Design of a Distributed Database

Erick Gonzalez Parada ID: 178145
Emiliano Ruiz Plancarte ID: 177478
Andre Francois Duhamel Gutierrez ID: 177315
Antonio Gutiérrez Blanco ID: 177442

April 22, 2025

Abstract

This document explores the practical design of a distributed database for managing multimedia content related to movies and music. The design includes a conceptual schema (UML), a relational schema, and a query interface for interacting with the database. We implement a Java CRUD application that demonstrates vertical fragmentation in CockroachDB, a distributed SQL database system.

Keywords: Multimedia, Database Design, Distributed Database, UML, Query Interface, Vertical Fragmentation, CockroachDB.

1 Objective

The objective of this exercise is to develop practical skills in database design by analyzing a given scenario and proposing a database solution. This includes designing the conceptual schema (UML), the relational schema, and a query interface for interacting with the database. Additionally, we implement a practical Java application that demonstrates vertical fragmentation in a distributed database environment using CockroachDB.

2 Theoretical Framework

Distributed database systems represent a paradigm shift from traditional centralized databases by distributing data across multiple geographic locations while maintaining a unified logical view for users [2]. These systems offer several advantages including improved performance, reliability, and availability.

2.1 Types of Data Distribution

There are two primary approaches to distributing data in a distributed database system:

- **Horizontal Fragmentation (Sharding):** Divides tables by rows across different locations. Each location contains a subset of the complete data set but maintains the same schema structure [3].
- **Vertical Fragmentation:** Divides tables by columns across different locations. Each location contains specific attributes of the data, which can be optimized based on access patterns or regional requirements [1].

Our implementation focuses on vertical fragmentation, where different attributes of the movie data are stored in different regions, optimizing data locality and access patterns.

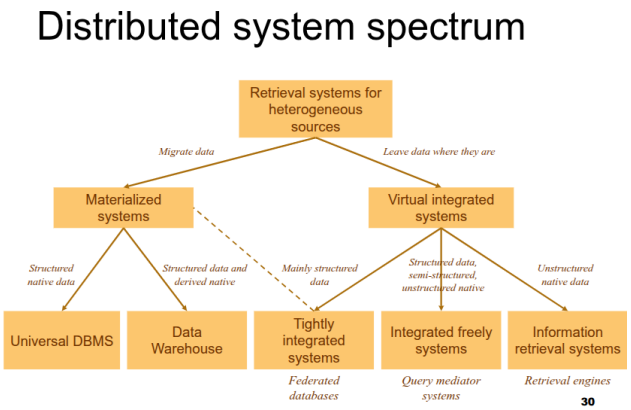


Figure 1: Distributed Database Spectrum

2.2 CockroachDB and Distributed SQL

CockroachDB is a distributed SQL database that automatically replicates and distributes data across multiple nodes while maintaining ACID compliance [1]. It uses a technique called "multi-active availability" to ensure data consistency across distributed nodes. Key features include:

- Automatic sharding and rebalancing
- Distributed transactions with serializable isolation
- Multi-region deployment capabilities
- Survivability during network partitions and node failures
- SQL compatibility with PostgreSQL

In our implementation, we leverage CockroachDB's multi-region capabilities to demonstrate vertical fragmentation by region, where different attributes of our movie data are stored in different geographic regions.

3 Summary of the Mini-World Description

Based on the description of the mini-world from the previous activity, we identified the following main entities:

- **Movie:** Title, duration, release date, classification, directors, producers, actors, and associated soundtracks.
- **Director:** Personal information + filmography.
- **Actor:** Personal information + filmography.
- **Producer:** Name, contact information, list of produced movies, and invested amount.
- **Soundtrack:** Title, author, performers, duration, date, classification.
- **Author:** Personal information + list of written soundtracks.
- **Performer:** Personal information + list of performed soundtracks.
- **Cinephile:** Personal information, favorite music, and favorite movies.

The MyCineMusic database is designed to manage multimedia content, particularly movies and music. It stores information about artists, albums, songs, movies, genres, user ratings, and playback history. To better understand the structure of the mini-world, a UML diagram was created, showing the different classes with their relationships and attributes.

