A Detailed Analysis of Backup and Recovery Techniques in Database Management Systems (DBMS)

V.E. Sathishkumar[1,*], Ng Soon Teck[1], Ng Wen Chao[1] and Samuel Lee Zhi Jian[1]

[1]*School of Engineering and Technology, Sunway University, No. 5, Jalan University, Bandar Sunway, 47500, Selangor Darul Ehsan, Malaysia*

Abstract: In this exponential growth age and frequent disruption, data has to be really secure, integral, and accessible. DBMS plays a keynote in all these aspects. This report reviews the backup and recovery techniques of four DBMSs: Cassandra, MongoDB, Neo4j, and Oracle. It discusses their efficacy in restoring data loss and maintaining schema integrity. This study throws light on the challenge of complete data and schema recovery considering operational nuances and system-specific requirements. The recovery processes were recreated in the controlled environments by using hands-on experimentation. Recovery percentages for each DBMS were computed from recovered data. The results of the study indicate that 100% success rate for all systems, which means these systems do not fail in case of a disaster. These results give critical insight into the reliability and application-specific advantages of each DBMS in guiding database administrators in the design of resilient systems.

Keywords: Backup, Recovery, Cassandra, MongoDB, Neo4j, Oracle.

1. INTRODUCTION

A. General Background

These are important to manage a huge volume of data generated in various industries such as e-commerce, finance, health care, and many more with the help of DBMS [1]. Advanced DBMS nowadays provides necessary tools for maintaining data integrity and availability to handle ever-increasing complexity because of distributed and cloud-based architectures [2]. Such systems are helping organizations to meet challenges related to big data and real-time analytics, which are becoming very important parts of decision-making [3].

In turn, the use of distributed systems brought more sophisticated mechanisms of data backup; for example, the snapshot-based backups have enabled Cassandra to bring fault tolerance into scalable environments [4]. Among popular NoSQL databases, MongoDB makes use of the recovery mechanisms grounded on optimized BSON for recovering semi-structured data, but significant challenges arise if datasets include deeply nested relationships [5]. Neo4j is a graph database with unrivaled capabilities in relationship-driven applications but requires specialized recovery strategies to be devised for making sure consistency is guaranteed upon restoration [6]. Relational DBMS like Oracle are strong in guaranteeing the integrity of the schemas, using advanced means for the recovery of data and schema, and have become very important in large enterprise deployments [7]. The COVID-19 pandemic highlighted the significance of resilient DBMS systems, with increased reliance on e-commerce and cloud-based applications exposing vulnerabilities in existing recovery mechanism [8]. Advanced recovery strategies that integrate automation, machine learning, and elastic networking are now essential to addressing the growing complexities of modern data environments [9].

B. Problem Statement

While DBMS technologies have evolved, recovery across platforms remains efficient and reliable, which is still a challenge to achieve [10]. Distributed systems, such as Cassandra, require very accurate node synchronization during restoration for consistency, especially in geographically distributed installations [4]. MongoDB uses BSON-based backups, which ease the storage but complicate the recovery process when datasets with complex relational interdependencies are handled [5]. Recovery processes in Neo4j need to ensure that relationships, which are at the core of graph-based analytics, remain intact [6].

While Oracle's advanced schema recovery tools are indeed powerful for recovering structured data, because of their complexity, they often require specialized skills, especially for very large applications [7]. Recovery processes in the cloud have to handle latency challenges in integration with real-time workflows for minimal disruption [11]. The pandemic underlined the scalability issue of recovery solutions, as many organizations found themselves unable to meet the heightened demands for data during the crisis [8].

C. Significance of the Problem

This results in massive financial losses, disruption of operations, and loss of reputation for organizations,

*Address correspondence to this author at the School of Engineering and Technology, Sunway University, No. 5, Jalan University, Bandar Sunway, 47500, Selangor Darul Ehsan, Malaysia; E-mail: sathishv@sunway.edu.my

Received on 28-11-2025 Accepted on 29-12-2025 Published on 31-12-2025

particularly those in data-intensive industries [1]. The COVID-19 pandemic placed unprecedented pressures on organizational data systems, while e-commerce platforms and remote operations increased stress on recovery infrastructures [8]. The latest developments in elastic recovery and AI-driven monitoring systems have already shown their promise in reducing restoration times and ensuring data resilience [9].

However, on the other hand, laws such as GDPR and HIPAA also demand huge recovery strategies in order to keep them compliant and avoid fines in case of data breach or downtime [12]. Meeting these challenges will definitely assure the sustainability of an organization in this digital era [13].

D. Objective

This report aims to:

- Backup and Restore Approaches: Document and confirm the approaches to backup and restore for Cassandra, MongoDB, Neo4j, and Oracle.
- Prudentially Specifying Strengths and Weaknesses: Appraise several practical advantages and weaknesses found in the experimental implementations.
- Practical Insights: Explain the efficiency in recovering data, the integrity of schema as well as the usability for disaster recovery planning decision making.

E. Scope of the Report

The results presented in this report come from controlled experiments on different DBMS types, which includes Cassandra, MongoDB, Neo4j, and Oracle, reporting the obtained recovery percentages, schema integrity, and system-specific challenges. This work will compile experimental results and related literature to provide some guidelines on how to enhance the recovery strategy of new DBMSs' distributed, graph-based, and cloud-integrated platforms.

2. LITERATURE REVIEW

A. Backup and Recovery in Cloud and Distributed Systems

Cloud-based systems introduce scalable and economical solutions for managing backup and recovery that enables an organization to continue its operations even during disruptions [1]. Singh and Battra highlighted the role of an automated recovery process, reducing human interference and making the process of disaster recovery smooth and seamless [1]. Javaraiah identified some advantages of cloud-based backup, especially for small and medium-size enterprises in terms of resource optimization [3].

Distributed systems like Cassandra use snapshot-based mechanisms to ensure scalability and fault tolerance. However, coordinating multi-node restoration remains an open challenge [4]. Cassandra uses snapshot-based backups for handling geographic node distribution quite efficiently, though coordinating multi-node restorations remains a challenge in case the system fails [14]. Bohora *et al*. identified the key to synchronization for maintaining data consistency in a distributed system, which is monitoring tools [4]. Advanced Elastic inter-data-center networks provide solutions for minimizing latency and enhancing recovery time in the case of large-scale migration, as demonstrated by Lu *et al*. [9].

B. MongoDB and Oracle Recovery Mechanisms

The BSON-based MongoDB system enhances recovery simplicity in semi-structured data environments but imposes complexity in managing datasets with complex interdependencies [5]. In this perspective, Kuyumdzhiev analyzed how nested relationships can further complicate the restoration process; such restoration requires additional configuration settings to achieve full recovery [5]. Preliminary attempts at hybrid approaches involving MongoDB and relational databases such as MySQL proved better transactional performance and efficiency in recovery [13].

Advanced schema integrity and data restoration tools in Oracle make it a preferred choice for use within structured environments, especially in enterprise-scale applications [7]. Choi *et al*. investigated Oracle's effectiveness in restoring deleted records and asserted its robustness in data integrity while restoring it to its original form [7]. According to Tiwari, regular testing is an essential attribute in optimizing Oracle's backup strategies, especially for large-scale deployment environments [15]. Comparison studies have also demonstrated that, although MongoDB outperforms the recovery process for semi-structured data, Oracle remains the best choice regarding robustness in relational data recovery [16].

C. Advances in Recovery Techniques

Emerging recovery strategies are focused on automation, proactive resource management, and hence reducing delays as much as possible in bringing the service back online. Kim *et al*. have researched write-ahead logging methods to maintain consistency in a transactional distributed system, hence reducing recovery time [17]. Ramesh *et al*. propose a machine learning-based model for monitoring databases that can dynamically allocate resources for better reliability and scalability in recovery [2].

Hasan *et al*. have identified the elasticity of cloud-based recovery systems during the COVID-19 pandemic. It shows how cloud-based recovery systems can keep data continuous even under extreme conditions [8]. Big data applications require elasticity in their recovery strategies due to latency problems in real-time analytics and IoT systems, to which it definitely contributes to system reliability [9].

D. Graph Databases and Neo4j-Specific Recovery

Graph databases, such as Neo4j, have recovery strategies that maintain relationships and metadata that form part of their intrinsic working [6]. Sauer *et al*. introduced metadata preservation methods that allow keeping the consistency of relationships during restoration; this is critical for applications such as recommendation systems and network analysis [18]. The performance analysis of Neo4j by Varghese *et al*. demonstrated that keeping the relationship consistency is vital for real-time analytics [6].

The recovery problem is no different in main-memory databases, where the majority of efforts are put into reducing downtime using memory-optimized techniques that provide reliability and consistency [19]. These techniques provide an important hint at how recovery can be improved in highly connected graph-based systems.

E. Specialized Systems and NoSQL Recovery

NoSQL systems, such as Cassandra and MongoDB, require customized recovery strategies due to their schema-less nature and distributed architecture [4, 5]. Kathpal and Sehgal presented the BARNS, a scalable framework for NoSQL database recovery, and highlighted its potential to facilitate disaster recovery simplification in distributed systems [12]. Krstić and Krstić provided an overview of performance benchmarking tools that provide valuable insights into optimizing recovery strategies across different DBMSs [20].

F. SUMMARY

Variations of the recovery approaches were found in the analysis of related literature. Differences in architectures and requirements of their operations mean distributed systems require scalability while graph and NoSQL databases require relationship and scalable data handling techniques. Promising approaches such as AI-based monitoring, elastic recovery, and write-ahead logging have been proposed from these premises to handle these challenges. Following these premises, the study experimentally investigates backup and recovery mechanisms for Cassandra, MongoDB, Neo4j, and Oracle and offers practical recommendations.

3. RESEARCH METHODOLOGY

A. Introduction to the Methodology

The detailed and organized procedure that follows was undertaken to assess the applicability of backup and recovery techniques in four well-recognized Database Management Systems: namely, Cassandra, MongoDB, Neo4j, and Oracle. The methodology that will be followed targets implementation and validation whereby the prime focus is toward measuring recovered data percentage and restored schema percentage under controlled experimental settings. The relating strengths and limitations and procedural understanding propose a framework for DBMS studies that is replicable.

B. Research Design

The concept of the research study follows an experimental design whereby the actual implementation of realistic backup and recovery procedures is performed. This design encompasses:

- Quantitative Analysis: Measurement of the percent recovery for both data and schemas.
- Qualitative Observations: Identification of strengths and limitations that one comes across in executing the procedure.

Each of the DBMS was tested in an individual environment to minimize any disturbance from outside factors and to ensure that consistency would run throughout the experiments. Results have been drawn by comparing, step by step, restored data with that of original datasets.

C. Steps Involved in the Methodology

1) Step 1: Identification of Research Objectives

The major aims of this study are:

a) To implement and validate the efficiency of the backup and recovery methods to Cassandra, MongoDB, Neo4j, and Oracle.

b) Recovery rate for each DBMS. All rows, columns, and schema attributes must be recovered.

c) Benefits and shortcomings in the approach of recoveries in every DBMS

2) Step 2: Literature Study and Review

In this paper, an extensive review of the literature is done to find the standard of practices and optimal ways of performing backup and recovery in the DBMSs under consideration. Also, technical documentation and official documents and academic research work have been explored to get specific further information, which shall serve to form the basis of comparison. The regular problems related to recoveries concerning handling massive data or schema integrity, etc., are reviewed too

3) Step 3: Evaluation Criteria

The evaluations shall be done based on two major parameters:

a) Data Completeness: these refer to the records recovered against original records present in data. This could be worked out using a formula as given below:

Recovery Percentage (Data) = Original Records / Records Restored X 100

b) Schema Integrity: the restored database schemas key constraints, relationships are compared with the original one.

4) Step 4: Data Collection

Experimental datasets for each DBMS were developed to simulate real-world use cases:

- Cassandra: A playlist table with 10 rows with composite keys, such as ID and song order.
- MongoDB: A JSON dataset with 100 customer records, including fields such as name, email, and orders.
- Neo4j: A graph database that contains 10 nodes & 6 relationships, all divided into three categories: friends, family, colleagues.
- Oracle: A books table with 10 rows, having attributes such as book ID, title of the book, author, genre, year of publication.

5) Step 5: Experimental Setup

- Each DBMS was deployed in a controlled environment using the platform-specific tool:
- Cassandra: The DBMS is deployed on MacOS, and nodetool is configured for snapshots.
- MongoDB: MongoDB is running on windows, and execute mongodump and mongorestore.
- Neo4j: Neo4j is installed on Windows using Cypher queries and neo4j-admin for all the database operations.
- Oracle: Oracle is deployed on Windows using SQL*Plus and Data Pump utilities (expdp and impdp).

6) Step 6: Implementation of Solution

Each DBMS did follow the same procedure for creating a backup and recovery, namely

a) Backup Creation

- Cassandra: The Snapshots were created using the nodetool snapshot command.
- MongoDB: Utilized mongodump to take a backup of the data in the BSON files.
- Neo4j: Utilized neo4j-admin to backup the nodes, their relationships, and indexes.
- Oracle: Using the expdp command, exported books table to a dump file along with all info defining it’s metadata

b) Data Loss Simulation

Simulated data loss by dropping database or tables using like commands:

- *DROP DATABASE* - for MongoDB & Neo4j
- *DROP TABLE* – for Oracle
- *DROP KEYSPACE* – for Cassandra.

c) Data Recovery

- Cassandra: Recreated keyspace along with the table and restored all the data by using snapshot files.
- MongoDB: Database was rebuilt through mongorestore out of the BSON Files.
- Neo4j: Restored the Neo4j DB through neo4j-admin & neo4j-admin register-system-database true
- Oracle: impdp was used to import the books table along with the Schema from the dumped file of the Table.

d) Dataset Validation & Verification

- Wrote queries on the data that were recovered to check the restored dataset have the same stuff as the old one.
- Compared schema definitions—verify data Integrity—all data, primary keys constraints, and relation.

D. Tools and Technologies Used

Following are some of the utilities that supported various Backup and Recover Techniques:

- Cassandra: nodetool, to make and restore snapshots
- MongoDB: mongodump and mongorestore—the export and import of data.
- Neo4j: Cypher query language and neo4j-admin for the management of database backups
- Oracle: SQL*Plus for database operations and Datapump utilities for the export and import of the database

E. Flow Diagram

Further, the methodology was designed to work in the following sequence of steps:

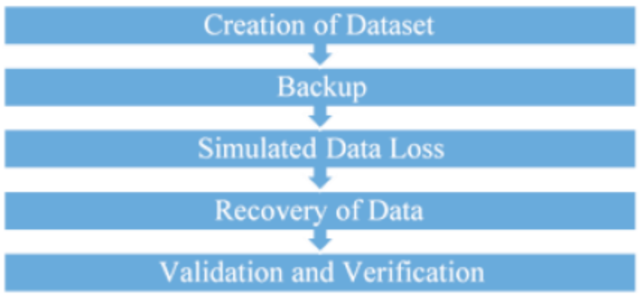


Figure 1: Data Backup and Recovery Process Flow.

F. Challenges and Limitations

a) Challenges:

- System-specific utility programs: Each DBMS used its utility programs and sequences of instructions. Hence, the complexity of the implementation was more.
- Synthetic Data: Controlled datasets cannot exhibit realistic variability and challenges
- Setup Environment: Setting up utilities and their compatibility with the experimental environment consumed a lot of time and effort

b) Limitations:

- Scalability: Since this research has targeted smaller datasets, it cannot be said to represent how the backup and recovery techniques shall work over much larger or distributed databases.
- Homogeneity: Operating conditions are homogeneous and hence cannot reflect the diverse environments in which DBMSs operate.

G. Summary

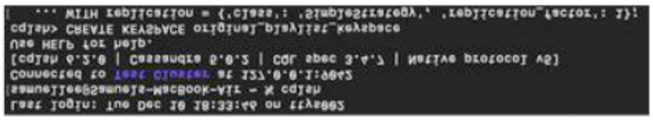
It will give a clear and elaborated structure for the implementation and analysis to be performed by Cassandra, MongoDB, Neo4j, and Oracle for their respective creation, backup, and recovery procedures. This step-by-step process will provide an effective measurement regarding the recovery percentage and schema integrity of the strengths and challenges that each system faces. The results would present a tangible basis in understanding reliability in recovery techniques of DBMSs and optimization of disaster recovery strategies.

4. RESULTS AND DISCUSSION

In this section, we describe the results of our experiment on the implementation of backup and recovery techniques in Cassandra, MongoDB, Neo4j and Oracle systems. It assesses the efficacy of these processes, measured by recovery percentages and the integrity of the schema, and then conducts a comparative study of the pros and cons noted during real-world implementation. These results intend to prove the robustness of those systems in cases of loss of data and to provide important aspects regarding database structure, data reliability, and scalability.

A. Cassandra

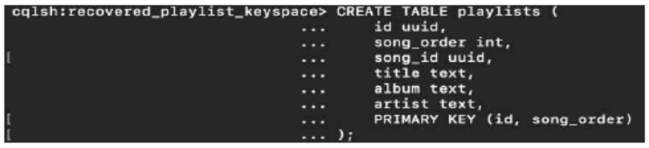
1) Data Creation



```
Last login: Tue Dec 16 18:33:46 on ttys002
samuellee@Samuels-MacBook-Air ~ % cqlsh
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.2.0 | Cassandra 5.0.2 | CQL spec 3.4.7 | Native protocol v5]
Use HELP for help.
cqlsh> CREATE KEYSPACE original_playlist_keyspace
   ... WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

Figure 2: Keyspace Creation in Cassandra.

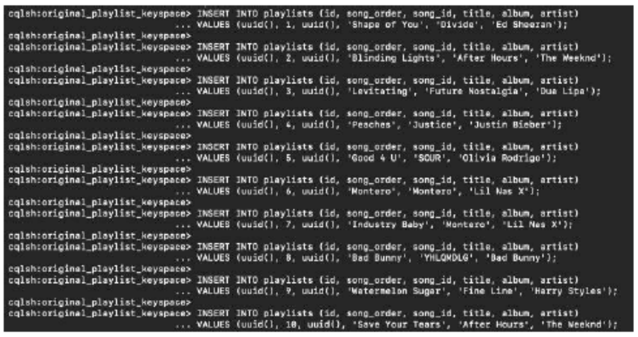
Figure **2** shows the process of creating a keyspace named original_playlist_keyspace with a simple replication strategy and replication factor of 1. Hence, the keyspace is added to the database. Then, switch to the original_playlist_keyspace keyspace for subsequent operations.



```
cqlsh:recovered_playlist_keyspace> CREATE TABLE playlists (
                               ...     id uuid,
                               ...     song_order int,
                               ...     song_id uuid,
                               ...     title text,
                               ...     album text,
                               ...     artist text,
                               ...     PRIMARY KEY (id, song_order)
                               ... );
```

Figure 3: Table Creation in Cassandra.

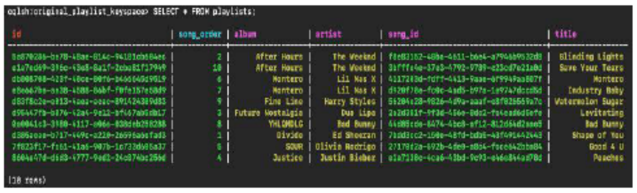
Based on Figure **3**, it defines a table named playlists with id and song_order as the composite primary key. The table structure is set for storing playlist information.



