CSC582

Data Warehouse and Mining Systems

Relational to Key-Value Database Migration

E-Commerce Database with Redis Cluster

Course:	CSC582 - Data Warehouse and Mining Systems
Project Type:	Relational to Key-Value Database Mapping
Database Domain:	E-Commerce (Customer, Product, Order)
Key-Value Store:	Redis Cluster
Mapping Format:	TableName:TupleID:Attribute $\rightarrow$ Value
Cluster Config:	4 Master Nodes + 4 Replica Nodes
Date:	December 2024

Table of Contents

1. Introduction

This project demonstrates the migration of a relational E-Commerce database to Redis, a key-value NoSQL database. The project implements the mapping format **TableName:TupleID:Attribute** $\rightarrow$ **Value** where each column in a relational row becomes a separate key-value pair in Redis. The Redis Cluster is configured with 8 nodes (4 masters + 4 replicas) to demonstrate both **sharding** (data distribution) and **replication** (high availability).

2. Relational Model and Schema

2.1 Entity-Relationship Diagram

```
+-------------------------+         +-------------------------+
|        CUSTOMER         |         |         PRODUCT         |
+-------------------------+         +-------------------------+
| PK customer_id          |         | PK product_id           |
|    first_name           |         |    product_name         |
|    last_name            |         |    category             |
|    email (UNIQUE)       |         |    price                |
|    phone                |         |    stock_quantity       |
|    city                 |         +-------------------------+
|    country              |
+-------------------------+
          |
          | 1:M (One customer has many orders)
          v
+-------------------------+
|         ORDER           |
+-------------------------+
| PK order_id             |
| FK customer_id          |
|    order_date           |
|    status               |
|    total_amount         |
+-------------------------+
```

2.2 Sample Data

Customer Table (5 records):

ID	First Name	Last Name	Email	City
1	Ahmed	Al-Rashid	ahmed.rashid@email.com	Riyadh
2	Fatima	Hassan	fatima.hassan@email.com	Jeddah
3	Mohammed	Al-Saud	mohammed.saud@email.com	Dammam
4	Sara	Abdullah	sara.abdullah@email.com	Riyadh
5	Khalid	Omar	khalid.omar@email.com	Mecca

Product Table (6 records):

ID	Product Name	Category	Price	Stock
101	Laptop Pro 15	Electronics	4500	25
102	Wireless Mouse	Electronics	150	100
103	Office Chair	Furniture	850	30
104	Standing Desk	Furniture	2200	15
105	Headphones	Electronics	1200	50
106	USB-C Hub	Electronics	280	75

Order Table (7 records):

Order ID	Customer	Date	Status	Total

1001	1	2024-06-01	delivered	4650
1002	2	2024-06-05	delivered	3050
1003	1	2024-06-10	shipped	1200
1004	3	2024-06-15	processing	2480
1005	4	2024-06-20	pending	850
1006	5	2024-06-25	delivered	4780
1007	2	2024-07-01	shipped	430

3. Mapping Strategy: TableName:TupleID:Attribute

3.1 Key Format

Each column in a relational row becomes a **separate key-value pair** in Redis. **Format:** TableName:TupleID:Attribute → Value This format allows: - Direct access to any attribute: GET Customer:1:email - Pattern-based queries: KEYS Customer:*:email (find all emails) - Clear identification of table, row, and column

3.2 Customer Table Mapping Example

```
Relational Row:
+-------------+------------+-----------+------------------------+---------------+
| customer_id | first_name | last_name | email                  | city          |
|      1      | Ahmed      | Al-Rashid | ahmed.rashid@email.com | Riyadh        |
+-------------+------------+-----------+------------------------+---------------+

                              ↓ MAPPING ↓

Key-Value Pairs in Redis:
+--------------------------------+----------------------------+
| KEY                            | VALUE                      |
+--------------------------------+----------------------------+
| Customer:1:first_name          | "Ahmed"                    |
| Customer:1:last_name           | "Al-Rashid"                |
| Customer:1:email               | "ahmed.rashid@email.com"   |
| Customer:1:phone               | "+966501234567"            |
| Customer:1:city                | "Riyadh"                   |
| Customer:1:country             | "Saudi Arabia"             |
+--------------------------------+----------------------------+
```

3.3 Product Table Mapping Example

```
Relational Row:
+------------+-----------------+-------------+-------+----------------+
| product_id | product_name    | category    | price | stock_quantity |
|    101     | Laptop Pro 15   | Electronics | 4500  |       25       |
+------------+-----------------+-------------+-------+----------------+

                              ↓ MAPPING ↓

Key-Value Pairs in Redis:
+--------------------------------+----------------------------+
| KEY                            | VALUE                      |
+--------------------------------+----------------------------+
| Product:101:product_name       | "Laptop Pro 15"            |
| Product:101:category           | "Electronics"              |
| Product:101:price              | "4500.0"                   |
| Product:101:stock_quantity     | "25"                       |
+--------------------------------+----------------------------+
```