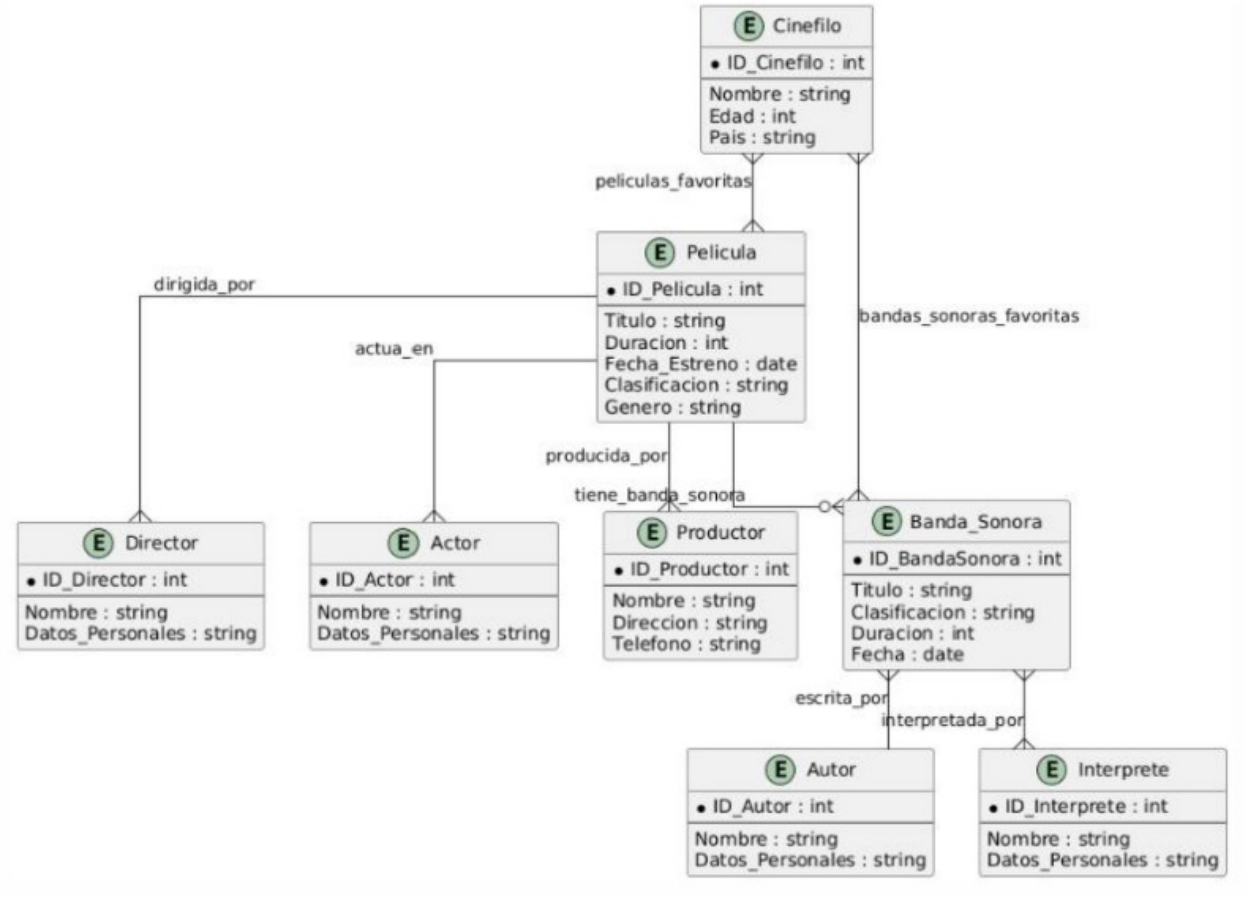


Figure 2: UML Diagram of the MyCineMusic Database

4 Query Space and Interface

Our implementation consists of a Java CRUD (Create, Read, Update, Delete) terminal application that interacts with a CockroachDB distributed database. This application demonstrates how vertical fragmentation works in a practical distributed database environment.

4.1 System Architecture

The application follows a three-tier architecture:

- **Presentation Layer:** A console-based user interface (ConsoleUI class) that handles user input and displays results.
- **Business Logic Layer:** Manages the application logic and data transformations (Movie class).
- **Data Access Layer:** Handles database connections and operations (DatabaseManager class).