```
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 1, uuid(), 'Shape of You', 'Divide', 'Ed Sheeran');
cqlsh:original_playlist_keyspace>
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 2, uuid(), 'Blinding Lights', 'After Hours', 'The Weeknd');
cqlsh:original_playlist_keyspace>
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 3, uuid(), 'Levitating', 'Future Nostalgia', 'Dua Lipa');
cqlsh:original_playlist_keyspace>
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 4, uuid(), 'Peaches', 'Justice', 'Justin Bieber');
cqlsh:original_playlist_keyspace>
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 5, uuid(), 'Good 4 U', 'SOUR', 'Olivia Rodrigo');
cqlsh:original_playlist_keyspace>
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 6, uuid(), 'Montero', 'Montero', 'Lil Nas X');
cqlsh:original_playlist_keyspace>
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 7, uuid(), 'Industry Baby', 'Montero', 'Lil Nas X');
cqlsh:original_playlist_keyspace>
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 8, uuid(), 'Bad Bunny', 'YHLQMDLG', 'Bad Bunny');
cqlsh:original_playlist_keyspace>
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 9, uuid(), 'Watermelon Sugar', 'Fine Line', 'Harry Styles');
cqlsh:original_playlist_keyspace>
cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)
                              ... VALUES (uuid(), 10, uuid(), 'Save Your Tears', 'After Hours', 'The Weeknd');
```

Figure 4: Data Insertion in Cassandra.

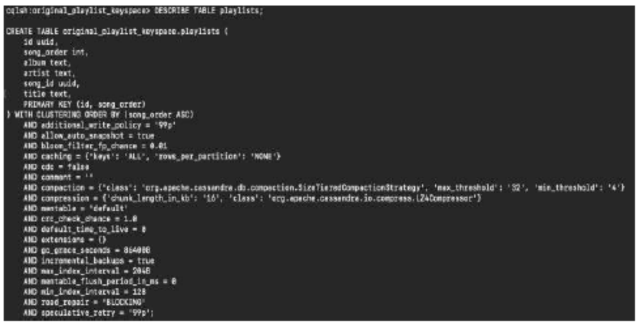
In Figure **4**, 10 rows are inserted into the playlists table with unique IDs and details about each song.



```
cqlsh:original_playlist_keyspace> SELECT * FROM playlists;

 id                                   | song_order | album            | artist         | song_id                              | title
--------------------------------------+------------+------------------+----------------+--------------------------------------+------------------
 8e870286-be78-48ae-814c-94181cb684ec |          2 |      After Hours |     The Weeknd | f8e83162-48be-4611-b6e4-a7946b9522d8 |  Blinding Lights
 e1a7ed69-396e-43e8-8a1f-2cbe81f17949 |         10 |      After Hours |     The Weeknd | 3bfffe4e-37e3-4792-9789-e23cd7e21a0d |  Save Your Tears
 db008708-423f-48ce-80f6-b4d6640d9619 |          6 |          Montero |      Lil Nas X | 4117283d-fdff-4413-9aae-6f9949aa807f |          Montero
 eb6e67be-ea38-4888-8abf-f0fe157e58d9 |          7 |          Montero |      Lil Nas X | d920f70e-fc0c-4ad5-b97a-1e9747dccd6d |    Industry Baby
 d83f8c2e-e813-4aea-aeac-891424389d83 |          9 |        Fine Line |   Harry Styles | 56204e28-9826-4d9a-aaaf-e8f826669a7c | Watermelon Sugar
 d95447fb-b876-42a4-9e12-8f467ab6db17 |          3 | Future Nostalgia |       Dua Lipa | 2a2d381f-9f3d-456e-8de2-fa4ead6d6efe |       Levitating
 0a0041c3-3980-4117-a06e-838deb298288 |          8 |         YHLQMDLG |      Bad Bunny | 44c89cde-8474-4bc8-af12-812d64d2aee5 |        Bad Bunny
 d385aeae-b717-449c-a220-26695abefad3 |          1 |           Divide |     Ed Sheeran | 7bdd3cc2-150e-48fd-bdb5-43f491442443 |     Shape of You
 7f823f17-fe61-41a6-907b-1e732d685a37 |          5 |             SOUR | Olivia Rodrigo | 27178f2a-692b-4de9-a8d4-fcce642bba84 |         Good 4 U
 8604e47d-d6d3-4777-9ed1-24c874bc256d |          4 |          Justice |  Justin Bieber | e1a7138e-4ca6-43bd-9e93-e46e844ad78d |          Peaches

(10 rows)
```

Figure 5: Table Content of the Original Keyspace in Cassandra.

Figure **5** shows the verification of the data by querying it. Thus, it retrieves all data from the playlists table.

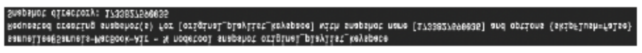


```
cqlsh:original_playlist_keyspace> DESCRIBE TABLE playlists;

CREATE TABLE original_playlist_keyspace.playlists (
    id uuid,
    song_order int,
    album text,
    artist text,
    song_id uuid,
    title text,
    PRIMARY KEY (id, song_order)
) WITH CLUSTERING ORDER BY (song_order ASC)
    AND additional_write_policy = '99p'
    AND allow_auto_snapshot = true
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND cdc = false
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND memtable = 'default'
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND extensions = {}
    AND gc_grace_seconds = 864000
    AND incremental_backups = true
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';
```

Figure 6: Table Structure of the Original Keyspace in Cassandra.

Based on Figure **6**, it outputs schema details like primary keys, compression, and compaction settings. Hence, this is useful for validating table design and configuration.

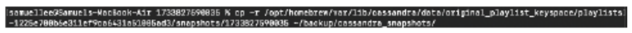
2) Data Backup



```
samuellee@Samuels-MacBook-Air ~ % nodetool snapshot original_playlist_keyspace
Requested creating snapshot(s) for [original_playlist_keyspace] with snapshot name [1733827590035] and options {skipFlush=false}
Snapshot directory: 1733827590035
```

Figure 7: Nodetool Snapshot Command in Cassandra.

In Figure **7**, the nodetool snapshot command is used to create a point-in-time snapshot of the keyspace. It captures the current state of all the SSTable files in the keyspace without interrupting the database's operations. The skip Flush=false option ensures that any data in the memtable (in-memory storage) is flushed to the SSTables on disk, guaranteeing the backup includes the most recent writes. As a result, a directory named after the snapshot (1733827590035) is created within the data directory of each table in the keyspace.

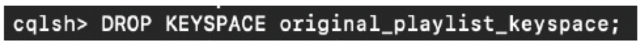


```
samuellee@Samuels-MacBook-Air 1733827590035 % cp -r /opt/homebrew/var/lib/cassandra/data/original_playlist_keyspace/playlists
-1226e700b6e311ef9ca6431a61006ad3/snapshots/1733827590035 ~/backup/cassandra_snapshots/
```

Figure 8: Copying Snapshot Files to a Backup Location in Cassandra.

Figure **8** shows how to copy the snapshot files from Cassandra's data directory to a new backup location (~/backup/cassandra_snapshots/) to protect data from accidental deletions or hardware failures. If the snapshot files be backed up to an external place, then important data can be restored even if the original Cassandra data directory is hacked. This phase is crucial for catastrophe recovery planning.

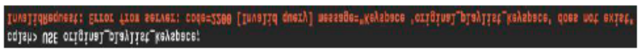
3) Data Loss Simultation



```
cqlsh> DROP KEYSPACE original_playlist_keyspace;
```

Figure 9: Drop Command to Simulate Data Loss in Cassandra.

Figure **9** shows deletion of the original_playlist_keyspace and everything in it from the database to mimic a loss or intentional cleanup. The keyspace is no longer in Cassandra.

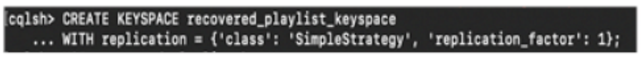


```
cqlsh> USE original_playlist_keyspace;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Keyspace 'original_playlist_keyspace' does not exist"
```

Figure 10: Verification of the Dropped Keyspace in Cassandra.

Figure **10** shows the command USE original_playlist_keyspace, which is used to switch the current context in the cqlsh shell to the keyspace named original_playlist_keyspace. As a result, the error message indicates that the original_playlist_keyspace keyspace no longer exists in the Cassandra database. This is due to the keyspace being deleted, trying to USE it resulted in an error because Cassandra could no longer find it.

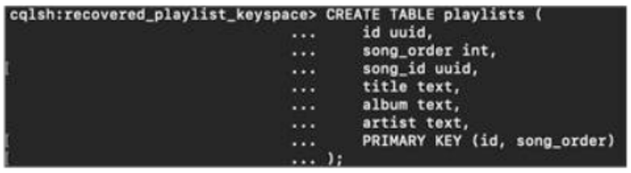
4) Data Restoration



```
cqlsh> CREATE KEYSPACE recovered_playlist_keyspace
   ... WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

Figure 11: New Keyspace Creation for Recovery in Cassandra.

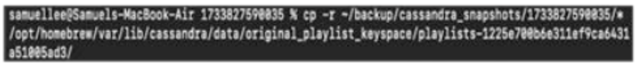
In Figure **11**, it sets up a new keyspace named recovered_playlist_keyspace for restoring data.



```
cqlsh:recovered_playlist_keyspace> CREATE TABLE playlists (
                              ...     id uuid,
                              ...     song_order int,
                              ...     song_id uuid,
                              ...     title text,
                              ...     album text,
                              ...     artist text,
                              ...     PRIMARY KEY (id, song_order)
                              ... );
```

Figure 12: Recreation of the Table in the New Keyspace in Cassandra.

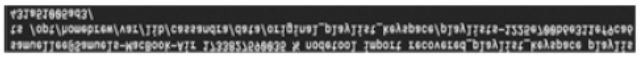
Figure **12** defines the same table schema in the new keyspace.



```
samuellee@Samuels-MacBook-Air 1733827590035 % cp -r ~/backup/cassandra_snapshots/1733827590035/*
/opt/homebrew/var/lib/cassandra/data/original_playlist_keyspace/playlists-1225e700b6e311ef9ca6431
a51005ad3/
```

Figure 13: Snapshot Files are Copied to New Keyspace in Cassandra.

Figure **13** shows the preparation for data restoration by placing snapshot files into the appropriate directory in Cassandra's data structure. The source directory (~/backup/cassandra_snapshots/1733827590035/) contains the snapshot files created earlier, such as, manifest.json, schema.cql and data files (nb-1-big-Data.db, etc.). However, the destination directory (/opt/homebrew/var/lib/cassandra/...) is where Cassandra expects to find the data files for the table in the restored keyspace. The folder name (playlists-1225e700b6e311ef9ca6431a51005ad3) is unique to the table. As a result, the snapshot files are now located in the exact folder where Cassandra manages the table's data, enabling the restoration process.

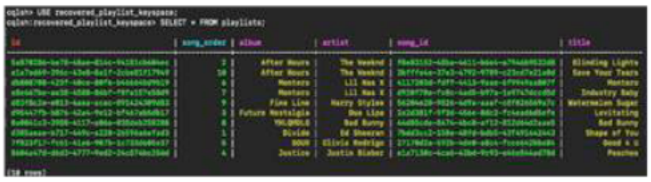


```
samuellee@Samuels-MacBook-Air 1733827590035 % nodetool import recovered_playlist_keyspace playlis
ts /opt/homebrew/var/lib/cassandra/data/original_playlist_keyspace/playlists-1225e700b6e311ef9ca
6431a51005ad3/
```

Figure 14: The Use of Nodetool Import to Restore the Snapshot Data in Cassandra.

In Figure **14**, the table's data is restored into Cassandra by importing snapshot files into a new keyspace and table. The process utilizes nodetool, a powerful command-line utility for managing and monitoring Cassandra, along with the import command, which reads the snapshot files and integrates them into the table. The restoration is performed in a new keyspace, recovered_playlist_keyspace, and specifically targets the playlists table within that keyspace. The source path, /opt/homebrew/var/lib/cassandra/..., points to the directory where the snapshot files were copied earlier, containing both the data and metadata required for table restoration. As a result, Cassandra processes the snapshot files, recreates the table's data within the specified keyspace, and makes it fully accessible for queries.

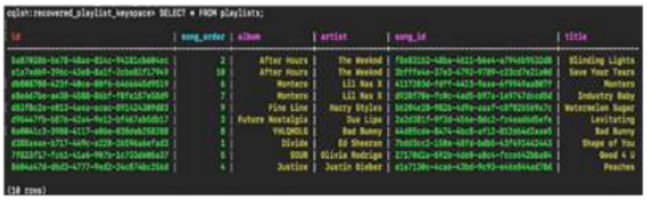
5) Verification of Data Restoration



```
cqlsh> USE recovered_playlist_keyspace;
cqlsh:recovered_playlist_keyspace> SELECT * FROM playlists;

 id             | song_order | album            | artist         | song_id        | title
----------------+------------+------------------+----------------+----------------+------------------
 [illegible]    |          2 |      After Hours |     The Weeknd | [illegible]    |  Blinding Lights
 [illegible]    |         10 |      After Hours |     The Weeknd | [illegible]    |  Save Your Tears
 [illegible]    |          6 |          Montero |      Lil Nas X | [illegible]    |          Montero
 [illegible]    |          7 |          Montero |      Lil Nas X | [illegible]    |    Industry Baby
 [illegible]    |          9 |        Fine Line |   Harry Styles | [illegible]    | Watermelon Sugar
 [illegible]    |          3 | Future Nostalgia |       Dua Lipa | [illegible]    |       Levitating
 [illegible]    |          8 |         YHLQMDLG |      Bad Bunny | [illegible]    |        Bad Bunny
 [illegible]    |          1 |           Divide |     Ed Sheeran | [illegible]    |     Shape of You
 [illegible]    |          5 |             SOUR | Olivia Rodrigo | [illegible]    |         Good 4 U
 [illegible]    |          4 |          Justice |  Justin Bieber | [illegible]    |          Peaches

(10 rows)
```

Figure 15: Table Content of the Recovered Keyspace in Cassandra.

Figure **15** shows the successful restoration of the playlists table from the recovered_playlist_keyspace, complete with all rows and columns, including id, song_order, album, artist, song_id, and title. This implies that both the integrity of the data and the schema were preserved throughout the restoration process.

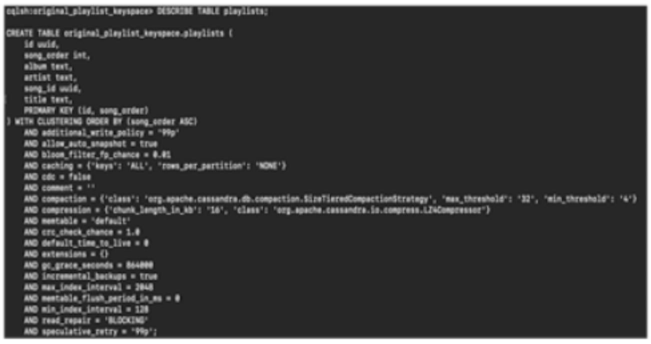


```
cqlsh:recovered_playlist_keyspace> SELECT * FROM playlists;

 id             | song_order | album            | artist         | song_id        | title
----------------+------------+------------------+----------------+----------------+------------------
 [illegible]    |          2 |      After Hours |     The Weeknd | [illegible]    |  Blinding Lights
 [illegible]    |         10 |      After Hours |     The Weeknd | [illegible]    |  Save Your Tears
 [illegible]    |          6 |          Montero |      Lil Nas X | [illegible]    |          Montero
 [illegible]    |          7 |          Montero |      Lil Nas X | [illegible]    |    Industry Baby
 [illegible]    |          9 |        Fine Line |   Harry Styles | [illegible]    | Watermelon Sugar
 [illegible]    |          3 | Future Nostalgia |       Dua Lipa | [illegible]    |       Levitating
 [illegible]    |          8 |         YHLQMDLG |      Bad Bunny | [illegible]    |        Bad Bunny
 [illegible]    |          1 |           Divide |     Ed Sheeran | [illegible]    |     Shape of You
 [illegible]    |          5 |             SOUR | Olivia Rodrigo | [illegible]    |         Good 4 U
 [illegible]    |          4 |          Justice |  Justin Bieber | [illegible]    |          Peaches

(10 rows)
```

Figure 16: Table Structure of the Recovered Keyspace in Cassandra.

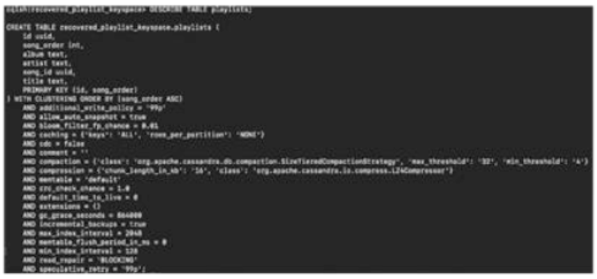
Figure **16** provides the table structure of the playlists table, confirming it matches the original schema, including properties like the PRIMARY KEY, clustering order, compression settings, and other table-level configurations. The verification ensures that the restored data is complete, accurate, and consistent with the pre-loss state, affirming the reliability of the recovery process. The schema and row count of the original and restored keyspaces are compared. This can verify that no data was lost or corrupted during the restoration process.

a) Comparison of the Table Structure of Original Keyspace and Recovered Keyspace



```
cqlsh:original_playlist_keyspace> DESCRIBE TABLE playlists;

CREATE TABLE original_playlist_keyspace.playlists (
    id uuid,
    song_order int,
    album text,
    artist text,
    song_id uuid,
    title text,
    PRIMARY KEY (id, song_order)
) WITH CLUSTERING ORDER BY (song_order ASC)
    AND additional_write_policy = '99p'
    AND allow_auto_snapshot = true
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND cdc = false
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND memtable = 'default'
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND extensions = {}
    AND gc_grace_seconds = 864000
    AND incremental_backups = true
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';
```

Figure 17: The Table Structure of the Original Keyspace.

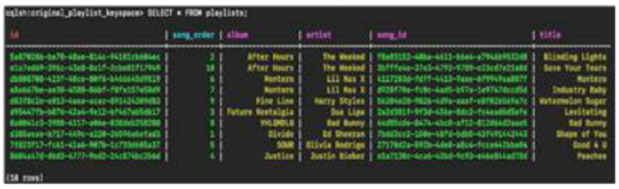


```
cqlsh:recovered_playlist_keyspace> DESCRIBE TABLE playlists;

CREATE TABLE recovered_playlist_keyspace.playlists (
    id uuid,
    song_order int,
    album text,
    artist text,
    song_id uuid,
    title text,
    PRIMARY KEY (id, song_order)
) WITH CLUSTERING ORDER BY (song_order ASC)
    AND additional_write_policy = '99p'
    AND allow_auto_snapshot = true
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND cdc = false
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND memtable = 'default'
    AND crc_check_chance = 1.0
    AND default_time_to_live = 0
    AND extensions = {}
    AND gc_grace_seconds = 864000
    AND incremental_backups = true
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair = 'BLOCKING'
    AND speculative_retry = '99p';
```

Figure 18: The Table Structure of the Recovered Keyspace.

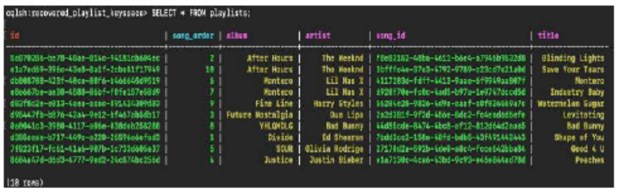
Both table definitions as shown in Figure **17** and **18** are exactly the same in schema, clustering, and configuration. This ensures that both original_playlist_keyspace and recovered_playlist_keyspace have identical structure and behavior. This confirms that the table properties remain consistent between the two keyspaces.

b) Comparison of the Number of Rows of Original Keyspace and Recovered Keyspace



```
cqlsh:original_playlist_keyspace> SELECT * FROM playlists;

 id                                   | song_order | album            | artist         | song_id                              | title
--------------------------------------+------------+------------------+----------------+--------------------------------------+----------------
 5a870286-be78-48ae-814c-94181cb604ec |          2 |      After Hours |     The Weeknd | f8e93162-48be-4611-b6e4-a7946b9532d8 |  Blinding Lights
 e1a7ed69-396c-43e8-8a1f-2cbe81f17949 |         10 |      After Hours |     The Weeknd | 3bfffe4e-37e3-4792-9789-c23cd7e21a8d |  Save Your Tears
 db008708-423f-48ce-88f6-b46dd44d9519 |          6 |          Montero |      Lil Nas X | 4117283d-fdff-4413-9aee-6f9949aa807f |          Montero
 e0e667be-ae38-4588-86bf-f0fe157e68d9 |          7 |          Montero |      Lil Nas X | d920f70e-fc8c-4ad5-b97a-1e9747dccd5d |    Industry Baby
 d63f8d2e-e813-4aea-acae-891424309d83 |          9 |        Fine Line |   Harry Styles | 6620e428-9826-4d9a-aa5f-c0f926469a7c | Watermelon Sugar
 d95447fb-b876-42a4-9e12-bf467ab6db17 |          3 | Future Nostalgia |       Dua Lipa | 2a2d281f-9f2d-46ee-8dc2-fc4eaded6efe |       Levitating
 8e0061c3-3988-4117-a06e-838deb258288 |          8 |          YHLQMDLG |       Bad Bunny | 44d8fcde-8474-4bc8-af12-8126d4d2aa65 |        Bad Bunny
 d386eeae-b717-449c-a220-2659a6e6fad3 |          1 |           Divide |     Ed Sheeran | 7bdd1cc2-156e-40fd-bdb6-43f491442443 |     Shape of You
 7f023f17-fc61-41a6-987b-1c731d606a37 |          5 |             SOUR | Olivia Rodrigo | 27178d2a-892b-4de8-a8c4-fcce642bba84 |         Good 4 U
 8684a47d-d6d3-4777-9ed2-26c874bc256d |          4 |          Justice |  Justin Bieber | e1a7130c-4ca6-43bd-9e93-e46e844ad78d |          Peaches

(10 rows)
```

Figure 19: The Number of Rows of Original Keyspace (10 Rows).