3.4 Order Table Mapping Example

```
Relational Row:
+----------+-------------+------------+-----------+--------------+
| order_id | customer_id | order_date | status    | total_amount |
|   1001   |      1      | 2024-06-01 | delivered |     4650     |
+----------+-------------+------------+-----------+--------------+

                              ↓ MAPPING ↓

Key-Value Pairs in Redis:
+--------------------------------+----------------------------+
| KEY                            | VALUE                      |
+--------------------------------+----------------------------+
| Order:1001:customer_id         | "1"                        |
| Order:1001:order_date          | "2024-06-01"               |
| Order:1001:status              | "delivered"                |
| Order:1001:total_amount        | "4650.0"                   |
+--------------------------------+----------------------------+
```

4. Redis Cluster Configuration

4.1 Cluster Architecture (8 Nodes)

```
+-------------+  +-------------+  +-------------+  +-------------+
|  MASTER 0   |  |  MASTER 1   |  |  MASTER 2   |  |  MASTER 3   |
|  Port 7000  |  |  Port 7001  |  |  Port 7002  |  |  Port 7003  |
| Slots 0-4095|  |Slots 4096-  |  |Slots 8192-  |  |Slots 12288- |
|             |  |  8191       |  |  12287      |  |  16383      |
+------+------+  +------+------+  +------+------+  +------+------+
       |                |                |                |
       | Replication    | Replication    | Replication    | Replication
       v                v                v                v
+-------------+  +-------------+  +-------------+  +-------------+
|  REPLICA 0  |  |  REPLICA 1  |  |  REPLICA 2  |  |  REPLICA 3  |
|  Port 7004  |  |  Port 7005  |  |  Port 7006  |  |  Port 7007  |
+-------------+  +-------------+  +-------------+  +-------------+
```

4.2 Hash-Based Partitioning

Redis Cluster uses **CRC16 hash function** to determine which slot (and thus which master node) stores each key: **Formula:** slot = CRC16(key) mod 16384 Total slots: 16384 (distributed across 4 master nodes)

Master Node	Port	Slot Range	Replica Port
Master 0	7000	0 - 4095	7004
Master 1	7001	4096 - 8191	7005
Master 2	7002	8192 - 12287	7006
Master 3	7003	12288 - 16383	7007

5. Sharding Demonstration

5.1 How Sharding Works

Sharding distributes data across multiple master nodes. Each key is assigned to a **hash slot** (0-16383), and each slot belongs to a specific master node.

```
Example: Key "Customer:1:first_name"

Step 1: Calculate hash
        CRC16("Customer:1:first_name") = 5855

Step 2: Calculate slot
        slot = 5855 mod 16384 = 5855

Step 3: Find node (slot 5855 is in range 4096-8191)
        → Stored on MASTER 1 (Port 7001)
```

5.2 Key Distribution Across Nodes

Key	Hash Slot	Master Node
Customer:1:first_name	5855	Master 1 (7001)
Customer:2:email	11008	Master 2 (7002)
Product:101:product_name	9352	Master 2 (7002)
Product:103:price	12604	Master 3 (7003)
Order:1001:status	16118	Master 3 (7003)
Order:1005:total_amount	13005	Master 3 (7003)

```
Verification Commands:

redis-cli -p 7000 CLUSTER KEYSLOT "Customer:1:first_name"
> (integer) 5855

redis-cli -p 7000 CLUSTER SLOTS
> Shows which node owns which slot range

This proves keys are distributed across different master nodes based on hash!
```

6. Replication Demonstration

6.1 How Replication Works

Each master node has a **replica (slave)** node that maintains an exact copy of the master's data. This provides: - **High Availability:** If master fails, replica is promoted automatically - **Read Scaling:** Read operations can be served by replicas - **Data Safety:** Data is stored in multiple locations

6.2 Write to Master, Read from Replica

```
DEMONSTRATION: Write to Master 0, Read from Replica 0
=====================================================

Step 1: WRITE to Master 0 (Port 7000)
-------------------------------------
redis-cli -p 7000 SET "Customer:1:first_name" "Ahmed"
> OK

Step 2: Verify on Master 0
--------------------------
redis-cli -p 7000 GET "Customer:1:first_name"
> "Ahmed"

Step 3: READ from Replica 0 (Port 7004)
---------------------------------------
redis-cli -p 7004 READONLY
> OK

redis-cli -p 7004 GET "Customer:1:first_name"
> "Ahmed"

SUCCESS! Data written to master is automatically replicated to replica!
```