4.2 Vertical Fragmentation Implementation

Our implementation achieves vertical fragmentation through CockroachDB's multi-region capabilities. The `pelicula` table is partitioned by the `region` column, which means that different parts of the movie data are physically stored in different geographic regions based on the region value.

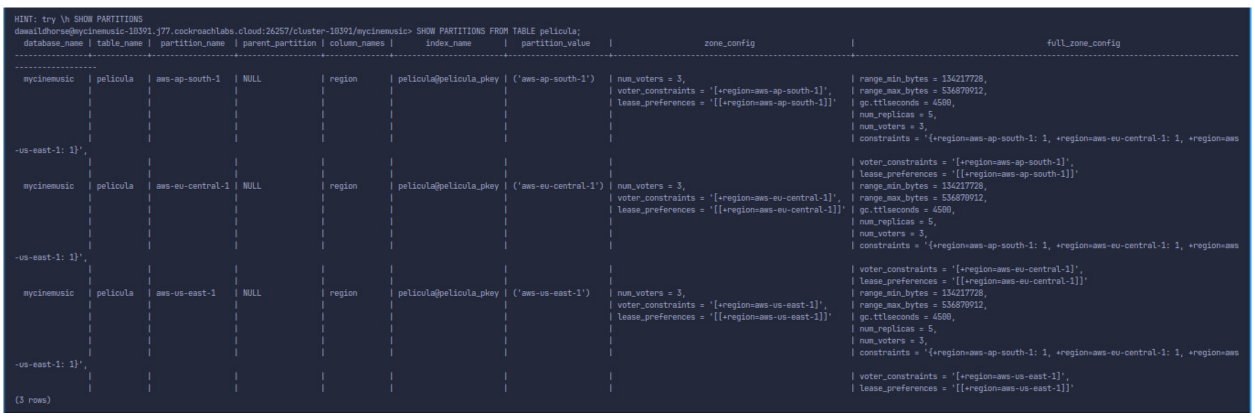
Listing 1: Table Schema with Region-Based Partitioning

```sql
CREATE TABLE pelicula (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  titulo STRING NOT NULL,
  duracion INTERVAL,
  fecha_estreno DATE,
  clasificacion STRING,
  region public.crdb_internal_region NOT NULL,
  created_at TIMESTAMP DEFAULT now()
);
```

The `region` column uses the special `public.crdb_internal_region` type, which CockroachDB uses to determine where to physically store the data. When a new movie record is created, the application assigns it to a specific region, and CockroachDB ensures that the data is stored in the corresponding geographic location.

This approach offers several advantages:

- **Data Locality:** Data is stored closer to where it's most frequently accessed, reducing latency.
- **Regulatory Compliance:** Certain data can be kept within specific geographic regions to comply with data sovereignty regulations.
- **Fault Tolerance:** The system remains operational even if one region experiences an outage.



HINT: try \h SHOW PARTITIONS
dawaildhorse@mycinemusic-10391.j77.cockroachlabs.cloud:26257/cluster-10391/mycinemusic> SHOW PARTITIONS FROM TABLE pelicula;

database_name	table_name	partition_name	parent_partition	column_names	index_name	partition_value	zone_config	full_zone_config
mycinemusic	pelicula	aws-ap-south-1	NULL	region	pelicula@pelicula_pkey	('aws-ap-south-1')	num_voters = 3, voter_constraints = '[+region=aws-ap-south-1]', lease_preferences = '[[+region=aws-ap-south-1]]'	range_min_bytes = 134217728, range_max_bytes = 536870912, gc.ttlseconds = 4500, num_replicas = 5, num_voters = 3, constraints = '{+region=aws-ap-south-1: 1, +region=aws-eu-central-1: 1, +region=aws-us-east-1: 1}', voter_constraints = '[+region=aws-ap-south-1]', lease_preferences = '[[+region=aws-ap-south-1]]'
mycinemusic	pelicula	aws-eu-central-1	NULL	region	pelicula@pelicula_pkey	('aws-eu-central-1')	num_voters = 3, voter_constraints = '[+region=aws-eu-central-1]', lease_preferences = '[[+region=aws-eu-central-1]]'	range_min_bytes = 134217728, range_max_bytes = 536870912, gc.ttlseconds = 4500, num_replicas = 5, num_voters = 3, constraints = '{+region=aws-ap-south-1: 1, +region=aws-eu-central-1: 1, +region=aws-us-east-1: 1}', voter_constraints = '[+region=aws-eu-central-1]', lease_preferences = '[[+region=aws-eu-central-1]]'
mycinemusic	pelicula	aws-us-east-1	NULL	region	pelicula@pelicula_pkey	('aws-us-east-1')	num_voters = 3, voter_constraints = '[+region=aws-us-east-1]', lease_preferences = '[[+region=aws-us-east-1]]'	range_min_bytes = 134217728, range_max_bytes = 536870912, gc.ttlseconds = 4500, num_replicas = 5, num_voters = 3, constraints = '{+region=aws-ap-south-1: 1, +region=aws-eu-central-1: 1, +region=aws-us-east-1: 1}', voter_constraints = '[+region=aws-us-east-1]', lease_preferences = '[[+region=aws-us-east-1]]'