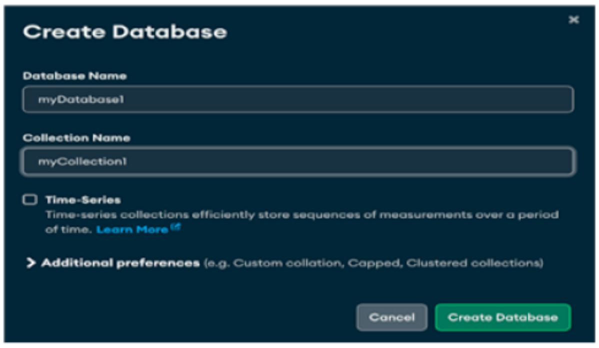


```
cqlsh:recovered_playlist_keyspace> SELECT * FROM playlists;

 id                                   | song_order | album            | artist         | song_id                              | title
--------------------------------------+------------+------------------+----------------+--------------------------------------+----------------
 8a870286-be78-48ae-814c-94181cb604ec |          2 |      After Hours |     The Weeknd | f8e93162-48be-4611-b6e4-a7946b9532d8 |  Blinding Lights
 e1a7ed69-396c-43e8-8a1f-2cbe81f17949 |         10 |      After Hours |     The Weeknd | 3bfffe4e-37e3-4792-9789-c23cd7e21a8d |  Save Your Tears
 db008708-423f-48ce-88f6-b46dd44d9519 |          6 |          Montero |      Lil Nas X | 4117283d-fdff-4413-9aee-6f9949aa807f |          Montero
 e0e667be-ae38-4588-86bf-f0fe157e68d9 |          7 |          Montero |      Lil Nas X | d920f70e-fc8c-4ad5-b97a-1e9747dccd5d |    Industry Baby
 d63f8d2e-e813-4aea-acae-891424309d83 |          9 |        Fine Line |   Harry Styles | 6620e428-9826-4d9a-aa5f-c0f926469a7c | Watermelon Sugar
 d95447fb-b876-42a4-9e12-bf467ab6db17 |          3 | Future Nostalgia |       Dua Lipa | 2a2d281f-9f2d-46ee-8dc2-fc4eaded6efe |       Levitating
 8e0061c3-3988-4117-a06e-838deb258288 |          8 |          YHLQMDLG |       Bad Bunny | 44d8fcde-8474-4bc8-af12-8126d4d2aa65 |        Bad Bunny
 d386eeae-b717-449c-a220-2659a6e6fad3 |          1 |           Divide |     Ed Sheeran | 7bdd1cc2-156e-40fd-bdb6-43f491442443 |     Shape of You
 7f023f17-fc61-41a6-987b-1c731d606a37 |          5 |             SOUR | Olivia Rodrigo | 27178d2a-892b-4de8-a8c4-fcce642bba84 |         Good 4 U
 8684a47d-d6d3-4777-9ed2-26c874bc256d |          4 |          Justice |  Justin Bieber | e1a7130c-4ca6-43bd-9e93-e46e844ad78d |          Peaches

(10 rows)
```

Figure 20: The Number of Rows of Recovered Keyspace (10 Rows).

The number of rows are identical as shown in Figure **19** and **20**. Hence, the recovery process was successful and no data was lost or duplicated during the restoration.

B. MongoDB



Create Database

Database Name: myDatabase1

Collection Name: myCollection1

[ ] Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More

> Additional preferences (e.g. Custom collation, Capped, Clustered collections)
Cancel
Create Database

Figure 21: Creation of a new database using MongoDB Compass.

1) Data Creation

Based on Figure **21**, myDatabase1 was created in MongoDB to serve as the target repository for importing data

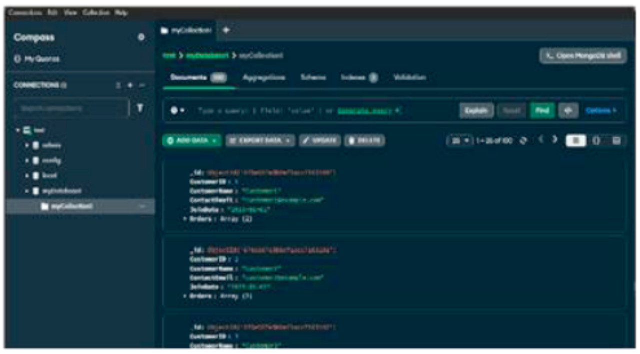
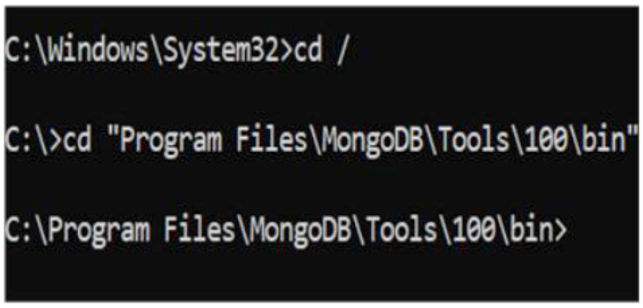


Figure 22: Imported data into the database using MongoDB Compass.

Figure **22** shows that a dataset was imported into the collection myCollection1 in the database myDatabase1 in JSON file format. The file includes 100 customer records. JSON files are suitable for MongoDB because they are compatible with the BSON (Binary JSON) data storage format of MongoDB.

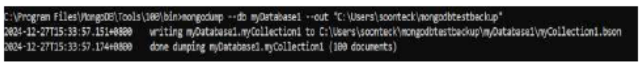
2) Data Backup



```
C:\Windows\System32>cd /

C:\>cd "Program Files\MongoDB\Tools\100\bin"

C:\Program Files\MongoDB\Tools\100\bin>
```

Figure 23: Navigation to the bin folder in MongoDB.

In Figure **23**, the command prompt was used to navigate to the bin folder of MongoDB. This folder contains executable programs, which are MongoDB's built-in tools. The tools that are used here are mongodump and mongorestore, which are for backup and restoring data.



```
C:\Program Files\MongoDB\Tools\100\bin>mongodump --db myDatabase1 --out "C:\Users\soonteck\mongodbtestbackup"
2024-12-27T15:33:57.151+0800    writing myDatabase1.myCollection1 to C:\Users\soonteck\mongodbtestbackup\myDatabase1\myCollection1.bson
2024-12-27T15:33:57.174+0800    done dumping myDatabase1.myCollection1 (100 documents)
```

Figure 24: Database backup in MongoDB.

Based on Figure **24**, mongodump command is used to create a backup for the database. This will generate a directory containing BSON files for data and metadata. The --db specifies the database to back up and the --out parameter means the destination directory to store the backup files. This command creates a backup of the database for disaster recovery.

3) Data Loss Simulation

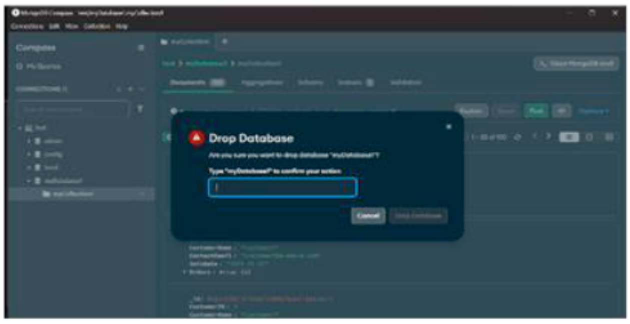


Figure 25: Database deletion in MongoDB Compass.

Figure **25** shows that the database was dropped or deleted in MongoDB to simulate a data loss scenario. All data that are included in the database is now erased.

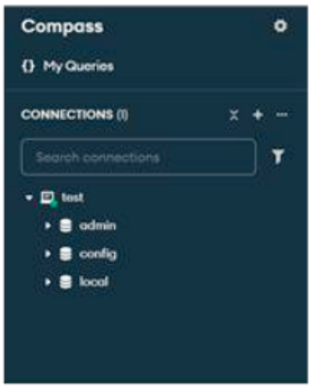


Figure 26: Verification of database deletion in MongoDB Compass.

Figure **26** illustrates that dropped database is no longer listed in the Compass interface. The remaining database are system databases such as admin, config, and local. This indicates that the database we created has been confirmed to be deleted.

4) Data Restoration

```
[illegible]
```

Figure 27: Database restoration in MongoDB.

Figure **27** shows the use of mongorestore command to restore the database that was backed up. The name of the database to be restored is specified by the --db option. The path to restore the database is the path where the backup file was located. From the command prompt, it shows that 100 documents were restored and 0 documents failed, which indicates that the restoration operation was successful. Therefore, the recovery rate was 100%.

5) Verification of Data Restoration

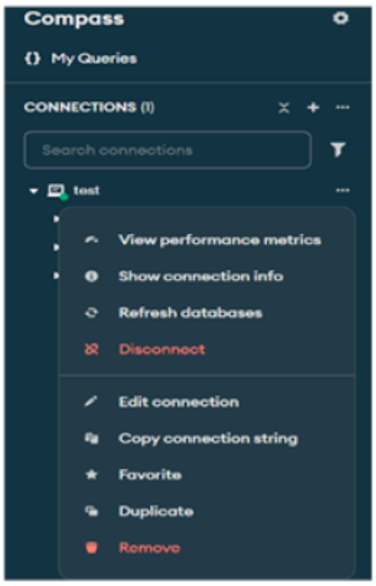


Figure 28: Database refresh in MongoDB.

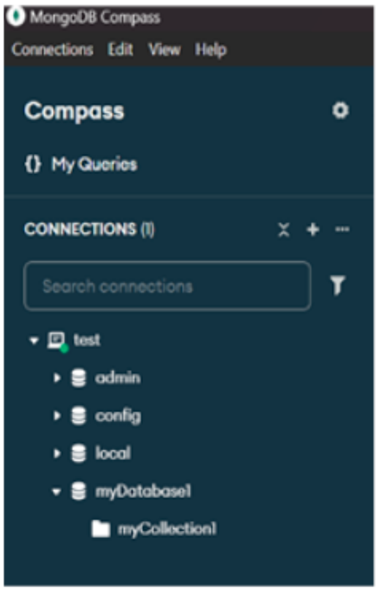


Figure 29: Restoration of myDatabase1in MongoDB Compass.

To see where the restored database and its collections is, we need to refresh the database (Figure **28**). The restored database myDatabase1 is mentioned under the connection test once the connection has been established, as shown in Figure **29**. The documents are seen when you navigate to myCollection1, confirming that the database has been successfully recovered and is queryable. This phase makes sure there are no connection or configuration problems and that the MongoDB Compass interface accurately displays the restored state.

a) Comparison of the Original Database and Recovered Database

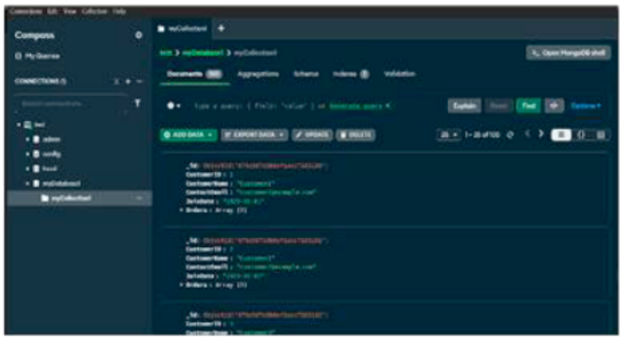


Figure 30: Original Database in MongoDB Compass.

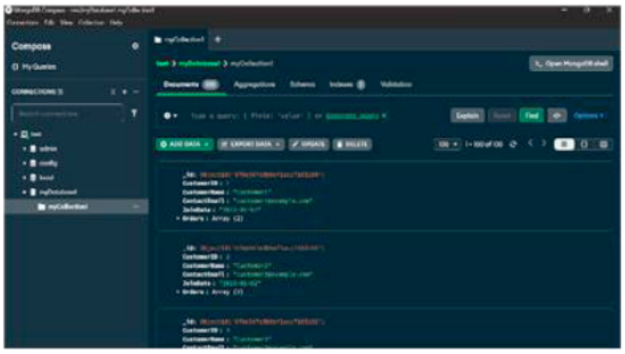
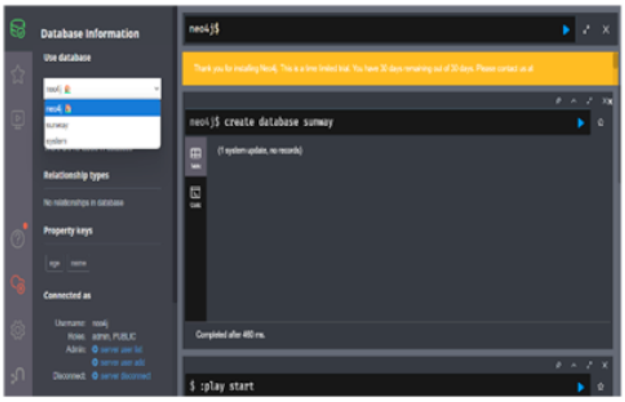


Figure 31: Recovered Database in MongoDB Compass.

The content of the restored database is compared to check for accuracy of restoration and the completeness of the restored data. By comparing Figure **30** and Figure **31**, it is shown that the restored content remains the same as the original data, and each document has fields such as Customer ID, Customer Name, Contact Email, Join Date, and Orders. This indicates that the values and structure of the restored database is consistent. This also verifies the integrity of the recovered database and guarantees that no documents were lost or corrupted during the restoration process. This experimental backup helps to verify the backup mechanism's dependability and the restoration operation's success.

C. Neo4j

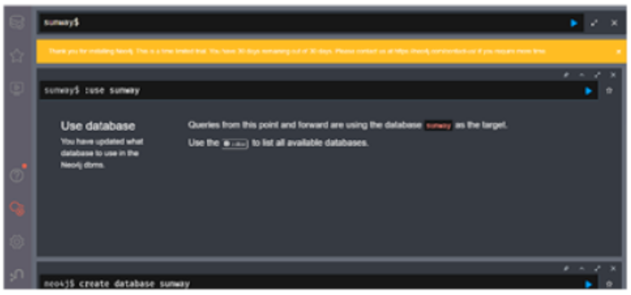
1) Data Cration



```
neo4j$ create database sunway
(1 system update, no records)
Completed after 480 ms.
$ :play start
```

Figure 32: New database creation in Neo4j.

In Figure **32**, a new database named ‘sunway’ is created by using the Cypher query in Neo4j.



```
sunway$ :use sunway
```

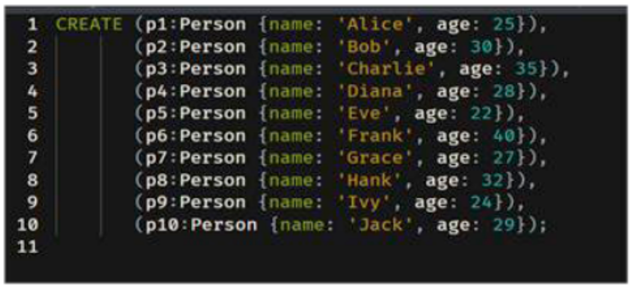
Use database
You have updated what database to use in the Neo4j dbms.

Queries from this point and forward are using the database sunway as the target.
Use the :dbs to list all available databases.

```
neo4j$ create database sunway
```

Figure 33: Switching to the new database in Neo4j.

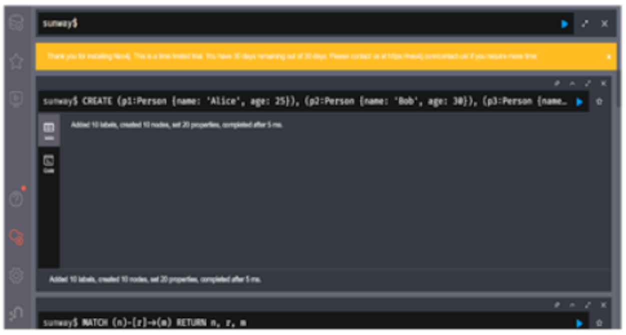
Based on Figure **33**, once the database is created, select it by clicking on its name in the Neo4j interface. When the database name ‘sunway’ appears next to the $ sign in the query window, this means that the user is now working within the selected database.



```
CREATE (p1:Person {name: 'Alice', age: 25}),
       (p2:Person {name: 'Bob', age: 30}),
       (p3:Person {name: 'Charlie', age: 35}),
       (p4:Person {name: 'Diana', age: 28}),
       (p5:Person {name: 'Eve', age: 22}),
       (p6:Person {name: 'Frank', age: 40}),
       (p7:Person {name: 'Grace', age: 27}),
       (p8:Person {name: 'Hank', age: 32}),
       (p9:Person {name: 'Ivy', age: 24}),
       (p10:Person {name: 'Jack', age: 29});
```

Figure 34: Nodes creation in Neo4j.

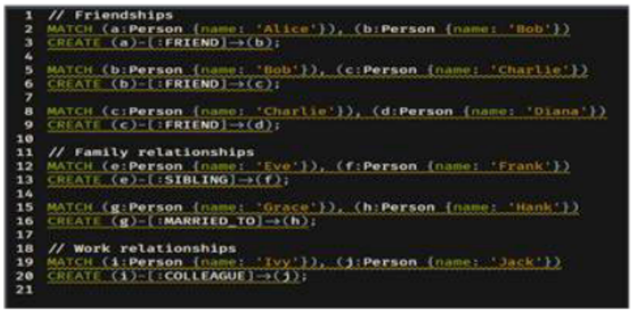
Figure **34** shows how to build nodes in a Neo4j database using the Cypher query. Each node is a person which contains both age and name. The CREATE statement is used to create the nodes. The nodes can represent people, ages and more by putting the characteristics inside the curly brackets. A placeholder variable is added in front of each node—such as p1, p2, etc. It is for future use in creating associations.



```
sunway$ CREATE (p1:Person {name: 'Alice', age: 25}), (p2:Person {name: 'Bob', age: 30}), (p3:Person {name…
Added 10 labels, created 10 nodes, set 20 properties, completed after 5 ms.
Added 10 labels, created 10 nodes, set 20 properties, completed after 5 ms.
sunway$ MATCH (n)-[r]->(m) RETURN n, r, m
```

Figure 35: Successful node creation in Neo4j.

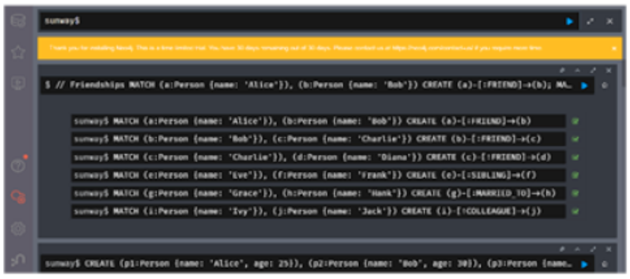
Figure **35** shows that the CREATE query have been successfully executed in the Neo4j interface. 10 nodes were successfully added to the graph database, each with two properties (name and age). The message at the bottom of the interface indicates that the nodes were created correctly, and the graph database now contains individual entities ready to be interacted in the future.



```
// Friendships
MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'})
CREATE (a)-[:FRIEND]->(b);

MATCH (b:Person {name: 'Bob'}), (c:Person {name: 'Charlie'})
CREATE (b)-[:FRIEND]->(c);

MATCH (c:Person {name: 'Charlie'}), (d:Person {name: 'Diana'})
CREATE (c)-[:FRIEND]->(d);

// Family relationships
MATCH (e:Person {name: 'Eve'}), (f:Person {name: 'Frank'})
CREATE (e)-[:SIBLING]->(f);

MATCH (g:Person {name: 'Grace'}), (h:Person {name: 'Hank'})
CREATE (g)-[:MARRIED_TO]->(h);

// Work relationships
MATCH (i:Person {name: 'Ivy'}), (j:Person {name: 'Jack'})
CREATE (i)-[:COLLEAGUE]->(j);
```

Figure 36: Defining relationships in Neo4j.