6.3 Replication Verification

```
Check Replication Status:

redis-cli -p 7000 INFO replication
> role:master
> connected_slaves:1
> slave0:ip=127.0.0.1,port=7004,state=online

redis-cli -p 7004 INFO replication
> role:slave
> master_host:127.0.0.1
> master_port:7000
> master_link_status:up

This proves replication is working - Replica 7004 is synced with Master 7000!
```

6.4 Failover Scenario

If Master 0 (Port 7000) fails: 1. Redis Cluster detects the failure 2. Replica 0 (Port 7004) is automatically promoted to master 3. Cluster continues to operate without data loss 4. Slots 0-4095 are now served by the new master (Port 7004) This automatic failover ensures **high availability** of the system.

7. Data Migration

7.1 Migration Process

The migration from relational database to Redis follows these steps: **Step 1:** Connect to source relational database (SQLite) **Step 2:** For each table, query all records **Step 3:** For each row, create key-value pairs using format: TableName:TupleID:Attribute → Value **Step 4:** Execute SET command for each key-value pair **Step 5:** Verify data integrity

7.2 Migration Commands

```
# Customer Migration
SET Customer:1:first_name "Ahmed"
SET Customer:1:last_name "Al-Rashid"
SET Customer:1:email "ahmed.rashid@email.com"
SET Customer:1:phone "+966501234567"
SET Customer:1:city "Riyadh"
SET Customer:1:country "Saudi Arabia"

# Product Migration
SET Product:101:product_name "Laptop Pro 15"
SET Product:101:category "Electronics"
SET Product:101:price "4500.0"
SET Product:101:stock_quantity "25"

# Order Migration
SET Order:1001:customer_id "1"
SET Order:1001:order_date "2024-06-01"
SET Order:1001:status "delivered"
SET Order:1001:total_amount "4650.0"
```

7.3 Total Keys Created

Table	Records	Attributes per Record	Total Keys
Customer	5	6	30
Product	6	4	24
Order	7	4	28
TOTAL	**18**	**-**	**82**

8. Key-Value Operations

8.1 GET by Key (Direct Lookup)

```
# Get specific attribute
GET Customer:1:first_name
> "Ahmed"

GET Customer:1:email
> "ahmed.rashid@email.com"

GET Product:101:price
> "4500.0"

GET Order:1001:status
> "delivered"
```

8.2 GET by Value (Pattern Matching)

```
# Find all emails
KEYS Customer:*:email
> 1) "Customer:1:email"
> 2) "Customer:2:email"
> 3) "Customer:3:email"
> 4) "Customer:4:email"
> 5) "Customer:5:email"

# Find all prices
KEYS Product:*:price
> 1) "Product:101:price"
> 2) "Product:102:price"
> ...

# Find all order statuses
KEYS Order:*:status
> 1) "Order:1001:status"
> 2) "Order:1002:status"
> ...
```

8.3 Simulating JOIN Operation

```
# Get Order 1001 with Customer Name (simulating SQL JOIN)

Step 1: GET Order:1001:customer_id
> "1"

Step 2: GET Customer:1:first_name
> "Ahmed"

Step 3: GET Customer:1:last_name
> "Al-Rashid"

Result: Order 1001 belongs to customer "Ahmed Al-Rashid"
```

9. Conclusion

This project successfully demonstrated the migration of a relational E-Commerce database to Redis key-value store with the following achievements: **1. Mapping Strategy:** Implemented TableName:TupleID:Attribute → Value format where each relational column becomes a separate key-value pair. **2. Sharding:** Demonstrated hash-based partitioning using CRC16 algorithm to distribute 82 keys across 4 master nodes. Verified that different keys are stored on different nodes based on their hash slot. **3. Replication:** Configured 4 replica nodes and demonstrated that data written to a master node is automatically replicated to its replica. Verified read operations from replica nodes return the same data as master nodes. **4. Key-Value Operations:** Demonstrated GET by key (direct lookup), GET by value (pattern matching with KEYS command), and JOIN simulation using multiple GET operations. **5. High Availability:** The 8-node cluster architecture (4 masters + 4 replicas) ensures data is never lost even if a master node fails, as replicas can be automatically promoted to masters.

Component	Details
Database	E-Commerce (Customer, Product, Order)
Total Records	18 (5 + 6 + 7)
Mapping Format	TableName:TupleID:Attribute → Value
Total Keys	82
Cluster Nodes	8 (4 Masters + 4 Replicas)
Hash Slots	16384 (4096 per master)
Sharding	CRC16 hash-based partitioning
Replication	Each master has 1 replica