(3 rows)

Figure 3: Fragmentation observed in CockrouchDB CLI

4.3 Query Interface

Our Java application provides a terminal-based interface for interacting with the distributed database. The interface offers the following operations:

1. **Add a new movie:** Creates a new movie record with title, duration, release date, classification, and region.
2. **View all movies:** Displays all movies stored in the database across all regions.
3. **View movie by ID:** Retrieves and displays a specific movie by its UUID.
4. **Update a movie:** Modifies the details of an existing movie.
5. **Delete a movie:** Removes a movie from the database.

6. **Show database partitions:** Displays information about how the data is partitioned across regions, demonstrating the vertical fragmentation.

7. **Check database connection:** Verifies the connection to the CockroachDB cluster.

4.4 Demonstrating Vertical Fragmentation

The application includes a special feature to visualize the vertical fragmentation through the "Show database partitions" option. This executes the `SHOW PARTITIONS FROM TABLE pelicula` command, which returns information about how the data is distributed across different regions.

The output shows that different rows are physically stored in different regions based on the `region` value, demonstrating vertical fragmentation in action. This approach allows for region-specific optimizations and ensures that data is stored close to where it's most frequently accessed.

5 Implementation in Java

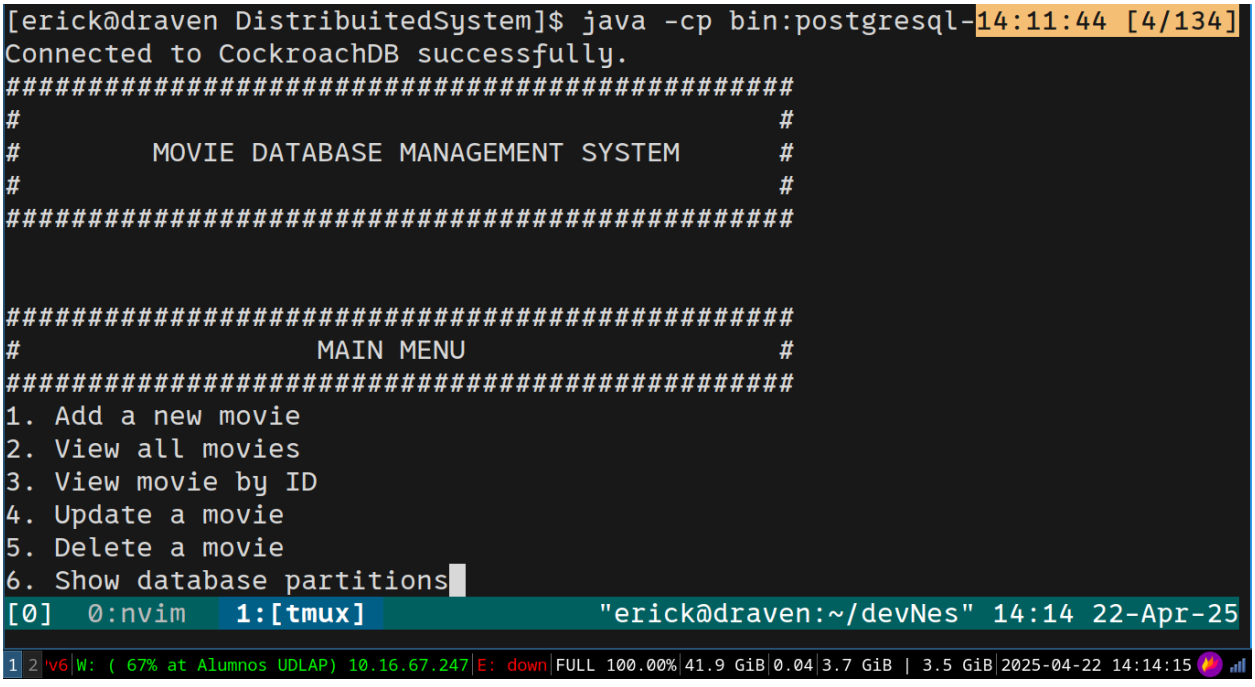


Figure 4: Java Implementation of the MyCineMusic Database, menu

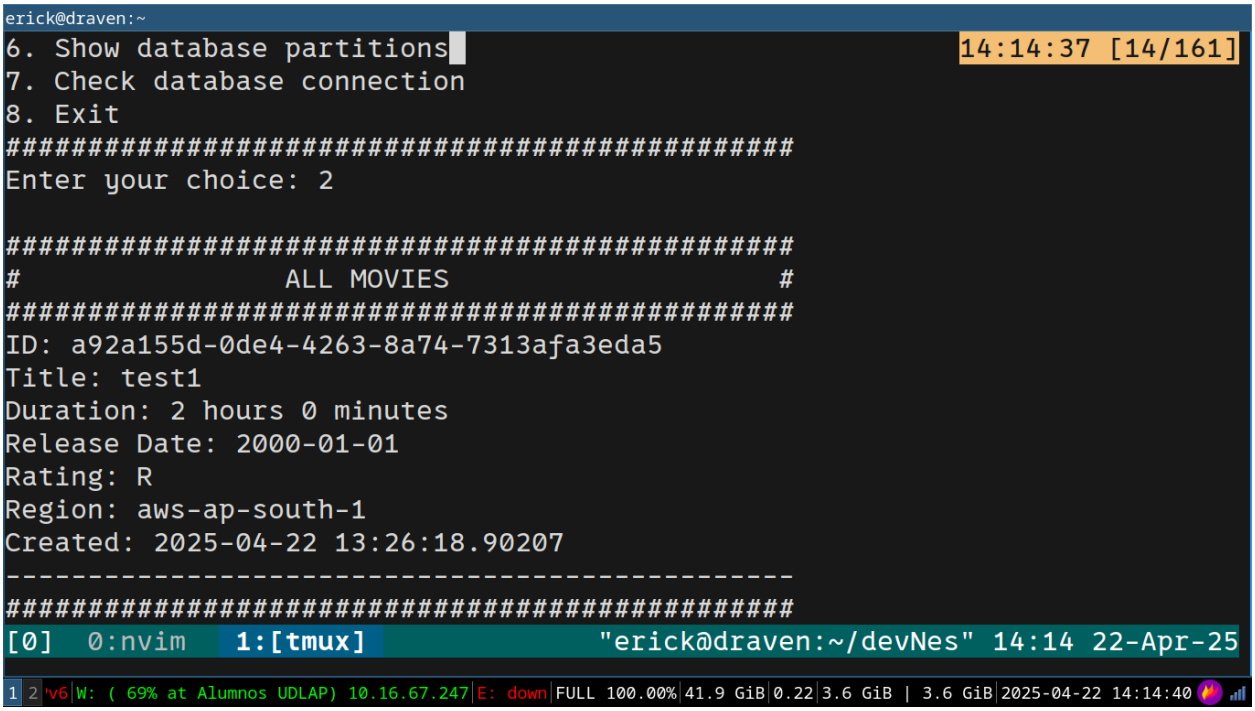


Figure 5: Showing movie data

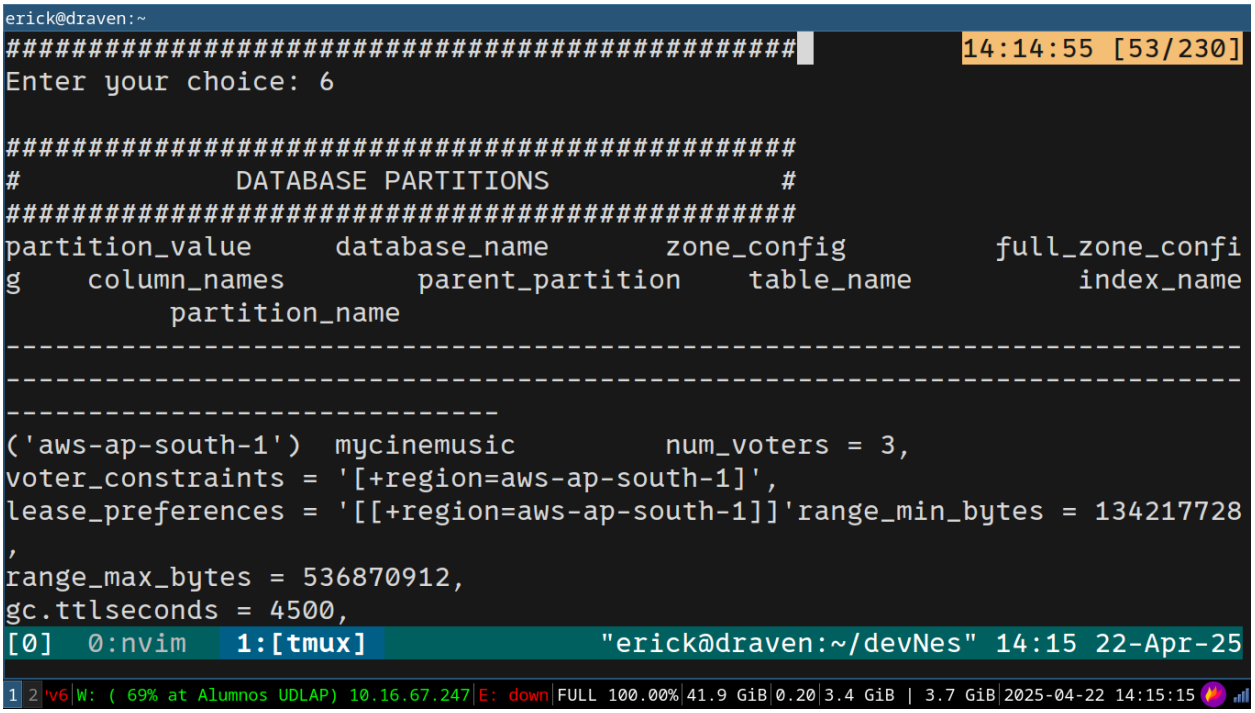


Figure 6: Showing fragmentation here too

6 Setting Up and Running the Software

This section provides detailed instructions for setting up and running the distributed database application. Following these steps will allow you to experience the vertical fragmentation implementation firsthand.

6.1 Prerequisites

Before you begin, ensure you have the following prerequisites installed:

- Java Development Kit (JDK) 21 or higher
- PostgreSQL JDBC Driver (version 42.7.5 or compatible)
- Git (for cloning the repository)
- Internet connection (for accessing the CockroachDB cluster)

6.2 Environment Setup

6.2.1 1. Clone the Repository