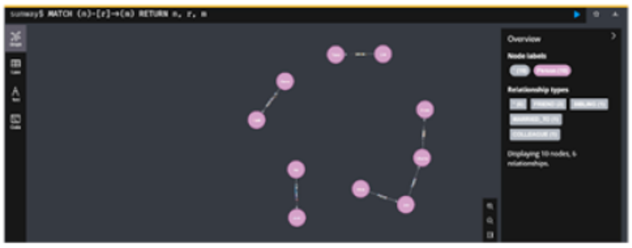
Figure **36** shows the Cypher queries used to create relationships between the previously created nodes. In this example, the relationships are separated into three categories, which are friendships, family ties, and work associations. Existing nodes are identified based on their properties such as name using the MATCH statement, and the CREATE statement is used to define relationships such as FRIEND, SIBLING, MARRIED_TO, and COLLEAGUE. Based on the figure, Alice is connected to Bob as a friend, Eve is linked to Frank as a sibling and many more. This shows how relationships connect different nodes together in a graph database.



```
sunway$
$ // Friendships MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'}) CREATE (a)-[:FRIEND]->(b); MA…

sunway$ MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'}) CREATE (a)-[:FRIEND]->(b)
sunway$ MATCH (b:Person {name: 'Bob'}), (c:Person {name: 'Charlie'}) CREATE (b)-[:FRIEND]->(c)
sunway$ MATCH (c:Person {name: 'Charlie'}), (d:Person {name: 'Diana'}) CREATE (c)-[:FRIEND]->(d)
sunway$ MATCH (e:Person {name: 'Eve'}), (f:Person {name: 'Frank'}) CREATE (e)-[:SIBLING]->(f)
sunway$ MATCH (g:Person {name: 'Grace'}), (h:Person {name: 'Hank'}) CREATE (g)-[:MARRIED_TO]->(h)
sunway$ MATCH (i:Person {name: 'Ivy'}), (j:Person {name: 'Jack'}) CREATE (i)-[:COLLEAGUE]->(j)

sunway$ CREATE (p1:Person {name: 'Alice', age: 25}), (p2:Person {name: 'Bob', age: 30}), (p3:Person {name…
```

Figure 37: Successful relationship creation in Neo4j.

Based on Figure **37**, the relationship queries in the are successfully executed. This was indicated by the green checkmark beside each statement on the UI. This ensures that the graph structure now has edges joining the nodes, which can be used to demonstrate the real-world connections such as social ties.

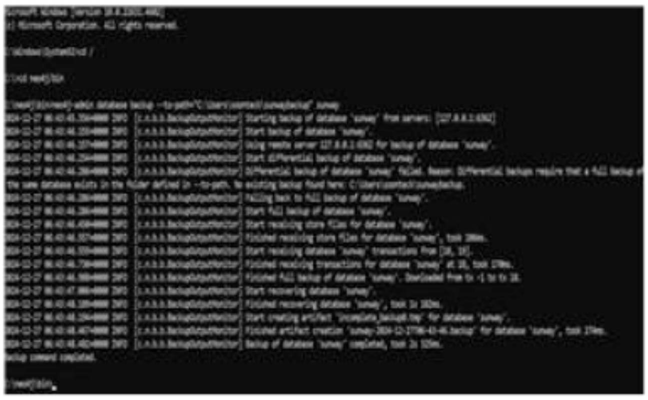


```
sunway$ MATCH (n)-[r]->(m) RETURN n, r, m
```

Figure 38: Graph visualisation in Neo4j.

The graph which contains nodes and their relationshipare shown in the Neo4j browser interface in Figure **38**. It includes ten nodes, each with a designated 'Person' and is joined by a total of six relationships. Four relationship types—FRIEND, SIBLING, MARRIED_TO, and COLLEAGUE—are confirmed to exist by the metadata in the interface's right-hand panel. With relationships like Grace being married to Hank and Alice being Bob's friend shown in the graph, the visualization offers a clear and understandable depiction of the data. This attests to the effective completion of the node and relationship formation processes, producing a graph with a clear structure.

2) Data Backup



```
Microsoft Windows [Version [illegible]]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd /

C:\>cd neo4j\bin

C:\neo4j\bin>neo4j-admin database backup --to-path="C:\Users\[illegible]\sunwaybackup" sunway
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Starting backup of database 'sunway' from servers: [127.0.0.1:6362]
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Start backup of database 'sunway'.
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Using remote server 127.0.0.1:6362 for backup of database 'sunway'.
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Start differential backup of database 'sunway'.
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Differential backup of database 'sunway' failed. Reason: Differential backups require that a full backup of
the same database exists in the folder defined in --to-path. No existing backup found here: C:\Users\[illegible]\sunwaybackup.
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Falling back to full backup of database 'sunway'.
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Start full backup of database 'sunway'.
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Start receiving store files for database 'sunway'.
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Finished receiving store files for database 'sunway', took [illegible].
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Start receiving database 'sunway' transactions from [illegible].
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Finished receiving transactions for database 'sunway' at [illegible], took [illegible].
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Finished full backup of database 'sunway'. Downloaded from tx -1 to tx [illegible].
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Start recovering database 'sunway'.
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Finished recovering database 'sunway', took [illegible].
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Start creating artifact 'incomplete_backup[illegible].tmp' for database 'sunway'.
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Finished artifact creation 'sunway-2024-12-27T[illegible].backup' for database 'sunway', took [illegible].
2024-12-27 [illegible] INFO  [c.n.b.b.BackupOutputMonitor] Backup of database 'sunway' completed, took [illegible].
Backup command completed.

C:\neo4j\bin>
```

Figure 39: Database backup in Neo4j.

In Figure **39**, a new Command Prompt was opened and navigated to the bin folder in the Neo4j installation directory. To use the backup and restore feature in Neo4j, the neo4j-admin command is used. To backup a database, we specified the target backup folder and database name. The command format is:

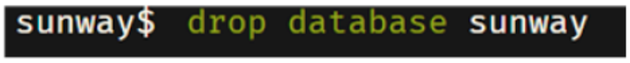
neo4j-admin database backup --to-path=<path_name> <database_name>

For example, we run this command to backup the 'sunway' database:

neo4j-admin database backup --to-path=C:¥backup sunway

This command creates a full backup of the database in the specified folder, ensuring all data, relationships, and indexes are saved securely for future restoration.

3) Data Loss Simulation



```
sunway$ drop database sunway
```

Figure 40: Database deletion in Neo4j.

Based on Figure **40**, the *DROP DATABASE* command is used to drop or delete a database in the Neo4j interface. The syntax for this command is:

DROP DATABASE <NAME>;

Here*, <NAME>* should be replaced with the name of the database to be deleted. In this example, we will be deleting the 'sunway' database, therefore the command will be:

DROP DATABASE sunway;

This command will remove the 'sunway' database from Neo4j. This step ensures the environment is ready for testing the restoration process.

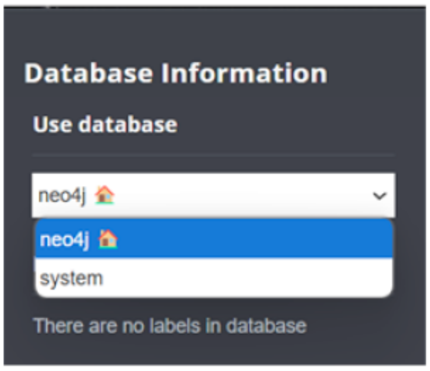
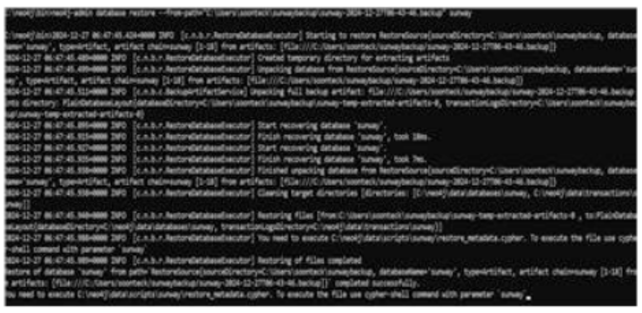


Figure 41: Verification of database deletion in Neo4j.

Based on Figure **41**, after executing the *DROP DATABASE sunway* command, the 'sunway' database is no longer available. The dropdown menu only shows the default neo4j and system databases, indicating that 'sunway' database has been successfully deleted from Neo4j. This verification step ensures that the environment is now ready for testing the restoration process.

4) Data Restoration



```
C:\neo4j\bin>neo4j-admin database restore --from-path="C:\Users\soontech\sunwaybackup\sunway-2024-12-27T06-43-46.backup" sunway

C:\neo4j\bin>2024-12-27 06:47:45.424+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Starting to restore RestoreSource{sourceDirectory=C:\Users\soontech\sunwaybackup, database
Name='sunway', type=Artifact, artifact chain=sunway [1-18] from artifacts: [file:///C:/Users/soontech/sunwaybackup/sunway-2024-12-27T06-43-46.backup]}
2024-12-27 06:47:45.489+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Created temporary directory for extracting artifacts
2024-12-27 06:47:45.499+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Unpacking database from RestoreSource{sourceDirectory=C:\Users\soontech\sunwaybackup, databaseName='sun
way', type=Artifact, artifact chain=sunway [1-18] from artifacts: [file:///C:/Users/soontech/sunwaybackup/sunway-2024-12-27T06-43-46.backup]}
2024-12-27 06:47:45.511+0000 INFO  [c.n.b.c.BackupArtifactService] Unpacking full backup artifact: file:///C:/Users/soontech/sunwaybackup/sunway-2024-12-27T06-43-46.backup
into directory: PlainDatabaseLayout{databaseDirectory=C:\Users\soontech\sunwaybackup\sunway-temp-extracted-artifacts-0, transactionLogsDirectory=C:\Users\soontech\sunwayba
ckup\sunway-temp-extracted-artifacts-0}
2024-12-27 06:47:45.895+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Start recovering database 'sunway'.
2024-12-27 06:47:45.915+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Finish recovering database 'sunway', took 18ms.
2024-12-27 06:47:45.927+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Start recovering database 'sunway'.
2024-12-27 06:47:45.935+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Finish recovering database 'sunway', took 7ms.
2024-12-27 06:47:45.938+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Finished unpacking database from RestoreSource{sourceDirectory=C:\Users\soontech\sunwaybackup, database
Name='sunway', type=Artifact, artifact chain=sunway [1-18] from artifacts: [file:///C:/Users/soontech/sunwaybackup/sunway-2024-12-27T06-43-46.backup]}
2024-12-27 06:47:45.938+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Cleaning target directories [directories: [C:\neo4j\data\databases\sunway, C:\neo4j\data\transactions\s
unway]]
2024-12-27 06:47:45.940+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Restoring files [from:C:\Users\soontech\sunwaybackup\sunway-temp-extracted-artifacts-0 , to:PlainDataba
seLayout{databaseDirectory=C:\neo4j\data\databases\sunway, transactionLogsDirectory=C:\neo4j\data\transactions\sunway}]
2024-12-27 06:47:45.988+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] You need to execute C:\neo4j\data\scripts\sunway\restore_metadata.cypher. To execute the file use cyphe
r-shell command with parameter `sunway`
2024-12-27 06:47:45.989+0000 INFO  [c.n.b.r.RestoreDatabaseExecutor] Restoring of files completed
Restore of database `sunway` from path= RestoreSource{sourceDirectory=C:\Users\soontech\sunwaybackup, databaseName='sunway', type=Artifact, artifact chain=sunway [1-18] fro
m artifacts: [file:///C:/Users/soontech/sunwaybackup/sunway-2024-12-27T06-43-46.backup]}` completed successfully.
You need to execute C:\neo4j\data\scripts\sunway\restore_metadata.cypher. To execute the file use cypher-shell command with parameter `sunway`.
```

Figure 42: Database restoration in Neo4j.

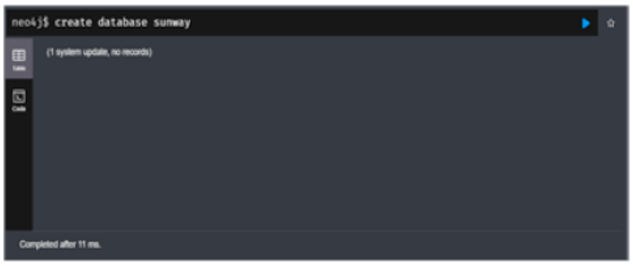
To restore a database in Neo4j, the Command Prompt was used to navigate to the bin directory of the Neo4j installation as shown in Figure **42**. Use the neo4j-admin tool with the database restore command. By specifying the path where the backup file is located and the name of the database, it can locate the file and restore the specified file. The command format is as follows:

neo4j-admin database restore --from-path=<path_name> <database_name>

In this case, to restore a database named 'sunway' from a backup stored in C:¥backup, the command would be:

neo4j-admin database restore --from-path=C:¥backup sunway

This command will restore everything that is in the database, including all nodes, relationships, and data from the backup file into the specified database. However, although the database has been successfully restored, it does not automatically appear in the Neo4j interface.



```
neo4j$ create database sunway

(1 system update, no records)

Completed after 11 ms.
```

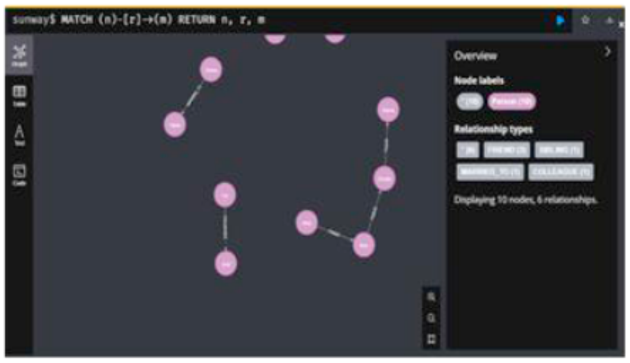
Figure 43: Recreation of database in Neo4j.

To ensure that the restored database becomes visible in the Neo4j interface, it must be recreated as shown in Figure **43**. This can be done using the *CREATE DATABASE* command. For example, if the database is named sunway, we need to create the 'sunway' database again by executing this command:

CREATE DATABASE sunway;

This creates the 'sunway' database in the Neo4j system, allowing it to appear in the interface for further interactions and queries.

5) Verification of Data Restoration



```
sunway$ MATCH (n)-[r]->(m) RETURN n, r, m
```

Figure 44: The process of verifying data restoration in Neo4j.

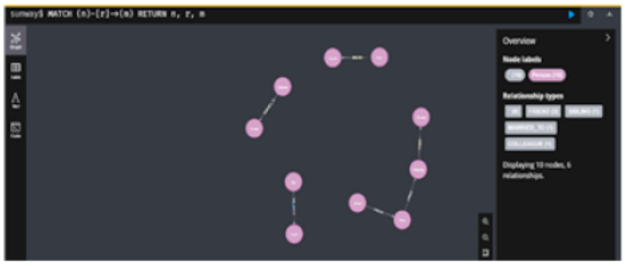
Figure **44** demonstrates the process of verifying data restoration in the database to ensure that all nodes and relationships are intact after restoration.

To show all nodes and relationships, we can use a Cypher query:

MATCH (n)-[r]->(m) RETURN n, r, m;

The reason of this step is to verify that the restoration process has been successful by checking the integrity and structure of the restored data.

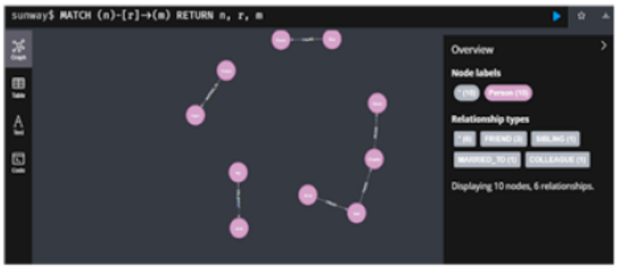
a) Comparison of the Original Database and Recovered Database



```
sunway$ MATCH (n)-[r]→(m) RETURN n, r, m
```

Figure 45: Original database.

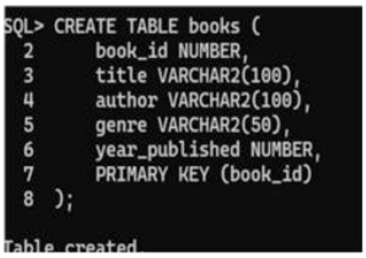
The comparison between the original (Figure **45**) and restored database (Figure **46**) confirms that the backup and restoration process is succesful. Both graphs show identical results, with 10 nodes and 6 relationships displayed in the "Overview" panel. The node labels and relationship types, remain consistent before and after restoration of data, which indicates that no data was loss during the backup and restoration process. The database was fully restored with a 100% accuracy, maintaining its original structure and content.



```
sunway$ MATCH (n)-[r]→(m) RETURN n, r, m
```

Figure 46: Recovered database.

D. Oracle

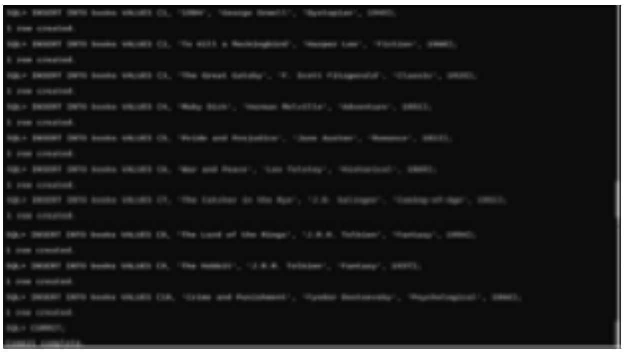


```
SQL> CREATE TABLE books (
  2      book_id NUMBER,
  3      title VARCHAR2(100),
  4      author VARCHAR2(100),
  5      genre VARCHAR2(50),
  6      year_published NUMBER,
  7      PRIMARY KEY (book_id)
  8  );

Table created.
```

Figure 47: The process of creating a table with the desired schema in SQL.

1) Data Creation

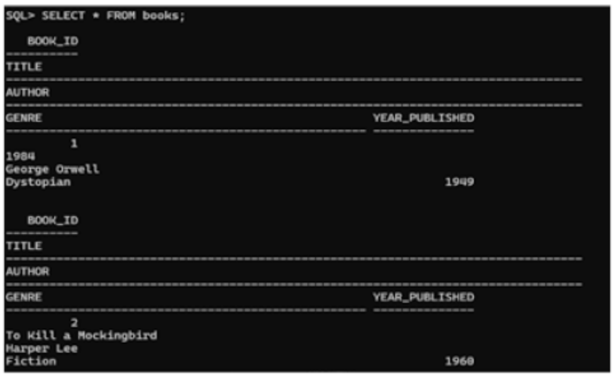
The books table is created with columns for book_id, title, author, genre, and year_published, as seen in Figure **47**. As the primary key, the book_id guarantees that every book is uniquely identified. The successful creation of the table is confirmed by the message "Table created."



```
SQL> INSERT INTO books VALUES (1, '1984', 'George Orwell', 'Dystopian', 1949);
1 row created.
SQL> INSERT INTO books VALUES (2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction', 1960);
1 row created.
SQL> INSERT INTO books VALUES (3, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', 1925);
1 row created.
SQL> INSERT INTO books VALUES (4, 'Moby Dick', 'Herman Melville', 'Adventure', 1851);
1 row created.
SQL> INSERT INTO books VALUES (5, 'Pride and Prejudice', 'Jane Austen', 'Romance', 1813);
1 row created.
SQL> INSERT INTO books VALUES (6, 'War and Peace', 'Leo Tolstoy', 'Historical', 1869);
1 row created.
SQL> INSERT INTO books VALUES (7, 'The Catcher in the Rye', 'J.D. Salinger', 'Coming-of-Age', 1951);
1 row created.
SQL> INSERT INTO books VALUES (8, 'The Lord of the Rings', 'J.R.R. Tolkien', 'Fantasy', 1954);
1 row created.
SQL> INSERT INTO books VALUES (9, 'The Hobbit', 'J.R.R. Tolkien', 'Fantasy', 1937);
1 row created.
SQL> INSERT INTO books VALUES (10, 'Crime and Punishment', 'Fyodor Dostoevsky', 'Psychological', 1866);
1 row created.
SQL> COMMIT;
Commit complete.
```

Figure 48: Data insertion in SQL.

