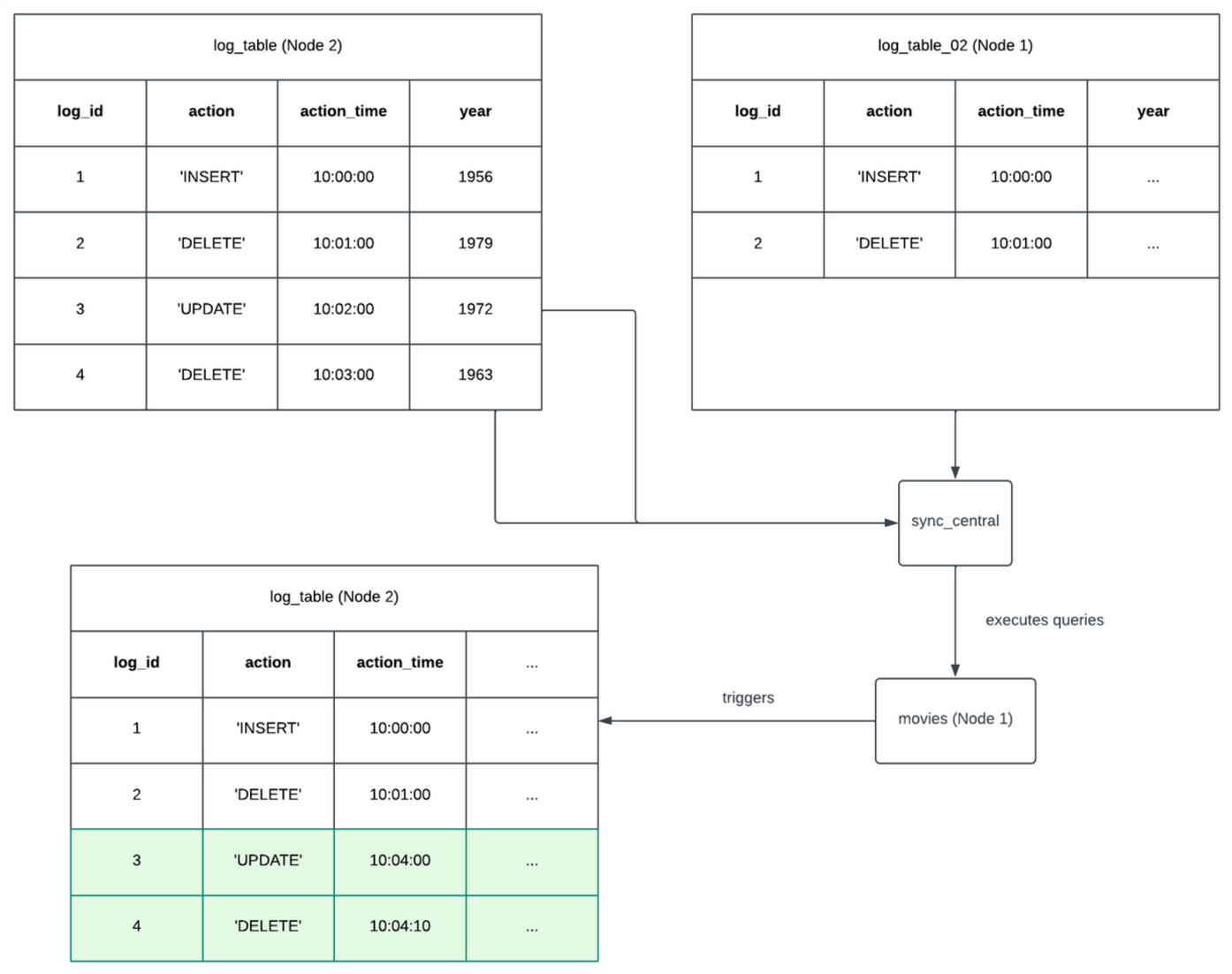
# sync_fragment

1. Query maximum log_id values in given fragment node's log table and its respective Node 1 log table
2. Check if Node 1 has a higher maximum log_id than fragment node
3. Take all log records from Node 1 log table starting from last log_id in fragment node
4. Parse each log record into appropriate queries
5. Execute each query on fragment node

# sync_central

1. Query maximum log_id values in both of Node 1's log tables and Node 2 and 3's log tables
2. Compare if Node 2 and 3's log tables have higher maximum log_id's than Node 1's
3. If both or one is true, take all log records from Node 2 and 3's log tables starting from last log_id in each of Node 1's log tables
4. Concatenate Node 2 and 3's log records and sort them by timestamp starting from the oldest
5. Parse each log record into appropriate queries
6. Execute each query on Node 1

# Example Flow Diagram for sync_central



# Recovery Strategy

# Log Tables

Node 1 -> 2 Log tables

Node 2 & 3 -> 1 Log table

```sql
CREATE TABLE log_table (
    `log_id` INT NOT NULL AUTO_INCREMENT,
    `action` VARCHAR(255) NOT NULL,
    `action_time` TIMESTAMP NOT NULL,
    `id` INT,
    `title` VARCHAR(255),
    `year` INT,
    `genre` VARCHAR(255),
    `director` VARCHAR(255),
    `actor` VARCHAR(255),
    PRIMARY KEY (`log_id`)
);
```

# Recovery Algorithm

1. Log-Tables
    a. INSERT
    b. DELETE
    c. UPDATE
2. App-level synchronizer functions
    a. called after every transaction
    b. called after 1 second

When a node fails, it can have a fast and efficient recovery

# Experiments

# Concurrency Control

1. 2 Transactions reading the same data item
2. 1 Transaction updates data item and 1 Transactions reads the data item
3. 2 Transactions updates the same data item

# Global Failure & Recovery

1. Central Node is Down while a Transaction is happening
2. Node 2/3 is Down while a Transaction is happening
3. Replication Error in Central Node
4. Replication Error in Node 2/3

# Discussion & Conclusion

Our **replication strategies** enabled our system to keep data consistent across all nodes when any updates (INSERT, UPDATE, DELETE queries) are executed on one node. Data is correctly replicated to their respective nodes based on the given year.

In terms of **recovery**, our automatic synchronizer makes sure that failed nodes immediately try to resync their data with the rest of the nodes.

End

# Thank you

Do you have any questions?

STADVDB MCO2

# A Three-Node Implementation of a MySQL Distributed Database System

**Dalisay, Pinawin, Salvador, Sy**