First, clone the repository to your local machine:

Listing 2: Cloning the Repository

```
git clone --recursive https://github.com/HugeErick/DistribuitedSystem.git
cd DistribuitedSystem
```

6.2.2 2. Download the CockroachDB Root Certificate

The application connects to a CockroachDB cluster using SSL. You need to download the root certificate to establish a secure connection.

For Unix/Linux/macOS:

Listing 3: Downloading Certificate on Unix/Linux/macOS

```
curl --create-dirs -o $HOME/.postgresql/root.crt 'https://cockroachlabs.cloud/clusters/690e
```

For Windows:

Listing 4: Downloading Certificate on Windows

```
# Create the directory if it doesn't exist
mkdir -p "$env:APPDATA\postgresql"

# Download the certificate
curl -o "$env:APPDATA\postgresql\root.crt" "https://cockroachlabs.cloud/clusters/690e5a03-
```

6.2.3 3. Set Up the Environment Variable

The application requires a JDBC connection string to connect to the CockroachDB cluster. Set up the environment variable as follows:

For Unix/Linux/macOS:

Listing 5: Setting Environment Variable on Unix/Linux/macOS

```
export JDBC_DATABASE_URL="jdbc:postgresql://free-tier14.aws-us-east-1.cockroachlabs.cloud:2
```