Adding additional rows to the books table is shown in Figure **48**. Each insert statement's success is confirmed by "1 row created." The "Commit complete" message certifies that the process was completed, and the final *COMMIT;* command ensures that any changes are kept in the database forever.



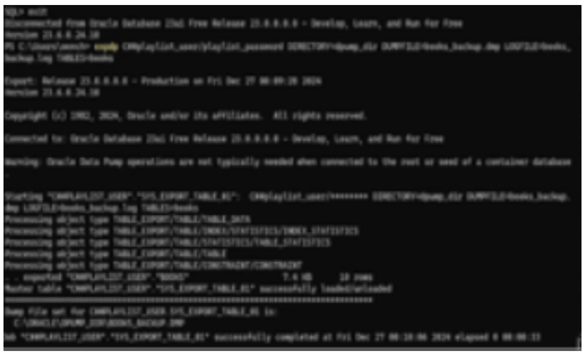
```
SQL> SELECT * FROM books;

   BOOK_ID
----------
TITLE
--------------------------------------------------------------------------------
AUTHOR
--------------------------------------------------------------------------------
GENRE                                              YEAR_PUBLISHED
-------------------------------------------------- --------------
         1
1984
George Orwell
Dystopian                                                    1949

   BOOK_ID
----------
TITLE
--------------------------------------------------------------------------------
AUTHOR
--------------------------------------------------------------------------------
GENRE                                              YEAR_PUBLISHED
-------------------------------------------------- --------------
         2
To Kill a Mockingbird
Harper Lee
Fiction                                                      1960
```

Figure 49: Verification of the inserted data.

In Figure **49**, a *SELECT * FROM books;* query is run to extract each entry and column from the books table. The results display the details of each book, such as BOOK_ID, TITLE, AUTHOR, GENRE, and YEAR_PUBLISHED. Consequently, the data's successful addition and compliance with the requirements are confirmed.

2) Data Backup

As shown in Figure **50**, a backup of the books table is made using the *expdp* (Export Data Pump) command. After SQL*Plus has been ended, the command supplies the directory, user credentials, dump file name (books_backup.dmp), and log file name (books_backup.log) for the export process.



```
SQL> exit
Disconnected from Oracle Database 23ai Free Release 23.0.0.0.0 - Develop, Learn, and Run for Free
Version 23.6.0.24.10
PS C:\Users\wench> expdp C##playlist_user/playlist_password DIRECTORY=dpump_dir DUMPFILE=books_backup.dmp LOGFILE=books_
backup.log TABLES=books

Export: Release 23.0.0.0.0 - Production on Fri Dec 27 00:09:20 2024
Version 23.6.0.24.10

Copyright (c) 1982, 2024, Oracle and/or its affiliates.  All rights reserved.

Connected to: Oracle Database 23ai Free Release 23.0.0.0.0 - Develop, Learn, and Run for Free

Warning: Oracle Data Pump operations are not typically needed when connected to the root or seed of a container database
.

Starting "C##PLAYLIST_USER"."SYS_EXPORT_TABLE_01":  C##playlist_user/******** DIRECTORY=dpump_dir DUMPFILE=books_backup.
dmp LOGFILE=books_backup.log TABLES=books
Processing object type TABLE_EXPORT/TABLE/TABLE_DATA
Processing object type TABLE_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
Processing object type TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
Processing object type TABLE_EXPORT/TABLE/TABLE
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
. . exported "C##PLAYLIST_USER"."BOOKS"                    7.4 KB      10 rows
Master table "C##PLAYLIST_USER"."SYS_EXPORT_TABLE_01" successfully loaded/unloaded
******************************************************************************
Dump file set for C##PLAYLIST_USER.SYS_EXPORT_TABLE_01 is:
  C:\ORACLE\DPUMP_DIR\BOOKS_BACKUP.DMP
Job "C##PLAYLIST_USER"."SYS_EXPORT_TABLE_01" successfully completed at Fri Dec 27 00:10:06 2024 elapsed 0 00:00:33
```

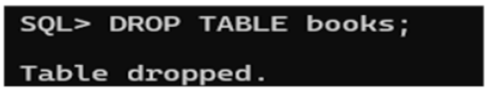
Figure 50: The process of exporting the books table using data pump.

The syntax for the command is as follows:

C##playlist_password/playlist_user expdp

The dump file is *DIRECTORY=dpump_dir Books_backup.dmp.* For books, *LOGFILE=backup.log TABLES = books*. The outcome confirms that the export process correctly processed all objects related to the books table, including table data, statistics, indexes, and constraints. The *BOOKS_BACKUP.DMP* file located in the specified directory contains a backup of the 10-row table. The last message shows how much time has elapsed since the export process, signifying that it was successfully finished. The table will be securely backed up and prepared for repair at this point.

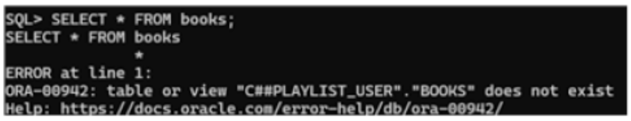
3) Data Loss Simulation



```
SQL> DROP TABLE books;

Table dropped.
```

Figure 51: Deletion of table.

Figure **51** shows how the *DROP TABLE* command is executed in SQL. This process prepares the ground for evaluating the system's restoration capabilities by simulating real-world scenarios where data loss could occur due to human error or unanticipated conditions.



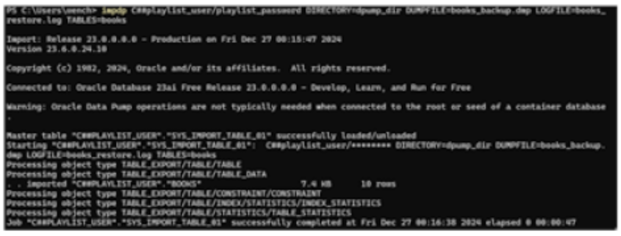
```
SQL> SELECT * FROM books;
SELECT * FROM books
              *
ERROR at line 1:
ORA-00942: table or view "C##PLAYLIST_USER"."BOOKS" does not exist
Help: https://docs.oracle.com/error-help/db/ora-00942/
```

Figure 52: The process of verifying table deletion.

Figure **52** uses the SELECT * FROM books; command to try to query the books table after it has been dumped. The error message that shows confirms that the table was successfully deleted from the database.

4) Data Restoration

As shown in Figure **53**, the impdp (Import Data Pump) function is used to restore the books table. For the restoration process, the command supplies the directory, user credentials, log file (books_restore.log), and dump file (books_backup.dmp). The command in use has the following syntax:



```
PS C:\Users\wench> impdp C##playlist_user/playlist_password DIRECTORY=dpump_dir DUMPFILE=books_backup.dmp LOGFILE=books_
restore.log TABLES=books

Import: Release 23.0.0.0.0 - Production on Fri Dec 27 00:15:47 2024
Version 23.6.0.24.10

Copyright (c) 1982, 2024, Oracle and/or its affiliates.  All rights reserved.

Connected to: Oracle Database 23ai Free Release 23.0.0.0.0 - Develop, Learn, and Run for Free

Warning: Oracle Data Pump operations are not typically needed when connected to the root or seed of a container database
.

Master table "C##PLAYLIST_USER"."SYS_IMPORT_TABLE_01" successfully loaded/unloaded
Starting "C##PLAYLIST_USER"."SYS_IMPORT_TABLE_01":  C##playlist_user/******** DIRECTORY=dpump_dir DUMPFILE=books_backup.
dmp LOGFILE=books_restore.log TABLES=books
Processing object type TABLE_EXPORT/TABLE/TABLE
Processing object type TABLE_EXPORT/TABLE/TABLE_DATA
. . imported "C##PLAYLIST_USER"."BOOKS"                    7.4 KB      10 rows
Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Processing object type TABLE_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS
Processing object type TABLE_EXPORT/TABLE/STATISTICS/TABLE_STATISTICS
Job "C##PLAYLIST_USER"."SYS_IMPORT_TABLE_01" successfully completed at Fri Dec 27 00:16:38 2024 elapsed 0 00:00:47
```

Figure 53: The process of restoring the books table using data pump.

C##playlist_user impdp/playlist_password Books_backup

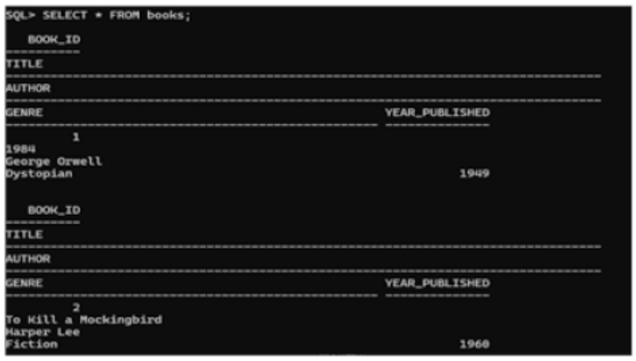
The directory is *dpump_dir*.The dump file is called *Dmp*. The log file, Books_restore.log, is *TABLES = books*.

The report confirms that the books table, including its data and structure, was successfully imported after 10 rows in total were restored. It handles all necessary elements, like as constraints, statistics, and indexes, to guarantee that the table is fully operational during restoration. On "Fri Dec 27 00:16:38 2024," the job "SYS_IMPORT_TABLE_01" was successfully finished after 47 seconds, according to the final message. This stage confirms that the backup file can successfully restore missing data and structure.

5) Verification of Data Restoration

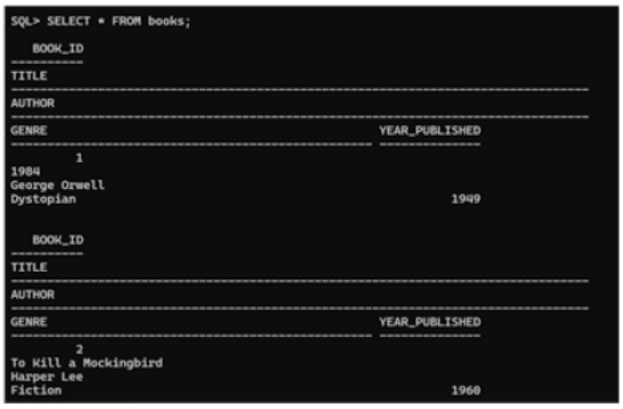
The table's owner used SQL*Plus to confirm that the table restoration process had been successful. Then, queries were executed against the repaired table to retrieve all of the data. This verification step was done to ensure data integrity by confirming that the restoration process had successfully restored all of the data and that the table's schema was still intact.

a) Comparison of the Content of the Original Table and Recovered Table



```
SQL> SELECT * FROM books;

   BOOK_ID
----------
TITLE
--------------------------------------------------------------------------------
AUTHOR
--------------------------------------------------------------------------------
GENRE                                                YEAR_PUBLISHED
-------------------------------------------------- --------------
         1
1984
George Orwell
Dystopian                                                      1949

   BOOK_ID
----------
TITLE
--------------------------------------------------------------------------------
AUTHOR
--------------------------------------------------------------------------------
GENRE                                                YEAR_PUBLISHED
-------------------------------------------------- --------------
         2
To Kill a Mockingbird
Harper Lee
Fiction                                                        1960
```

Figure 54: Content of the original table.

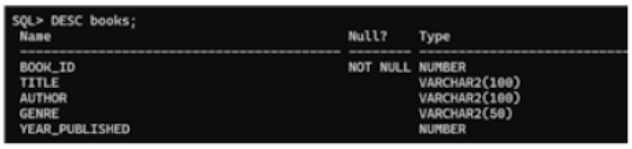


```
SQL> SELECT * FROM books;

   BOOK_ID
----------
TITLE
--------------------------------------------------------------------------------
AUTHOR
--------------------------------------------------------------------------------
GENRE                                              YEAR_PUBLISHED
-------------------------------------------------- --------------
         1
1984
George Orwell
Dystopian                                                    1949

   BOOK_ID
----------
TITLE
--------------------------------------------------------------------------------
AUTHOR
--------------------------------------------------------------------------------
GENRE                                              YEAR_PUBLISHED
-------------------------------------------------- --------------
         2
To Kill a Mockingbird
Harper Lee
Fiction                                                      1960
```

Figure 55: Content of the recovered table.

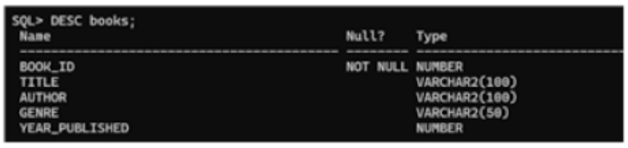
The comparison of Figure **54** and Figure **55** shows that the restoration process was successfully finished. As you can see from both tables, the original data for the two book entries is present, demonstrating the data preservation feature. Furthermore, the names of the columns and data types seem to be the same, indicating that the table's structure was maintained. Based on this data, we may say that the "books" table was successfully restored.

b) Comparison of the Structure of the Original Table and Recovered Table



```
SQL> DESC books;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 BOOK_ID                                   NOT NULL NUMBER
 TITLE                                              VARCHAR2(100)
 AUTHOR                                             VARCHAR2(100)
 GENRE                                              VARCHAR2(50)
 YEAR_PUBLISHED                                     NUMBER
```

Figure 56: The structure of the original table.

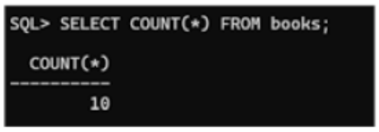


```
SQL> DESC books;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 BOOK_ID                                   NOT NULL NUMBER
 TITLE                                              VARCHAR2(100)
 AUTHOR                                             VARCHAR2(100)
 GENRE                                              VARCHAR2(50)
 YEAR_PUBLISHED                                     NUMBER
```

Figure 57: The structure of the recovered table.

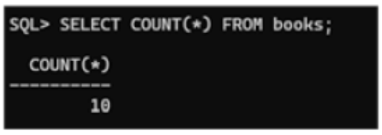
The recovered table in Figure **57** preserves the exact same schema as the original table in Figure **56**, as shown by the "DESC books;" command. Nullability requirements, data types, and column names are all the same. This finding demonstrates that the restoration procedure maintained the table's structural integrity in addition to its data, guaranteeing that the recovered table will operate as intended.

c) Comparison of the Number of Rows of the Original Table and Recovered Table



```
SQL> SELECT COUNT(*) FROM books;

  COUNT(*)
----------
        10
```

Figure 58: The number of rows of the original table.



```
SQL> SELECT COUNT(*) FROM books;

  COUNT(*)
----------
        10
```

Figure 59: The number of rows of the recovered table.

Both Figure **58** and Figure **59** display the results of the *SELECT COUNT(*) FROM books;* query, indicating the number of rows in the "books" table. The label *"COUNT(*)"* is returned by both with the value "10." The fact that this outcome is the same for both the original and recovered tables indicates that the number of rows was actually maintained and that no data was lost or duplicated during the recovery procedure.

A successful restore requires confirming the accuracy of the data content, table structure, and row count of the "books" table. Since the recovered table is a perfect duplicate of the original data, the data is considered intact. Schema integrity is preserved since the table structure, including column names, data types, and nullability restrictions, is identical to the original. Lastly, nothing was lost or duplicated because the number of rows in the recovered table matches the number of rows in the original. All things considered, these results demonstrate that the "books" table was successfully recovered and that the revised table accurately replicates the original.

Based on Table **1**, all the database systems restored the data completely with no loss and no

Table 1: DBMS Data and Schema Recovery Analysis

DBMS	Data Recovery (%)	Schema Recovery (%)	Key Observations
Cassandra	100	100	Reliable snapshot-based recovery; requires careful directory management for snapshots.
MongoDB	100	100	Efficient BSON-based recovery; intuitive GUI support with Compass.
Neo4j	100	100	Accurate graph data restoration; additional step required to register the restored database in Neo4j.
Oracle	100	100	Comprehensive schema and data restoration; robust logging in Data Pump utilities for debugging.

duplication, hence showing integrity in the data. Schema integrity was also upheld across all systems, with no discrepancies observed between the original and restored schemas. On usability, MongoDB has quite an intuitive and user-friendly recovery process through its GUI tools; however, the Neo4j restoration requires extra steps to finalize the registration of the restored database.

E. Overall Discussion

1) Key Findings

The study showed that all four DBMSs—Cassandra, MongoDB, Neo4j, and Oracle—achieved 100% recovery efficiency in the restoration of data and schemas. This result shows that these systems are reliable in disaster recovery scenarios, regardless of the data model used.

a) Cassandra and Neo4j Specialization:

- Cassandra excelled in distributed data recovery, making it a first choice for systems managing vast datasets across many nodes.
- Neo4j, on the other hand, showed strength in graph data recovery, making it ideal for applications relying on complex relationships, such as social networks or recommendation systems.

b) Structured and Semi-Structured Data Recovery:

- Oracle performed exceptionally well in recovering structured data, consistent with its reputation for robust enterprise solutions.
- MongoDB exhibited strong performance in semi-structured data recovery, leveraging its flexible document-based architecture to simplify the restoration process.

2) Implications

These findings have strong implications for DBMS selection based on the specific recovery needs of various applications:

- Distributed Systems: Cassandra's scalability and distributed architecture make it an ideal solution for disaster recovery in high-throughput, large-scale environments. Its ability to handle node failures without compromising data integrity reinforces its suitability.
- Semi-Structured Data: MongoDB's BSON-based structure and recovery mechanisms offer significant advantages for the rapid recovery of semi-structured data, particularly in e-commerce and IoT environments.
- Graph-Based Applications: Neo4j's specialized graph restoration capabilities are crucial for maintaining complex relationships, as required in domains such as fraud detection and supply chain analysis.
- Enterprise Settings: Oracle's comprehensive toolset and strong error-handling capabilities make it highly suitable for enterprise-level applications that demand reliable recovery of structured data.

3) Limitations

The findings should be considered in light of the following limitations:

- Synthetic Datasets: The experiments used synthetic datasets to mimic real-world scenarios. However, these datasets may not fully capture the characteristics and variability of real production environments.
- Concurrent Workload Performance: The study focused solely on data and schema recovery, excluding evaluations of DBMS performance under concurrent workloads, which are critical in practical applications.
- Neo4j Registration Overhead: Neo4j's recovery process involved an additional manual step for database registration, increasing the recovery time compared to other DBMSs.

F. Visual Representation of Results

Table 2: DBMS Data Restoration Summary

DBMS	Rows Restored	Schema Match
Cassandra	10	Yes
MongoDB	100	Yes
Neo4j	10 nodes, 6 rels	Yes
Oracle	10	Yes

Table **2** highlights the uniform success across all systems in achieving full recovery. Bar chart in Figure **60** visually demonstrates recovery percentages, reinforcing the robustness of the evaluated techniques.

Verifying the recovery mechanisms for the chosen DBMSs was the goal of the paper. The outcomes demonstrate their ability to guarantee full data and schema recovery. The following were the strengths of each system:

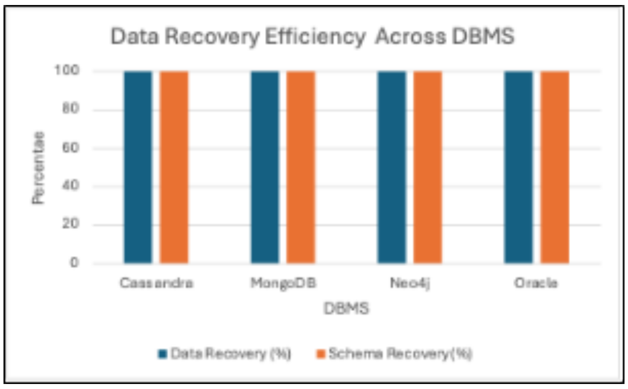


Figure 60: Bar Chart Illustrating Recovery Efficiency Across DBMS.

- MongoDB: It was incredibly simple to use and recover, especially for semi-structured data, because to its user-friendly GUI.
- Oracle: Even under simulated failure conditions, a smooth recovery was guaranteed by its robust error management.
- Cassandra: It effectively recovers data across nodes by utilizing its distributed architecture.
- Neo4j: It shows excellent graph restoration, but it necessitates additional database registration steps, which may restrict its usability.

