CSE 512: Distributed Database Systems

Project: Distributed Database System for a Smart Building

Part – 2

Horizontal & Vertical Fragmentation:

In PostgreSQL, fragmentation can be simulated using table partitioning. Table partitioning allows the division of a large table into smaller, more manageable pieces called partitions. Each partition represents a subset of the data based on a specific criterion, such as a range of values.

MongoDB, being a NoSQL database, does not natively support the same type of fragmentation as in traditional relational databases. MongoDB's document-oriented nature allows for flexible and dynamic schema designs. Thus, we use PostgreSQL's partitioning capabilities to simulate fragmentation techniques.

Horizontal fragmentation involves dividing a table into subsets of rows based on specific criteria. This is particularly useful for distributing data across different locations or nodes based on usage patterns or geographical considerations.

1) Access Logs Table - Timestamp:

- **Criteria:** The Access Logs table is horizontally fragmented based on the timestamp of the access events.
- **Implementation:** The table is split into three parts based on the year of the access events: 2021, 2022, and 2023.
- **Reasoning:**
  1. **Access Patterns:** Access Logs often exhibit time-based access patterns, where queries are frequently performed on access events within a specific time range.
  2. **Query Optimization:** Partitioning based on the timestamp allows for more efficient retrieval of logs within a particular time, optimizing query performance.
  3. **Data Archiving:** Older access logs can be easily archived or removed by dropping partitions that are no longer needed, helping in data management.

2) Users Table - Role:

- **Criteria:** The Users table is horizontally fragmented based on the role of the user.

- **Implementation:** The table is split into three parts based on user roles: Admin, Manager, and Employee.
- **Reasoning:**
    1. **Role-Based Access Control:** Many systems implement role-based access control (RBAC), where users with different roles have varying levels of access to resources.
    2. **Role-Specific Queries:** Partitioning the Users table based on roles facilitates more efficient queries for specific roles, especially in scenarios where role-related information is frequently queried together.
    3. **Security and Compliance:** In systems with sensitive data, partitioning by role can aid in implementing security and compliance measures.

Vertical fragmentation optimizes storage and retrieval by grouping columns that are frequently accessed together and separating them from less frequently accessed columns.

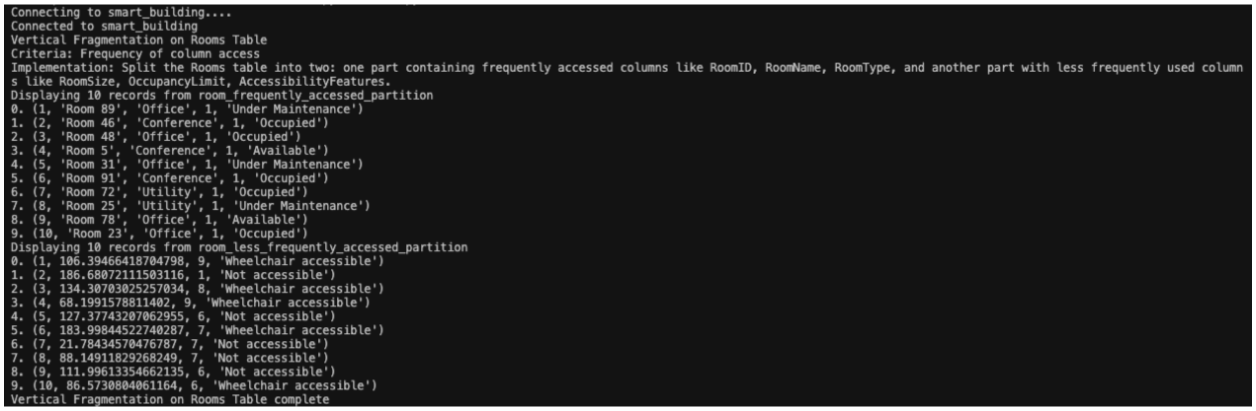
This strategy is beneficial when there is a significant difference in the access frequency of different columns, allowing for more efficient use of storage resources and potentially improving query performance.

1) Rooms Table - Frequency of Column Access:

- **Criteria:** The Rooms table is vertically fragmented based on the frequency of column access.
- **Implementation:** The table is split into two parts. One part contains frequently accessed columns (RoomID, RoomName, RoomType), and another part contains less frequently used columns (RoomSize, OccupancyLimit, AccessibilityFeatures).
- **Reasoning:**
    1. **Column Access Patterns:** Certain columns in the Rooms table may be accessed more frequently than others based on application requirements.
    2. **Query Performance:** Grouping frequently accessed columns together in one partition enhances query performance for common access patterns.
    3. **Storage Efficiency:** Less frequently accessed columns can be stored in a separate partition, optimizing storage space, and potentially improving cache utilization.