For Windows (Command Prompt):

Listing 6: Setting Environment Variable on Windows (CMD)

```
set JDBC_DATABASE_URL=jdbc:postgresql://free-tier14.aws-us-east-1.cockroachlabs.cloud:26257
```

For Windows (PowerShell):

Listing 7: Setting Environment Variable on Windows (PowerShell)

```
$env:JDBC_DATABASE_URL="jdbc:postgresql://free-tier14.aws-us-east-1.cockroachlabs.cloud:262
```

Replace `<username>` and `<password>` with the actual database credentials.

6.3 Compiling and Running the Application

6.3.1 1. Compile the Java Code

Compile the Java source files using the following command:
For Unix/Linux/macOS:

Listing 8: Compiling on Unix/Linux/macOS

```
javac -d bin -cp postgresql-42.7.5.jar src/*.java
```

For Windows:

Listing 9: Compiling on Windows

```
javac -d bin -cp postgresql-42.7.5.jar src\*.java
```

6.3.2 2. Run the Application

Run the compiled application using the following command:
For Unix/Linux/macOS:

Listing 10: Running on Unix/Linux/macOS

```
java -cp bin:postgresql-42.7.5.jar Main
```

For Windows:

Listing 11: Running on Windows

```
java -cp bin;postgresql-42.7.5.jar Main
```

6.4 Using the Application

Once the application is running, you will see the main menu with the following options:

1. Add a new movie
2. View all movies
3. View movie by ID
4. Update a movie
5. Delete a movie
6. Show database partitions
7. Check database connection
8. Exit

6.4.1 Adding a Movie

To add a new movie:

1. Select option 1 from the main menu
2. Enter the movie title
3. Enter the duration (format: 2h 30m or 150m)
4. Enter the release date (format: YYYY-MM-DD)
5. Enter the rating (e.g., PG-13, R)
6. Select a region from the available options

6.4.2 Viewing Vertical Fragmentation

To observe the vertical fragmentation in action:

1. Select option 6 from the main menu
2. The application will display the partition information, showing how data is distributed across different regions

6.5 Troubleshooting Common Issues

6.5.1 Connection Issues

If the application fails to connect to the database:

1. Verify that the `JDBC_DATABASE_URL` environment variable is correctly set
2. Check that the username and password in the connection string are correct
3. Ensure you have internet connectivity
4. Verify that the PostgreSQL JDBC driver is in the classpath

6.5.2 ClassNotFoundException

If you encounter a `ClassNotFoundException` for `org.postgresql.Driver`:

1. Ensure the PostgreSQL JDBC driver JAR file is in the same directory as your application
2. Verify that the JAR file is correctly included in the classpath when running the application

7 Conclusions

The design of the MyCineMusic database addresses the fundamental needs for managing multimedia data related to music and cinema. It includes both the conceptual and relational schemas and provides a clear interface for interacting with the system through a defined set of queries.

Our practical implementation using Java and CockroachDB demonstrates how vertical fragmentation can be achieved in a distributed database environment. By partitioning data by region, we can optimize data locality, improve performance, and ensure regulatory compliance.

The implementation showcases several key concepts in distributed database systems:

- Vertical fragmentation through region-based partitioning
- ACID-compliant distributed transactions

- SQL interface for interacting with distributed data
- Visualization of data distribution across regions

This approach provides a solid foundation for building scalable, resilient, and globally distributed database applications.

References

[1] CockroachDB. (2023). Multi-Region Capabilities. CockroachDB Documentation. Retrieved April 22, 2025, from `https://www.cockroachlabs.com/docs/stable/multiregion-overview.html`

[2] GeeksforGeeks. (2018, June 11). Distributed Database System. Retrieved from `https://www.geeksforgeeks.org/distributed-database-system/`

[3] MongoDB. (2023). What is a Distributed Database? MongoDB Documentation. Retrieved April 22, 2025, from `https://www.mongodb.com/resources/basics/databases/distributed-database`

[4] Özsu, M. T., & Valduriez, P. (2020). Principles of Distributed Database Systems (4th ed.). Springer.

[5] Date, C. J. (2003). An Introduction to Database Systems (8th ed.). Addison-Wesley.