G. Linking to Literature Review

The results are in line with earlier research showing that recovery methods ought to be tailored to various data models, such as:

- Snapshot Techniques: Studies demonstrating the effectiveness of snapshot techniques in distributed situations are supported by Cassandra's dependence on the snapshot-based recovery mechanism.
- Document-based Recovery: Previous research on the benefits of MongoDB's adaptable data model is supported by the database's efficient processing of BSON documents.
- Graph Restore: Research that has advocated for specialized tools in graph-based systems is consistent with Neo4j's capacity to restore complex relationships.
- Utility Support: Oracle's extensive suite of recovery tools reflects its established standing as a reliable enterprise-class database management system.

In conclusion, the study found that backup and recovery systems in Cassandra, MongoDB, Neo4j, and Oracle are dependable; all evaluated systems returned to a 100% recovery efficiency, and neither data nor schema integrity was compromised. These findings provide practitioners with important information about how to select DBMSs depending on the specific context and recovery needs. The constraints must be addressed in future studies, primarily through testing in real-world settings and with concurrent workloads.

5. CONCLUSIONS AND FUTURE WORK

A. Summary of Key Findings

This paper elaborated on backup and recovery processes of four mainstream DBMSs, which were Oracle, Cassandra, MongoDB, and Neo4j. Four test systems demonstrated 100% recovery efficiency as a result of successful complete data recovery along with schema recovery.

- Due to its nature of being distributed in the case of data recovery, Cassandra is all advantageous in environments featuring high-intended data dispersion.
- Neo4j appeared to be very productive during the recovery of graph data. Such data is normally very critical regarding linkages within the context of graph-based applications.
- MongoDB recovered semi-structured data very efficiently due to its flexible document-oriented schema design and very user-friendly recovery commands.
- Oracle did recover structured data in an expected way due to its enterprise level and built-in strong error-handling capacity.

The findings above illustrate the flexibility and efficiency of DBMSs while working on a wide range of data formats and application scenarios.

B. Link to Objectives

The following are the answers for each of the objectives included within the introductory section of the paper:

- Evaluate the Recovery Mechanisms of Key DBMSs:

Indeed, the paper shows that Cassandra, MongoDB, Neo4j, and Oracle can all guarantee full recoveries of data and schema and hence guarantees against failure.

- Identify Strengths and Limitations:

Also, the strength points identified were Cassandra's scalability and ease of use for MongoDB, while Neo4j did have additional database registration procedures, adding onto its list of weaknesses.

- Provide Practical Insights for DBMS Selection:

It also gives some useful advice regarding the suitability of each DBMS to a certain operating context and recovery performance.

To this end, the current study provides a useful basis from which one could understand and comparatively analyze state-of-the-art recovery techniques.

C. Contribution of Research

- Comparative Analysis:

The paper carries out an elaborate comparison of the backup and recovery characteristics of four varied database management systems. Indeed, through making comparisons across different systems against one other, this is an area in which not so many research works have individually looked at different DBMSs. It casts more light on how different systems behave under similar recovery scenarios and hence forms a useful milestone for further research.

- Insights into Specialized Recovery:

Recovery techniques have to be tailored for each data model. The paper introduces the recovery-related mechanisms in the case of distributed systems, semi-structured data, graph databases, and relational data. The paper shows what each of the mentioned data models poses in terms of challenges and opportunities and suggests the recovery techniques more in line with the peculiarities of such models.

- Practical Guidance:

The provided paper gave real hints to database administrators and organizations from a practical perspective. That allowed practitioners to identify the most appropriate DBMSs, given their matching of unique recovery and operational needs that they have in place. For this reason, the study couples technical functionalities with business priorities; therefore, the paper helps decision-makers to raise the reliability of the system while optimizing resource utilization.

D. Practical Implications

- Distributed Systems:

It has thrown light on the high performance of Cassandra, hence proving it to be the best fit in large-scale systems that need high availability and fault tolerance. It thus makes Cassandra suitable for the following real-world scenarios: financial networks needing to stay up and running at any moment in time, content delivery platforms that need to propagate data with great speed, and IoT infrastructures with gigantic volumes of real-time data coming in from geographically dispersed devices.

- Semi-structured Data Recovery:

This ease of recoveries makes MongoDB quite a prime choice for industries dealing in semi-structured data. Speedy recoveries make it quite relevant for e-commerce platforms which need timely updates on inventory, health care systems keeping patients' records and diagnostic information, and logistic operations dependent upon dynamic tracking and routing information.

- Graph-Based Applications:

Neo4j returns power to the field of graph restoration for those applications that want to have complex relationships. In restoration capability, fraud detection requires fast rebuilding of patterns and their relations; recommendation systems make use of user-relationship data, and supply chain management totally depends on the complex networked relationships that are required between suppliers, manufacturers, and distributors.

- Enterprise Use Cases:

The strong performance of Oracle in recovering structured data denotes why it plays a crucial role in mission-critical enterprise applications. It is highly applicable in the field of banking, which wants high accuracy and integrity of data, and in telecommunication, which wants robustness and scalability of database systems, including government agencies that demand robust recovery solutions for keeping public services running and information secure.

E. Limitations

- Synthetic Datasets:

Limitations include that the experiments are based on synthetic data sets. These are convenient in allowing controlled and reproducible tests but may not capture all the complexities of real-world phenomena. These include support for a wide range of data formats and handling inconsistencies, scaling up to different data volumes, which may impinge differently on recovery performance when considering practical situations.

- Workload Concurrency Analysis:

It does not address the impact of parallel workloads on the recovery performance; hence, this remains the critical deficiency in recovery testing of the reliability of a DBMS in real-time, high-demand situations. The recovery processes running in an environment with concurrent read/write quotas, such as e-commerce sites during peak hours or financial trading systems, may behave differently and thus reduce the generalization capability of the findings.

- Cost and Time Efficiency:

This study focuses on recovery accuracy and doesn't go into the discussion of the recovery mechanisms from the viewpoint of cost and time efficiency. Still, these are also crucial dimensions that organizations consider during the deployment and maintenance of such recovery solutions. This is because scalability of the process may be challenged due to the utilisation of resource and budget and, moreover, its feasibility in the time scale deployment of a small scale.

F. Future Research Directions

- Real-World Data Usage:

The next studies shall conduct the evaluation of recovery mechanisms using real-world data under realistic and complicated settings so that practical insight can be derived into how the recovery processes behave concerning different formats, inconsistencies, and multiple scales of data so that such findings are more useful for practical applications across industry.

- Testing for Concurrent Workload:

This shall help attain the requirement of real-time systems. The aspect that could be studied in future studies is how the recovery mechanisms cope with the simultaneous read and write operation, and this shows another dimension of their reliability and efficiency in heavy demand applications, such as financial trading systems and online marketplaces.

- Cost and Time Analysis:

The research will in future encompass the capabilities of having cost and time efficiency metrics that enable assessment across all dimensions of recovery solutions. This trade-off, in turn a researcher can use to present to an organization the practical implications of a particular recovery strategy that hence can make effective decisions.

- Analysis of emergent DBMSs:

This scope will be extended towards emergent and hybrid DBMS technologies. This extended research will capture how recent trends in that particular area address the challenges of recovery. The study in the future shall focus on the capability of the new systems in regard to handling distributed, semi-structured, multi-model data with further enhancements captured in this comparison.

- Automation in Recovery:

Another very promising avenue in increasing the efficiency involves research in automation and AI usage in database recovery. Investigations shall work out towards minimizing manual intervention in database recovery, streamlining the recovery workflows, and minimizing downtimes for the highly dynamic and mission-critical environments.

G. Concluding Remarks

This is where the research has persuasively established the reliability of the backup and recovery processes for Oracle, Cassandra, MongoDB, and Neo4j. The 100% recovery efficiency of the systems showed that they were reliable to handle disaster situations. The result gives useful, practical recommendations with additional information on the advantages and disadvantages of various DBMS. It even opens up new directions of research.

The importance of the work is in filling the gap between theoretical recovery methods and their practical application, which would contribute to accelerating further efforts toward the improvement of disaster recovery processes when dealing with difficult and challenging data environments.

APPENDIX

APPENDIX

Cassandra:

```
samuellee@Samuels-MacBook-Air ~ % cqlsh

Connected to Test Cluster at 127.0.0.1:9042

[cqlsh 6.2.0 | Cassandra 5.0.2 | CQL spec 3.4.7 | Native protocol v5]

Use HELP for help.

cqlsh> CREATE KEYSPACE original_playlist_keyspace

   ... WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

cqlsh> USE original_playlist_keyspace;

cqlsh:original_playlist_keyspace> CREATE TABLE playlists(

                    ...   ...   id uuid,

                    ...   ...   song_order int,

                    ...   ...   song_id uuid,

                    ...   ...   title text,

                    ...   ...   album text,

                    ...   ...   artist text,

                    ...   ...   PRIMARY KEY (id, song_order)

                    ...

cqlsh:original_playlist_keyspace> CREATE TABLE playlists (

                    ...   id uuid,

                    ...   song_order int,

                    ...   song_id uuid,

                    ...   title text,

                    ...   album text,

                    ...   artist text,

                    ...   PRIMARY KEY (id, song_order)

                    ... );

cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 1, uuid(), 'Shape of You', 'Divide', 'Ed Sheeran');

cqlsh:original_playlist_keyspace>

cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 2, uuid(), 'Blinding Lights', 'After Hours', 'The Weeknd');

cqlsh:original_playlist_keyspace>

cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 3, uuid(), 'Levitating', 'Future Nostalgia', 'Dua Lipa');

cqlsh:original_playlist_keyspace>

cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 4, uuid(), 'Peaches', 'Justice', 'Justin Bieber');

cqlsh:original_playlist_keyspace>

cqlsh:original_playlist_keyspace> INSERT INTO playlists (id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 5, uuid(), 'Good 4 U', 'SOUR', 'Olivia Rodrigo');
```

```
cqlsh:original_playlist_keyspace>

cqlsh:original_playlist_keyspace> INSERT INTO playlists
(id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 6, uuid(),
'Montero', 'Montero', 'Lil Nas X');

cqlsh:original_playlist_keyspace>

cqlsh:original_playlist_keyspace> INSERT INTO playlists
(id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 7, uuid(), 'Industry
Baby', 'Montero', 'Lil Nas X');

cqlsh:original_playlist_keyspace>

cqlsh:original_playlist_keyspace> INSERT INTO playlists
(id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 8, uuid(), 'Bad
Bunny', 'YHLQMDLG', 'Bad Bunny');

cqlsh:original_playlist_keyspace>

cqlsh:original_playlist_keyspace> INSERT INTO playlists
(id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 9, uuid(),
'Watermelon Sugar', 'Fine Line', 'Harry Styles');

cqlsh:original_playlist_keyspace>

cqlsh:original_playlist_keyspace> INSERT INTO playlists
(id, song_order, song_id, title, album, artist)

                    ... VALUES (uuid(), 10, uuid(), 'Save
Your Tears', 'After Hours', 'The Weeknd');

cqlsh:original_playlist_keyspace> SELECT * FROM
playlists;


 id                                   | song_order | album            | artist
| song_id                              | title

-------------------------------------+------------+------------------
+---------------+-------------------------------------+------------
------

 5a870286-be78-48ae-814c-94181cb604ec |          2 |
After Hours |     The Weeknd | f8e83152-48ba-4611-b6e4-
a7946b9532d8 |  Blinding Lights

 e1a7ed69-396c-43e8-8a1f-2cbe81f17949 |         10 |
After Hours |     The Weeknd | 3bfffe4e-37e3-4792-9789-
c23cd7e21a0d |  Save Your Tears

 db808708-423f-40ce-80f6-b466645d9519 |          6 |
Montero |        Lil Nas X | 4117203d-fdff-4413-9aae-
6f9949aa807f |       Montero

 e8e667be-ae38-4588-86bf-f0fe157e58d9 |          7 |
Montero |        Lil Nas X | d920f70e-fc0c-4ad5-b97a-
1e9747dccd5d |   Industry Baby

 d83f8c2e-e013-4aea-acac-891424309d83 |          9 |
Fine Line |    Harry Styles | 56204e28-9826-4d9a-aaaf-
c8f026569a7c | Watermelon Sugar

 d95447fb-b876-42a4-9e12-bf467ab5db17 |          3 |
Future Nostalgia |        Dua Lipa | 2a2d381f-9f3d-456e-
8dc2-fc4ead6d5efe |      Levitating

 0a0041c3-3980-4117-a06e-038deb258288 |          8 |
YHLQMDLG |        Bad Bunny | 44d85cde-8474-4bc8-
af12-812d64d2aae5 |      Bad Bunny

 d385aeae-b717-449c-a220-26596a6efad3 |          1 |
Divide |         Ed Sheeran | 7bdd3cc2-150e-48fd-bdb5-
43f491442443 |    Shape of You

 7f823f17-fc61-41a6-907b-1c733d605a37 |          5 |
SOUR | Olivia Rodrigo | 27170d2a-592b-4de0-a8c4-
fcce642bba84 |       Good 4 U

 8604a47d-d6d3-4777-9ed2-24c874bc256d |          4 |
Justice |    Justin Bieber | e1a7130c-4ca6-43bd-9c93-
e46e844ad78d |       Peaches



(10 rows)

cqlsh:original_playlist_keyspace> DESCRIBE TABLE
playlists;



CREATE TABLE original_playlist_keyspace.playlists (

    id uuid,

    song_order int,

    album text,

    artist text,

    song_id uuid,

    title text,
    PRIMARY KEY (id, song_order)

) WITH CLUSTERING ORDER BY (song_order ASC)

    AND additional_write_policy = '99p'

    AND allow_auto_snapshot = true

    AND bloom_filter_fp_chance = 0.01

    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}

    AND cdc = false

    AND comment = ''

    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}

    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}

    AND memtable = 'default'

    AND crc_check_chance = 1.0

    AND default_time_to_live = 0

    AND extensions = {}

    AND gc_grace_seconds = 864000

    AND incremental_backups = true

    AND max_index_interval = 2048

    AND memtable_flush_period_in_ms = 0

    AND min_index_interval = 128

    AND read_repair = 'BLOCKING'

    AND speculative_retry = '99p';

cqlsh:original_playlist_keyspace> exit

samuellee@Samuels-MacBook-Air ~ % nodetool snapshot original_playlist_keyspace

Requested creating snapshot(s) for [original_playlist_keyspace] with snapshot name [1733827590035] and options {skipFlush=false}

Snapshot directory: 1733827590035

samuellee@Samuels-MacBook-Air ~ % cd /opt/homebrew/var/lib/cassandra/data/original_playlist_keyspace

samuellee@Samuels-MacBook-Air original_playlist_keyspace % ls

playlists-1225e700b6e311ef9ca6431a51005ad3

samuellee@Samuels-MacBook-Air original_playlist_keyspace % cd playlists-1225e700b6e311ef9ca6431a51005ad3

samuellee@Samuels-MacBook-Air playlists-1225e700b6e311ef9ca6431a51005ad3 % ls snapshots

1733827590035

samuellee@Samuels-MacBook-Air playlists-1225e700b6e311ef9ca6431a51005ad3 % cd snapshots/1733827590035 ls

cd: string not in pwd: snapshots/1733827590035

samuellee@Samuels-MacBook-Air playlists-1225e700b6e311ef9ca6431a51005ad3 % cd snapshots/1733827590035

samuellee@Samuels-MacBook-Air 1733827590035 % cp -r /opt/homebrew/var/lib/cassandra/data/original_playlist_keyspace/playlists-1225e700b6e311ef9ca6431a51005ad3/snapshots/1733827590035 ~/backup/cassandra_snapshots/

samuellee@Samuels-MacBook-Air 1733827590035 % ls ~/backup/cassandra_snapshots/1733827590035

manifest.json                   nb-1-big-Filter.db
        nb-1-big-TOC.txt

nb-1-big-CompressionInfo.db     nb-1-big-Index.db
        schema.cql

nb-1-big-Data.db        nb-1-big-Statistics.db

nb-1-big-Digest.crc32           nb-1-big-Summary.db

samuellee@Samuels-MacBook-Air 1733827590035 % cqlsh

Connected to Test Cluster at 127.0.0.1:9042

[cqlsh 6.2.0 | Cassandra 5.0.2 | CQL spec 3.4.7 | Native protocol v5]

Use HELP for help.
```

```
cqlsh> DROP KEYSPACE original_playlist_keyspace;

cqlsh> USE original_playlist_keyspace;

InvalidRequest: Error from server: code=2200 [Invalid query] message="Keyspace 'original_playlist_keyspace' does not exist"

cqlsh> CREATE KEYSPACE recovered_playlist_keyspace

  ... WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};

cqlsh> USE recovered_playlist_keyspace;

cqlsh:recovered_playlist_keyspace>

cqlsh:recovered_playlist_keyspace> CREATE TABLE playlists (

                    ...    id uuid,

                    ...    song_order int,

                    ...    song_id uuid,

                    ...    title text,

                    ...    album text,

                    ...    artist text,

                    ...    PRIMARY KEY (id, song_order)

                    ... );

cqlsh:recovered_playlist_keyspace> exit

samuellee@Samuels-MacBook-Air 1733827590035 % cp -r ~/backup/cassandra_snapshots/1733827590035/* /opt/homebrew/var/lib/cassandra/data/recovered_playlist_keyspace/playlists-1225e700b6e311ef9ca6431a51005ad3/

cp: /opt/homebrew/var/lib/cassandra/data/recovered_playlist_keyspace/playlists-1225e700b6e311ef9ca6431a51005ad3 is not a directory

samuellee@Samuels-MacBook-Air 1733827590035 % cp -r ~/backup/cassandra_snapshots/1733827590035/* /opt/homebrew/var/lib/cassandra/data/original_playlist_keyspace/playlists-1225e700b6e311ef9ca6431a51005ad3/

samuellee@Samuels-MacBook-Air 1733827590035 % nodetool import recovered_playlist_keyspace playlists /opt/homebrew/var/lib/cassandra/data/original_playlist_keyspace/playlists-1225e700b6e311ef9ca6431a51005ad3/

samuellee@Samuels-MacBook-Air 1733827590035 % cqlsh

Connected to Test Cluster at 127.0.0.1:9042

[cqlsh 6.2.0 | Cassandra 5.0.2 | CQL spec 3.4.7 | Native protocol v5]

Use HELP for help.

cqlsh> USE recovered_playlist_keyspace;

cqlsh:recovered_playlist_keyspace> SELECT * FROM playlists;


 id                                   | song_order | album           | artist | song_id                              | title

--------------------------------------+------------+-----------------+----------------+--------------------------------------+------------------

 5a870286-be78-48ae-814c-94181cb604ec |          2 | After Hours |    The Weeknd | f8e83152-48ba-4611-b6e4-a7946b9532d8 |  Blinding Lights

 e1a7ed69-396c-43e8-8a1f-2cbe81f17949 |         10 | After Hours |    The Weeknd | 3bfffe4e-37e3-4792-9789-c23cd7e21a0d |  Save Your Tears

 db808708-423f-40ce-80f6-b466645d9519 |          6 | Montero |       Lil Nas X | 4117203d-fdff-4413-9aae-6f9949aa807f |         Montero

 e8e667be-ae38-4588-86bf-f0fe157e58d9 |          7 | Montero |       Lil Nas X | d920f70e-fc0c-4ad5-b97a-1e9747dccd5d |    Industry Baby

 d83f8c2e-e013-4aea-acac-891424309d83 |          9 | Fine Line |    Harry Styles | 56204e28-9826-4d9a-aaaf-c8f026569a7c | Watermelon Sugar

 d95447fb-b876-42a4-9e12-bf467ab5db17 |          3 | Future Nostalgia |       Dua Lipa | 2a2d381f-9f3d-456e-8dc2-fc4ead6d5efe |       Levitating

 0a0041c3-3980-4117-a06e-038deb258288 |          8 | YHLQMDLG |       Bad Bunny | 44d85cde-8474-4bc8-af12-812d64d2aae5 |        Bad Bunny

 d385aeae-b717-449c-a220-26596a6efad3 |          1 | Divide |      Ed Sheeran | 7bdd3cc2-150e-48fd-bdb5-43f491442443 |     Shape of You
 7f823f17-fc61-41a6-907b-1c733d605a37 |               5 |
SOUR | Olivia Rodrigo | 27170d2a-592b-4de0-a8c4-
fcce642bba84 |        Good 4 U

 8604a47d-d6d3-4777-9ed2-24c874bc256d |               4 |
Justice | Justin Bieber | e1a7130c-4ca6-43bd-9c93-
e46e844ad78d |        Peaches


(10 rows)
```

```
cqlsh:recovered_playlist_keyspace> DESCRIBE TABLE
playlists;


CREATE TABLE recovered_playlist_keyspace.playlists (

    id uuid,

    song_order int,

    album text,

    artist text,

    song_id uuid,

    title text,

    PRIMARY KEY (id, song_order)

) WITH CLUSTERING ORDER BY (song_order ASC)

    AND additional_write_policy = '99p'

    AND allow_auto_snapshot = true

    AND bloom_filter_fp_chance = 0.01

    AND caching = {'keys': 'ALL', 'rows_per_partition':
'NONE'}

    AND cdc = false

    AND comment = ''

    AND compaction = {'class':
'org.apache.cassandra.db.compaction.SizeTieredCompacti
onStrategy', 'max_threshold': '32', 'min_threshold': '4'}

    AND compression = {'chunk_length_in_kb': '16', 'class':
'org.apache.cassandra.io.compress.LZ4Compressor'}

    AND memtable = 'default'

    AND crc_check_chance = 1.0

    AND default_time_to_live = 0

    AND extensions = {}

    AND gc_grace_seconds = 864000

    AND incremental_backups = true

    AND max_index_interval = 2048

    AND memtable_flush_period_in_ms = 0

    AND min_index_interval = 128

    AND read_repair = 'BLOCKING'

    AND speculative_retry = '99p';
```

MongoDB:

Database creation:

Create ‘myDatabase1’ database.