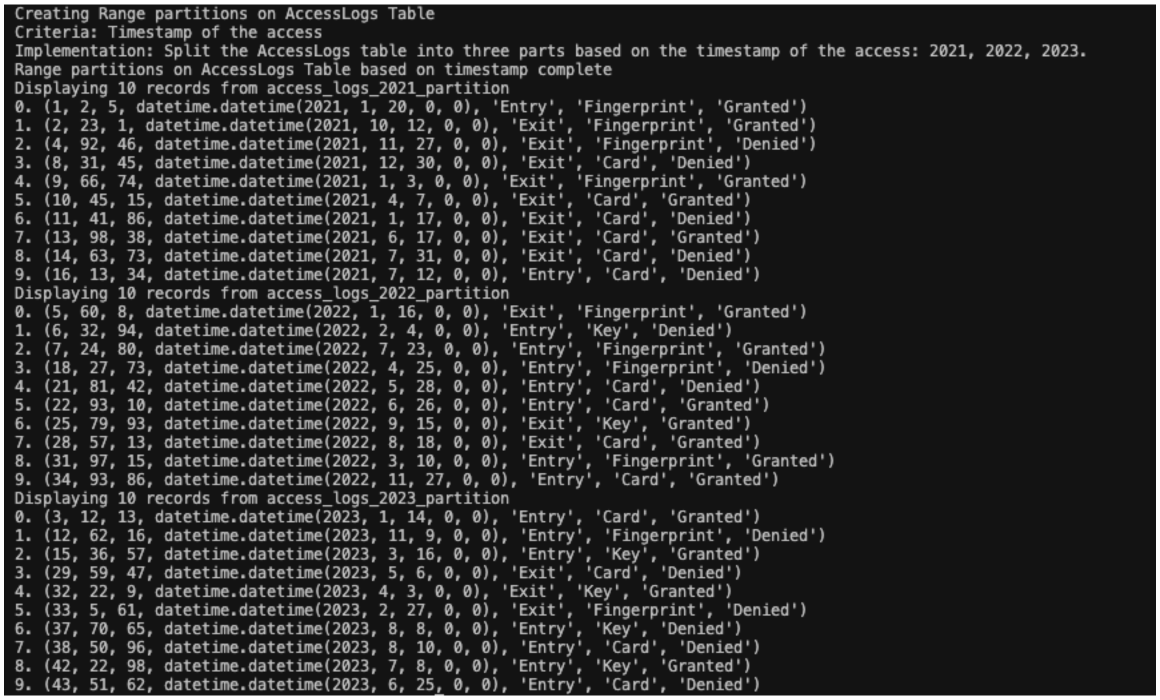
Proof of Fragmentation (Partitioning):