Create ‘myCollection1’ collection under ‘myDatabase1’.

Data creation:

```
[

  { "CustomerID": 1, "CustomerName": "Customer1",
"ContactEmail": "customer1@example.com", "JoinDate":
"2023-01-01", "Orders": ["Order1-A", "Order1-B"] },

  { "CustomerID": 2, "CustomerName": "Customer2",
"ContactEmail": "customer2@example.com", "JoinDate":
"2023-01-02", "Orders": ["Order2-A", "Order2-B"] },

  { "CustomerID": 3, "CustomerName": "Customer3",
"ContactEmail": "customer3@example.com", "JoinDate":
"2023-01-03", "Orders": ["Order3-A", "Order3-B"] },

  { "CustomerID": 4, "CustomerName": "Customer4",
"ContactEmail": "customer4@example.com", "JoinDate":
"2023-01-04", "Orders": ["Order4-A", "Order4-B"] },

  { "CustomerID": 5, "CustomerName": "Customer5",
"ContactEmail": "customer5@example.com", "JoinDate":
"2023-01-05", "Orders": ["Order5-A", "Order5-B"] },
```

{ "CustomerID": 6, "CustomerName": "Customer6", "ContactEmail": "customer6@example.com", "JoinDate": "2023-01-06", "Orders": ["Order6-A", "Order6-B"] },

{ "CustomerID": 7, "CustomerName": "Customer7", "ContactEmail": "customer7@example.com", "JoinDate": "2023-01-07", "Orders": ["Order7-A", "Order7-B"] },

{ "CustomerID": 8, "CustomerName": "Customer8", "ContactEmail": "customer8@example.com", "JoinDate": "2023-01-08", "Orders": ["Order8-A", "Order8-B"] },

{ "CustomerID": 9, "CustomerName": "Customer9", "ContactEmail": "customer9@example.com", "JoinDate": "2023-01-09", "Orders": ["Order9-A", "Order9-B"] },

{ "CustomerID": 10, "CustomerName": "Customer10", "ContactEmail": "customer10@example.com", "JoinDate": "2023-01-10", "Orders": ["Order10-A", "Order10-B"] },

{ "CustomerID": 11, "CustomerName": "Customer11", "ContactEmail": "customer11@example.com", "JoinDate": "2023-01-11", "Orders": ["Order11-A", "Order11-B"] },

{ "CustomerID": 12, "CustomerName": "Customer12", "ContactEmail": "customer12@example.com", "JoinDate": "2023-01-12", "Orders": ["Order12-A", "Order12-B"] },

{ "CustomerID": 13, "CustomerName": "Customer13", "ContactEmail": "customer13@example.com", "JoinDate": "2023-01-13", "Orders": ["Order13-A", "Order13-B"] },

{ "CustomerID": 14, "CustomerName": "Customer14", "ContactEmail": "customer14@example.com", "JoinDate": "2023-01-14", "Orders": ["Order14-A", "Order14-B"] },

{ "CustomerID": 15, "CustomerName": "Customer15", "ContactEmail": "customer15@example.com", "JoinDate": "2023-01-15", "Orders": ["Order15-A", "Order15-B"] },

{ "CustomerID": 16, "CustomerName": "Customer16", "ContactEmail": "customer16@example.com", "JoinDate": "2023-01-16", "Orders": ["Order16-A", "Order16-B"] },

{ "CustomerID": 17, "CustomerName": "Customer17", "ContactEmail": "customer17@example.com", "JoinDate": "2023-01-17", "Orders": ["Order17-A", "Order17-B"] },

{ "CustomerID": 18, "CustomerName": "Customer18", "ContactEmail": "customer18@example.com", "JoinDate": "2023-01-18", "Orders": ["Order18-A", "Order18-B"] },

{ "CustomerID": 19, "CustomerName": "Customer19", "ContactEmail": "customer19@example.com", "JoinDate": "2023-01-19", "Orders": ["Order19-A", "Order19-B"] },

{ "CustomerID": 20, "CustomerName": "Customer20", "ContactEmail": "customer20@example.com", "JoinDate": "2023-01-20", "Orders": ["Order20-A", "Order20-B"] },

{ "CustomerID": 21, "CustomerName": "Customer21", "ContactEmail": "customer21@example.com", "JoinDate": "2023-01-21", "Orders": ["Order21-A", "Order21-B"] },

{ "CustomerID": 22, "CustomerName": "Customer22", "ContactEmail": "customer22@example.com", "JoinDate": "2023-01-22", "Orders": ["Order22-A", "Order22-B"] },

{ "CustomerID": 23, "CustomerName": "Customer23", "ContactEmail": "customer23@example.com", "JoinDate": "2023-01-23", "Orders": ["Order23-A", "Order23-B"] },

{ "CustomerID": 24, "CustomerName": "Customer24", "ContactEmail": "customer24@example.com", "JoinDate": "2023-01-24", "Orders": ["Order24-A", "Order24-B"] },

{ "CustomerID": 25, "CustomerName": "Customer25", "ContactEmail": "customer25@example.com", "JoinDate": "2023-01-25", "Orders": ["Order25-A", "Order25-B"] },

{ "CustomerID": 26, "CustomerName": "Customer26", "ContactEmail": "customer26@example.com", "JoinDate": "2023-01-26", "Orders": ["Order26-A", "Order26-B"] },

{ "CustomerID": 27, "CustomerName": "Customer27", "ContactEmail": "customer27@example.com", "JoinDate": "2023-01-27", "Orders": ["Order27-A", "Order27-B"] },

{ "CustomerID": 28, "CustomerName": "Customer28", "ContactEmail": "customer28@example.com", "JoinDate": "2023-01-28", "Orders": ["Order28-A", "Order28-B"] },

{ "CustomerID": 29, "CustomerName": "Customer29", "ContactEmail": "customer29@example.com", "JoinDate": "2023-01-29", "Orders": ["Order29-A", "Order29-B"] },

{ "CustomerID": 30, "CustomerName": "Customer30", "ContactEmail": "customer30@example.com", "JoinDate": "2023-01-30", "Orders": ["Order30-A", "Order30-B"] },

{ "CustomerID": 31, "CustomerName": "Customer31", "ContactEmail": "customer31@example.com", "JoinDate": "2023-01-31", "Orders": ["Order31-A", "Order31-B"] },

{ "CustomerID": 32, "CustomerName": "Customer32", "ContactEmail": "customer32@example.com", "JoinDate": "2023-02-01", "Orders": ["Order32-A", "Order32-B"] },

{ "CustomerID": 33, "CustomerName": "Customer33", "ContactEmail": "customer33@example.com", "JoinDate": "2023-02-02", "Orders": ["Order33-A", "Order33-B"] },

{ "CustomerID": 34, "CustomerName": "Customer34", "ContactEmail": "customer34@example.com", "JoinDate": "2023-02-03", "Orders": ["Order34-A", "Order34-B"] },

{ "CustomerID": 35, "CustomerName": "Customer35", "ContactEmail": "customer35@example.com", "JoinDate": "2023-02-04", "Orders": ["Order35-A", "Order35-B"] },

{ "CustomerID": 36, "CustomerName": "Customer36", "ContactEmail": "customer36@example.com", "JoinDate": "2023-02-05", "Orders": ["Order36-A", "Order36-B"] },

{ "CustomerID": 37, "CustomerName": "Customer37", "ContactEmail": "customer37@example.com", "JoinDate": "2023-02-06", "Orders": ["Order37-A", "Order37-B"] },

{ "CustomerID": 38, "CustomerName": "Customer38", "ContactEmail": "customer38@example.com", "JoinDate": "2023-02-07", "Orders": ["Order38-A", "Order38-B"] },

{ "CustomerID": 39, "CustomerName": "Customer39", "ContactEmail": "customer39@example.com", "JoinDate": "2023-02-08", "Orders": ["Order39-A", "Order39-B"] },

{ "CustomerID": 40, "CustomerName": "Customer40", "ContactEmail": "customer40@example.com", "JoinDate": "2023-02-09", "Orders": ["Order40-A", "Order40-B"] },

{ "CustomerID": 41, "CustomerName": "Customer41", "ContactEmail": "customer41@example.com", "JoinDate": "2023-02-10", "Orders": ["Order41-A", "Order41-B"] },

{ "CustomerID": 42, "CustomerName": "Customer42", "ContactEmail": "customer42@example.com", "JoinDate": "2023-02-11", "Orders": ["Order42-A", "Order42-B"] },

{ "CustomerID": 43, "CustomerName": "Customer43", "ContactEmail": "customer43@example.com", "JoinDate": "2023-02-12", "Orders": ["Order43-A", "Order43-B"] },

{ "CustomerID": 44, "CustomerName": "Customer44", "ContactEmail": "customer44@example.com", "JoinDate": "2023-02-13", "Orders": ["Order44-A", "Order44-B"] },

{ "CustomerID": 45, "CustomerName": "Customer45", "ContactEmail": "customer45@example.com", "JoinDate": "2023-02-14", "Orders": ["Order45-A", "Order45-B"] },

{ "CustomerID": 46, "CustomerName": "Customer46", "ContactEmail": "customer46@example.com", "JoinDate": "2023-02-15", "Orders": ["Order46-A", "Order46-B"] },

{ "CustomerID": 47, "CustomerName": "Customer47", "ContactEmail": "customer47@example.com", "JoinDate": "2023-02-16", "Orders": ["Order47-A", "Order47-B"] },

{ "CustomerID": 48, "CustomerName": "Customer48", "ContactEmail": "customer48@example.com", "JoinDate": "2023-02-17", "Orders": ["Order48-A", "Order48-B"] },

{ "CustomerID": 49, "CustomerName": "Customer49", "ContactEmail": "customer49@example.com", "JoinDate": "2023-02-18", "Orders": ["Order49-A", "Order49-B"] },

{ "CustomerID": 50, "CustomerName": "Customer50", "ContactEmail": "customer50@example.com", "JoinDate": "2023-02-19", "Orders": ["Order50-A", "Order50-B"] },

{ "CustomerID": 51, "CustomerName": "Customer51", "ContactEmail": "customer51@example.com", "JoinDate": "2023-02-20", "Orders": ["Order51-A", "Order51-B"] },

{ "CustomerID": 52, "CustomerName": "Customer52", "ContactEmail": "customer52@example.com", "JoinDate": "2023-02-21", "Orders": ["Order52-A", "Order52-B"] },

{ "CustomerID": 53, "CustomerName": "Customer53", "ContactEmail": "customer53@example.com", "JoinDate": "2023-02-22", "Orders": ["Order53-A", "Order53-B"] },

{ "CustomerID": 54, "CustomerName": "Customer54", "ContactEmail": "customer54@example.com", "JoinDate": "2023-02-23", "Orders": ["Order54-A", "Order54-B"] },

{ "CustomerID": 55, "CustomerName": "Customer55", "ContactEmail": "customer55@example.com", "JoinDate": "2023-02-24", "Orders": ["Order55-A", "Order55-B"] },

{ "CustomerID": 56, "CustomerName": "Customer56", "ContactEmail": "customer56@example.com", "JoinDate": "2023-02-25", "Orders": ["Order56-A", "Order56-B"] },

{ "CustomerID": 57, "CustomerName": "Customer57", "ContactEmail": "customer57@example.com", "JoinDate": "2023-02-26", "Orders": ["Order57-A", "Order57-B"] },

{ "CustomerID": 58, "CustomerName": "Customer58", "ContactEmail": "customer58@example.com", "JoinDate": "2023-02-27", "Orders": ["Order58-A", "Order58-B"] },

{ "CustomerID": 59, "CustomerName": "Customer59", "ContactEmail": "customer59@example.com", "JoinDate": "2023-02-28", "Orders": ["Order59-A", "Order59-B"] },

{ "CustomerID": 60, "CustomerName": "Customer60", "ContactEmail": "customer60@example.com", "JoinDate": "2023-03-01", "Orders": ["Order60-A", "Order60-B"] },

{ "CustomerID": 61, "CustomerName": "Customer61", "ContactEmail": "customer61@example.com", "JoinDate": "2023-03-02", "Orders": ["Order61-A", "Order61-B"] },

{ "CustomerID": 62, "CustomerName": "Customer62", "ContactEmail": "customer62@example.com", "JoinDate": "2023-03-03", "Orders": ["Order62-A", "Order62-B"] },

{ "CustomerID": 63, "CustomerName": "Customer63", "ContactEmail": "customer63@example.com", "JoinDate": "2023-03-04", "Orders": ["Order63-A", "Order63-B"] },

{ "CustomerID": 64, "CustomerName": "Customer64", "ContactEmail": "customer64@example.com", "JoinDate": "2023-03-05", "Orders": ["Order64-A", "Order64-B"] },

{ "CustomerID": 65, "CustomerName": "Customer65", "ContactEmail": "customer65@example.com", "JoinDate": "2023-03-06", "Orders": ["Order65-A", "Order65-B"] },

{ "CustomerID": 66, "CustomerName": "Customer66", "ContactEmail": "customer66@example.com", "JoinDate": "2023-03-07", "Orders": ["Order66-A", "Order66-B"] },

{ "CustomerID": 67, "CustomerName": "Customer67", "ContactEmail": "customer67@example.com", "JoinDate": "2023-03-08", "Orders": ["Order67-A", "Order67-B"] },

{ "CustomerID": 68, "CustomerName": "Customer68", "ContactEmail": "customer68@example.com", "JoinDate": "2023-03-09", "Orders": ["Order68-A", "Order68-B"] },

{ "CustomerID": 69, "CustomerName": "Customer69", "ContactEmail": "customer69@example.com", "JoinDate": "2023-03-10", "Orders": ["Order69-A", "Order69-B"] },

{ "CustomerID": 70, "CustomerName": "Customer70", "ContactEmail": "customer70@example.com", "JoinDate": "2023-03-11", "Orders": ["Order70-A", "Order70-B"] },

{ "CustomerID": 71, "CustomerName": "Customer71", "ContactEmail": "customer71@example.com", "JoinDate": "2023-03-12", "Orders": ["Order71-A", "Order71-B"] },

{ "CustomerID": 72, "CustomerName": "Customer72", "ContactEmail": "customer72@example.com", "JoinDate": "2023-03-13", "Orders": ["Order72-A", "Order72-B"] },

{ "CustomerID": 73, "CustomerName": "Customer73", "ContactEmail": "customer73@example.com", "JoinDate": "2023-03-14", "Orders": ["Order73-A", "Order73-B"] },

{ "CustomerID": 74, "CustomerName": "Customer74", "ContactEmail": "customer74@example.com", "JoinDate": "2023-03-15", "Orders": ["Order74-A", "Order74-B"] },

{ "CustomerID": 75, "CustomerName": "Customer75", "ContactEmail": "customer75@example.com", "JoinDate": "2023-03-16", "Orders": ["Order75-A", "Order75-B"] },

{ "CustomerID": 76, "CustomerName": "Customer76", "ContactEmail": "customer76@example.com", "JoinDate": "2023-03-17", "Orders": ["Order76-A", "Order76-B"] },

{ "CustomerID": 77, "CustomerName": "Customer77", "ContactEmail": "customer77@example.com", "JoinDate": "2023-03-18", "Orders": ["Order77-A", "Order77-B"] },

{ "CustomerID": 78, "CustomerName": "Customer78", "ContactEmail": "customer78@example.com", "JoinDate": "2023-03-19", "Orders": ["Order78-A", "Order78-B"] },

{ "CustomerID": 79, "CustomerName": "Customer79", "ContactEmail": "customer79@example.com", "JoinDate": "2023-03-20", "Orders": ["Order79-A", "Order79-B"] },

{ "CustomerID": 80, "CustomerName": "Customer80", "ContactEmail": "customer80@example.com", "JoinDate": "2023-03-21", "Orders": ["Order80-A", "Order80-B"] },

{ "CustomerID": 81, "CustomerName": "Customer81", "ContactEmail": "customer81@example.com", "JoinDate": "2023-03-22", "Orders": ["Order81-A", "Order81-B"] },

{ "CustomerID": 82, "CustomerName": "Customer82", "ContactEmail": "customer82@example.com", "JoinDate": "2023-03-23", "Orders": ["Order82-A", "Order82-B"] },

{ "CustomerID": 83, "CustomerName": "Customer83", "ContactEmail": "customer83@example.com", "JoinDate": "2023-03-24", "Orders": ["Order83-A", "Order83-B"] },

{ "CustomerID": 84, "CustomerName": "Customer84", "ContactEmail": "customer84@example.com", "JoinDate": "2023-03-25", "Orders": ["Order84-A", "Order84-B"] },

{ "CustomerID": 85, "CustomerName": "Customer85", "ContactEmail": "customer85@example.com", "JoinDate": "2023-03-26", "Orders": ["Order85-A", "Order85-B"] },

{ "CustomerID": 86, "CustomerName": "Customer86", "ContactEmail": "customer86@example.com", "JoinDate": "2023-03-27", "Orders": ["Order86-A", "Order86-B"] },

{ "CustomerID": 87, "CustomerName": "Customer87", "ContactEmail": "customer87@example.com", "JoinDate": "2023-03-28", "Orders": ["Order87-A", "Order87-B"] },

{ "CustomerID": 88, "CustomerName": "Customer88", "ContactEmail": "customer88@example.com", "JoinDate": "2023-03-29", "Orders": ["Order88-A", "Order88-B"] },

{ "CustomerID": 89, "CustomerName": "Customer89", "ContactEmail": "customer89@example.com", "JoinDate": "2023-03-30", "Orders": ["Order89-A", "Order89-B"] },

{ "CustomerID": 90, "CustomerName": "Customer90", "ContactEmail": "customer90@example.com", "JoinDate": "2023-03-31", "Orders": ["Order90-A", "Order90-B"] },

{ "CustomerID": 91, "CustomerName": "Customer91", "ContactEmail": "customer91@example.com", "JoinDate": "2023-04-01", "Orders": ["Order91-A", "Order91-B"] },

{ "CustomerID": 92, "CustomerName": "Customer92", "ContactEmail": "customer92@example.com", "JoinDate": "2023-04-02", "Orders": ["Order92-A", "Order92-B"] },

{ "CustomerID": 93, "CustomerName": "Customer93", "ContactEmail": "customer93@example.com", "JoinDate": "2023-04-03", "Orders": ["Order93-A", "Order93-B"] },

{ "CustomerID": 94, "CustomerName": "Customer94", "ContactEmail": "customer94@example.com", "JoinDate": "2023-04-04", "Orders": ["Order94-A", "Order94-B"] },

{ "CustomerID": 95, "CustomerName": "Customer95", "ContactEmail": "customer95@example.com", "JoinDate": "2023-04-05", "Orders": ["Order95-A", "Order95-B"] },

```
{ "CustomerID": 96, "CustomerName": "Customer96", "ContactEmail": "customer96@example.com", "JoinDate": "2023-04-06", "Orders": ["Order96-A", "Order96-B"] },

{ "CustomerID": 97, "CustomerName": "Customer97", "ContactEmail": "customer97@example.com", "JoinDate": "2023-04-07", "Orders": ["Order97-A", "Order97-B"] },

{ "CustomerID": 98, "CustomerName": "Customer98", "ContactEmail": "customer98@example.com", "JoinDate": "2023-04-08", "Orders": ["Order98-A", "Order98-B"] },

{ "CustomerID": 99, "CustomerName": "Customer99", "ContactEmail": "customer99@example.com", "JoinDate": "2023-04-09", "Orders": ["Order99-A", "Order99-B"] },

{ "CustomerID": 100, "CustomerName": "Customer100", "ContactEmail": "customer100@example.com", "JoinDate": "2023-04-10", "Orders": ["Order100-A", "Order100-B"] }

]
```

Database backup:

```
C:\Program Files\MongoDB\Tools\100\bin>mongodump --db myDatabase1 --out "C:\Users\soonteck\mongodbtestbackup"

2024-12-27T15:33:57.151+0800 writing myDatabase1.myCollection1 to C:\Users\soonteck\mongodbtestbackup\myDatabase1\myCollection1.bson

2024-12-27T15:33:57.174+0800 done dumping myDatabase1.myCollection1 (100 documents)
```

Database recovery:

```
C:\Program Files\MongoDB\Tools\100\bin>mongorestore --db myDatabase1 "C:\Users\soonteck\mongodbtestbackup\myDatabase1"

2024-12-27T15:37:14.265+0800 The --db and --collection flags are deprecated for this use-case; please use --nsInclude instead, i.e. with --nsInclude=${DATABASE}.${COLLECTION}

2024-12-27T15:37:14.285+0800 building a list of collections to restore from C:\Users\soonteck\mongodbtestbackup\myDatabase1 dir

2024-12-27T15:37:14.286+0800 reading metadata for myDatabase1.myCollection1 from C:\Users\soonteck\mongodbtestbackup\myDatabase1\myCollection1.metadata.json

2024-12-27T15:37:14.306+0800 restoring myDatabase1.myCollection1 from C:\Users\soonteck\mongodbtestbackup\myDatabase1\myCollection1.bson

2024-12-27T15:37:14.322+0800 finished restoring myDatabase1.myCollection1 (100 documents, 0 failures)

2024-12-27T15:37:14.322+0800 no indexes to restore for collection myDatabase1.myCollection1

2024-12-27T15:37:14.323+0800 100 document(s) restored successfully. 0 document(s) failed to restore.
```

Neo4j:

Create data:

Insert people:

```
CREATE (p1:Person {name: 'Alice', age: 25}),

    (p2:Person {name: 'Bob', age: 30}),

    (p3:Person {name: 'Charlie', age: 35}),

    (p4:Person {name: 'Diana', age: 28}),

    (p5:Person {name: 'Eve', age: 22}),

    (p6:Person {name: 'Frank', age: 40}),

    (p7:Person {name: 'Grace', age: 27}),

    (p8:Person {name: 'Hank', age: 32}),

    (p9:Person {name: 'Ivy', age: 24}),

    (p10:Person {name: 'Jack', age: 29});
```

Insert relationship:

```
// Friendships

MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'})

CREATE (a)-[:FRIEND]->(b);



MATCH (b:Person {name: 'Bob'}), (c:Person {name: 'Charlie'})

CREATE (b)-[:FRIEND]->(c);
```

```
MATCH (c:Person {name: 'Charlie'}), (d:Person {name: 'Diana'})

CREATE (c)-[:FRIEND]->(d);



// Family relationships

MATCH (e:Person {name: 'Eve'}), (f:Person {name: 'Frank'})

CREATE (e)-[:SIBLING]->(f);



MATCH (g:Person {name: 'Grace'}), (h:Person {name: 'Hank'})

CREATE (g)-[:MARRIED_TO]->(h);



// Work relationships

MATCH (i:Person {name: 'Ivy'}), (j:Person {name: 'Jack'})

CREATE (i)-[:COLLEAGUE]->(j);
```

Database backup:

```
C:\neo4j\bin>neo4j-admin database backup --to-path="C:\Users\soonteck\sunwaybackup" sunway

2024-12-27 06:43:45.356+0000 INFO [c.n.b.b.BackupOutputMonitor] Starting backup of database 'sunway' from servers: [127.0.0.1:6362]

2024-12-27 06:43:46.155+0000 INFO [c.n.b.b.BackupOutputMonitor] Start backup of database 'sunway'.

2024-12-27 06:43:46.254+0000 INFO [c.n.b.b.BackupOutputMonitor] Using remote server 127.0.0.1:6362 for backup of database 'sunway'.

2024-12-27 06:43:46.286+0000 INFO [c.n.b.b.BackupOutputMonitor] Start differential backup of database 'sunway'.

2024-12-27 06:43:46.286+0000 INFO [c.n.b.b.BackupOutputMonitor] Differential backup of database 'sunway' failed. Reason: Differential backups require that a full backup of the same database exists in the folder defined in --to-path. No existing backup found here: C:\Users\soonteck\sunwaybackup.

2024-12-27 06:43:46.286+0000 INFO [c.n.b.b.BackupOutputMonitor] Falling back to full backup of database 'sunway'.

2024-12-27 06:43:46.286+0000 INFO [c.n.b.b.BackupOutputMonitor] Start full backup of database 'sunway'.

2024-12-27 06:43:46.386+0000 INFO [c.n.b.b.BackupOutputMonitor] Start receiving store files for database 'sunway', took 106ms.

2024-12-27 06:43:46.557+0000 INFO [c.n.b.b.BackupOutputMonitor] Finished receiving store files for database 'sunway'.

2024-12-27 06:43:46.559+0000 INFO [c.n.b.b.BackupOutputMonitor] Start receiving database 'sunway' transactions from [18, 19].

2024-12-27 06:43:46.759+0000 INFO [c.n.b.b.BackupOutputMonitor] Finished receiving transactions for database 'sunway'; took 170ms.

2024-12-27 06:43:46.759+0000 INFO [c.n.b.b.BackupOutputMonitor] Finished full backup of database 'sunway'. Downloaded from tx -1 to tx 18.

2024-12-27 06:43:46.941+0000 INFO [c.n.b.b.BackupOutputMonitor] Start recovering database 'sunway'.

2024-12-27 06:43:47.123+0000 INFO [c.n.b.b.BackupOutputMonitor] Finished recovering database 'sunway', took 182ms.

2024-12-27 06:43:48.198+0000 INFO [c.n.b.b.BackupOutputMonitor] Start creating artifact 'incomplete_backup.tmp' for database 'sunway'.

2024-12-27 06:43:48.467+0000 INFO [c.n.b.b.BackupOutputMonitor] Finished artifact creation 'sunway-2024-12-27T06-43-46.backup' for database 'sunway', took 274ms.

2024-12-27 06:43:48.482+0000 INFO [c.n.b.b.BackupOutputMonitor] Backup of database 'sunway' completed, took 2s 325ms.



Backup command completed.
```

Database recovery:

```
C:\neo4j\bin>neo4j-admin database restore --from-
path="C:\Users\soonteck\sunwaybackup\sunway-2024-12-
27T06-43-46.backup" sunway

C:\neo4j\bin>2024-12-27 06:47:45.424+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Starting to restore
RestoreSource{sourceDirectory=C:\Users\soonteck\sunwa
ybackup, databaseName='sunway', type=Artifact, artifact
chain=sunway [1-18] from artifacts:
[file:///C:/Users/soonteck/sunwaybackup/sunway-2024-12-
27T06-43-46.backup]}

2024-12-27 06:47:45.489+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Created temporary
directory for extracting artifacts

2024-12-27 06:47:45.499+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Unpacking database
from
RestoreSource{sourceDirectory=C:\Users\soonteck\sunwa
ybackup, databaseName='sunway', type=Artifact, artifact
chain=sunway [1-18] from artifacts:
[file:///C:/Users/soonteck/sunwaybackup/sunway-2024-12-
27T06-43-46.backup]}

2024-12-27 06:47:45.511+0000 INFO
[c.n.b.c.BackupArtifactService] Unpacking full backup
artifact: file:///C:/Users/soonteck/sunwaybackup/sunway-
2024-12-27T06-43-46.backup into directory:
PlainDatabaseLayout{databaseDirectory=C:\Users\soontec
k\sunwaybackup\sunway-temp-extracted-artifacts-0,
transactionLogsDirectory=C:\Users\soonteck\sunwayback
up\sunway-temp-extracted-artifacts-0}

2024-12-27 06:47:45.895+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Start recovering
database 'sunway'.

2024-12-27 06:47:45.915+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Finish recovering
database 'sunway', took 18ms.

2024-12-27 06:47:45.927+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Start recovering
database 'sunway'.

2024-12-27 06:47:45.935+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Finish recovering
database 'sunway', took 7ms.

2024-12-27 06:47:45.938+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Finished unpacking
database from
RestoreSource{sourceDirectory=C:\Users\soonteck\sunwa
ybackup, databaseName='sunway', type=Artifact, artifact
chain=sunway [1-18] from artifacts:
[file:///C:/Users/soonteck/sunwaybackup/sunway-2024-12-
27T06-43-46.backup]}

2024-12-27 06:47:45.938+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Cleaning target
directories [directories: [C:\neo4j\data\databases\sunway,
C:\neo4j\data\transactions\sunway]]

2024-12-27 06:47:45.940+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Restoring files
[from:C:\Users\soonteck\sunwaybackup\sunway-temp-
extracted-artifacts-0,
to:PlainDatabaseLayout{databaseDirectory=C:\neo4j\data\
databases\sunway,
transactionLogsDirectory=C:\neo4j\data\transactions\sunw
ay}]

2024-12-27 06:47:45.988+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] You need to execute
C:\neo4j\data\scripts\sunway\restore_metadata.cypher. To
execute the file use cypher-shell command with parameter
'sunway'

2024-12-27 06:47:45.989+0000 INFO
[c.n.b.r.RestoreDatabaseExecutor] Restoring of files
completed

Restore of database 'sunway' from
path="RestoreSource{sourceDirectory=C:\Users\soonteck\
sunwaybackup, databaseName='sunway', type=Artifact,
artifact chain=sunway [1-18] from artifacts:
[file:///C:/Users/soonteck/sunwaybackup/sunway-2024-12-
27T06-43-46.backup]}" completed successfully.
```

Oracle:

Create table

```
CREATE TABLE books (

   book_id NUMBER,

   title VARCHAR2(100),

   author VARCHAR2(100),

   genre VARCHAR2(50),

   year_published NUMBER,

   PRIMARY KEY (book_id)

);



SQL> INSERT INTO books VALUES (1, '1984', 'George Orwell', 'Dystopian', 1949);

1 row created.



SQL> INSERT INTO books VALUES (2, 'To Kill a Mockingbird', 'Harper Lee', 'Fiction', 1960);

1 row created.



SQL> INSERT INTO books VALUES (3, 'The Great Gatsby', 'F. Scott Fitzgerald', 'Classic', 1925);

1 row created.



SQL> INSERT INTO books VALUES (4, 'Moby Dick', 'Herman Melville', 'Adventure', 1851);

1 row created.



SQL> INSERT INTO books VALUES (5, 'Pride and Prejudice', 'Jane Austen', 'Romance', 1813);

1 row created.

SQL> INSERT INTO books VALUES (6, 'War and Peace', 'Leo Tolstoy', 'Historical', 1869);

1 row created.



SQL> INSERT INTO books VALUES (7, 'The Catcher in the Rye', 'J.D. Salinger', 'Coming-of-Age', 1951);

1 row created.



SQL> INSERT INTO books VALUES (8, 'The Lord of the Rings', 'J.R.R. Tolkien', 'Fantasy', 1954);

1 row created.



SQL> INSERT INTO books VALUES (9, 'The Hobbit', 'J.R.R. Tolkien', 'Fantasy', 1937);

1 row created.



SQL> INSERT INTO books VALUES (10, 'Crime and Punishment', 'Fyodor Dostoevsky', 'Psychological', 1866);

1 row created.



SQL> COMMIT;

Commit complete.



SQL> exit

Disconnected from Oracle Database 23c Free Release 23.0.0.0.0 - Develop, Learn, and Run for Free

PS C:\Users\user> expdp
CMyPlayList_user/playlist_password
DIRECTORY=dump_dir
DUMPFILE=books_backup.DMP
LOGFILE=books_backup.LOG TABLES=books



Export: Release 23.0.0.0.0 - Production on Fri Dec 27 09:00:28 2024
```

```
Version 23.6.0.24.10

Copyright (c) 1982, 2024, Oracle and/or its affiliates. All rights reserved.

Connected to: Oracle Database 23c Free Release 23.0.0.0.0 - Develop, Learn, and Run for Free

Warning: Oracle Data Pump operations are not typically needed when connected to the root or seed of a container database.

Starting "CMyPlayList_user"."SYS_EXPORT_TABLE_01": CMyPlayList_user/******** DIRECTORY=dump_dir DUMPFILE=books_backup.DMP LOGFILE=books_backup.LOG TABLES=books

Processing object type TABLE_EXPORT/TABLE/TABLE_DATA

Processing object type TABLE_EXPORT/TABLE/INDEX/STATISTICS/INDEX_STATISTICS

Processing object type TABLE_EXPORT/TABLE/INDEX/STATISTICS/TABLE_STATISTICS

Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/CONSTRAINT

. . exported "CMYPLAYLIST_USER"."BOOKS" 7.4 KB 10 rows

Master table "CMyPlayList_user"."SYS_EXPORT_TABLE_01" successfully loaded/unloaded

******************************************************************************

Dump file set for CMyPlayList_user.SYS_EXPORT_TABLE_01 is:

 C:\ORACLE\DUMP_DIR\BOOKS_BACKUP.DMP

Job "CMyPlayList_user"."SYS_EXPORT_TABLE_01" successfully completed at Fri Dec 27 09:10:06 2024 elapsed 0 00:00:33

PS C:\Users\wench> impdp C#playlist_user/playlist_password DIRECTORY=dpump_dir DUMPFILE=books_backup.dmp LOGFILE=books_restore.log TABLES=books

Import: Release 23.0.0.0.0 - Production on Fri Dec 27 00:15:47 2024

Version 23.6.0.24.10

Copyright (c) 1982, 2024, Oracle and/or its affiliates. All rights reserved.

Connected to: Oracle Database 23cai Free Release 23.0.0.0.0 - Develop, Learn, and Run for Free

Warning: Oracle Data Pump operations are not typically needed when connected to the root or seed of a container database.

Master table "C#PLAYLIST_USER"."SYS_IMPORT_TABLE_01" successfully loaded/unloaded

Starting "C#PLAYLIST_USER"."SYS_IMPORT_TABLE_01": C#playlist_user/******** DIRECTORY=dpump_dir DUMPFILE=books_backup.dmp LOGFILE=books_restore.log TABLES=books

Processing object type TABLE_EXPORT/TABLE/TABLE

Processing object type TABLE_EXPORT/TABLE/TABLE_DATA

. . imported "C#PLAYLIST_USER"."BOOKS" 7.4 KB 10 rows

Processing object type TABLE_EXPORT/TABLE/CONSTRAINT/CONSTRAINT
Processing                    object                    type
TABLE_EXPORT/TABLE/INDEX/STATISTICS/INDE
X_STATISTICS

Processing                    object                    type
TABLE_EXPORT/TABLE/STATISTICS/TABLE_STAT
ISTICS

Job
"C#PLAYLIST_USER"."SYS_IMPORT_TABLE_01"
successfully  completed  at  Fri  Dec  27  00:16:38  2024
elapsed 0 00:00:47
```

REFERENCES

[1] Singh and J. Battra, "Strategies for Data Backup and Recovery in the Cloud," International Journal of Performability Engineering, vol. 19, no. 11, pp. 728-735, 2023, https://doi.org/10.23940/ijpe.23.11.p3.728735

[2] G. Ramesh, J.Logeshwaran and V.Aravindarajan, "A Secured Database Monitoring Method to Improve Data Backup and Recovery," BOHR International Journal of Operations in Cloud Computing, vol. 2, no. 1, pp. 1-7, January 2023, https://doi.org/10.54646/bijcs.019

[3] V. Javaraiah, "Backup for cloud and disaster recovery for consumers and SMBs," 2011. https://ieeexplore.ieee.org/document/6163671 (accessed Mar. 06, 2020).

[4] K. Bohora, A. Bothe and D. S. a. R. Chopade, "Backup and Recovery Mechanisms of Cassandra Database: A Review," The Journal of Digital Forensics, Security and Law, 2021, https://doi.org/10.15394/jdfsl.2021.1613

[5] I. Kuyumdzhiev, "Backup and recovery of MongoDB database: features, state, problems," Journal of The Union of Scientists - Varna, Economic Sciences Series, no. 1, pp. 125-133, January 2015.

[6] M. F. P.F and R. K. a. S. M. Varghese, "Outcome Analysis Using Neo4j Graph Database," International Journal on Cybernetics & Informatics, vol. 5, no. 2, pp. 229-236, April 2016, https://doi.org/10.5121/ijci.2016.5225

[7] J.-H. Choi and D. W. J. a. S. Lee, "The method of recovery for deleted record in Oracle Database," Journal of the Korea Institute of Information Security and Cryptology, vol. 23, no. 5, pp. 947-955, October 2013, https://doi.org/10.13089/JKIISC.2013.23.5.947

[8] F. Y. H. Ahmed and R. S. a. M. Abdullah, "Enhancement of E-Commerce Database System During the COVID-19 Pandemic," 2021 IEEE 11th IEEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE), 2021, https://doi.org/10.1109/ISCAIE51753.2021.9431804

[9] P. Lu, L. Zhang, X. Liu and J. Y. a. Z. Zhu, "Highly efficient data migration and backup for big data applications in elastic optical inter-data-center networks," IEEE Network, 2015, https://doi.org/10.1109/MNET.2015.7293303

[10] R. PLAKA, "Backup & Data Recovery in Cloud Computing: A Systematic Mapping Study," Ingenious, vol. 2, no. 1, pp. 94-113, January 2022, https://doi.org/10.58944/pwhk4843

[11] M. Z. Hasan, N. Sarwar, I. Alam, M. Z. Hussain and A. A. S. a. A. Irshad, "Data Recovery and Backup Management: A Cloud Computing Impact," 2023 IEEE International Conference on Emerging Trends in Engineering, Sciences and Technology (ICES&T), 2023, https://doi.org/10.1109/ICEST56843.2023.10138852

[12] A. K. a. P. Sehgal, "{BARNS}: Towards Building Backup and Recovery for NoSQL Databases," January 2017.

[13] H. A. Mumtahana, "Optimization of Transaction Database Design with MySQL and MongoDB," SinkrOn, vol. 7, no. 3, pp. 883-890, July 2022, https://doi.org/10.33395/sinkron.v7i3.11528

[14] NoSQL Database: Cassandra is a Better Option to Handle Big Data," International Journal of Science and Research (IJSR), vol. 5, no. 1, pp. 24-26, January 2016, https://doi.org/10.21275/v5i1.NOV152557

[15] V. Tiwari, "Oracle Database Backup Testing," International Journal of Trend in Scientific Research and Development, vol. 2, no. 3, pp. 2043-2044, April 2018, https://doi.org/10.31142/ijtsrd11572

[16] A. Boicea and F. R. a. L. I. Agapin, "MongoDB vs Oracle -- Database Comparison," 2012 Third International Conference on Emerging Intelligent Data and Web Technologies, September 2012, https://doi.org/10.1109/EIDWT.2012.32

[17] H. Kim and H. Y. Y. a. Y. Son, "An Efficient Database Backup and Recovery Scheme using Write-Ahead Logging," IEEE Xplore, 2020. https://ieeexplore.ieee.org/document/9284224 (accessed May 11, 2022).

[18] C. Sauer and G. G. a. T. Härder, "Instant restore after a media failure," arXiv (Cornell University), January 2017, https://doi.org/10.1007/978-3-319-66917-5_21

[19] A. Magalhaes and J. M. M. a. A. Brayner, "Main Memory Database Recovery: A Survey," ACM Computing Surveys, vol. 54, no. 2, pp. 1-36, March 2021, https://doi.org/10.1145/3442197

[20] L. K. a. M. Krstić, "Testing the performance of NoSQL databases via teh database benchmark tool," Vojnotehnicki glasnik, vol. 66, no. 3, pp. 614-639, 2018. https://doi.org/10.5937/vojtehg66-15928