Vertical

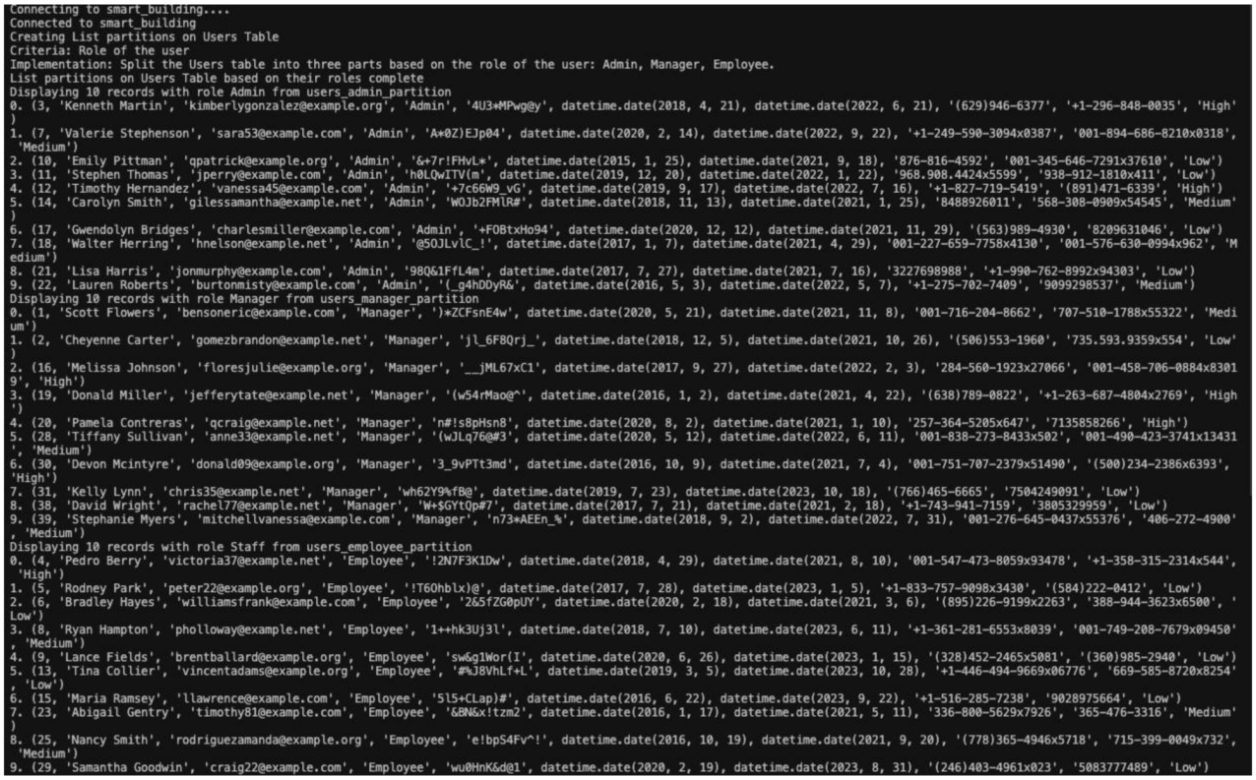


```
Connecting to smart_building....
Connected to smart_building
Vertical Fragmentation on Rooms Table
Criteria: Frequency of column access
Implementation: Split the Rooms table into two: one part containing frequently accessed columns like RoomID, RoomName, RoomType, and another part with less frequently used column
s like RoomSize, OccupancyLimit, AccessibilityFeatures.
Displaying 10 records from room_frequently_accessed_partition
0. (1, 'Room 89', 'Office', 1, 'Under Maintenance')
1. (2, 'Room 46', 'Conference', 1, 'Occupied')
2. (3, 'Room 48', 'Office', 1, 'Occupied')
3. (4, 'Room 5', 'Conference', 1, 'Available')
4. (5, 'Room 31', 'Office', 1, 'Under Maintenance')
5. (6, 'Room 91', 'Conference', 1, 'Occupied')
6. (7, 'Room 72', 'Utility', 1, 'Occupied')
7. (8, 'Room 25', 'Utility', 1, 'Under Maintenance')
8. (9, 'Room 78', 'Office', 1, 'Available')
9. (10, 'Room 23', 'Office', 1, 'Occupied')
Displaying 10 records from room_less_frequently_accessed_partition
0. (1, 106.39466418704798, 9, 'Wheelchair accessible')
1. (2, 186.68072111503116, 1, 'Not accessible')
2. (3, 134.30703025257034, 8, 'Wheelchair accessible')
3. (4, 68.1991578811402, 9, 'Wheelchair accessible')
4. (5, 127.37743207062955, 6, 'Not accessible')
5. (6, 183.99844522740287, 7, 'Wheelchair accessible')
6. (7, 21.78434570476787, 7, 'Not accessible')
7. (8, 88.14911829268249, 7, 'Not accessible')
8. (9, 111.99613354662135, 6, 'Not accessible')
9. (10, 86.5730804061164, 6, 'Wheelchair accessible')
Vertical Fragmentation on Rooms Table complete
```

Horizontal



```
Creating Range partitions on AccessLogs Table
Criteria: Timestamp of the access
Implementation: Split the AccessLogs table into three parts based on the timestamp of the access: 2021, 2022, 2023.
Range partitions on AccessLogs Table based on timestamp complete
Displaying 10 records from access_logs_2021_partition
0. (1, 2, 5, datetime.datetime(2021, 1, 20, 0, 0), 'Entry', 'Fingerprint', 'Granted')
1. (2, 23, 1, datetime.datetime(2021, 10, 12, 0, 0), 'Exit', 'Fingerprint', 'Granted')
2. (4, 92, 46, datetime.datetime(2021, 11, 27, 0, 0), 'Exit', 'Fingerprint', 'Denied')
3. (8, 31, 45, datetime.datetime(2021, 12, 30, 0, 0), 'Exit', 'Card', 'Denied')
4. (9, 66, 74, datetime.datetime(2021, 1, 3, 0, 0), 'Exit', 'Fingerprint', 'Granted')
5. (10, 45, 15, datetime.datetime(2021, 4, 7, 0, 0), 'Exit', 'Card', 'Granted')
6. (11, 41, 86, datetime.datetime(2021, 1, 17, 0, 0), 'Exit', 'Card', 'Denied')
7. (13, 98, 38, datetime.datetime(2021, 6, 17, 0, 0), 'Exit', 'Card', 'Granted')
8. (14, 63, 73, datetime.datetime(2021, 7, 31, 0, 0), 'Exit', 'Card', 'Denied')
9. (16, 13, 34, datetime.datetime(2021, 7, 12, 0, 0), 'Entry', 'Card', 'Denied')
Displaying 10 records from access_logs_2022_partition
0. (5, 60, 8, datetime.datetime(2022, 1, 16, 0, 0), 'Exit', 'Fingerprint', 'Granted')
1. (6, 32, 94, datetime.datetime(2022, 2, 4, 0, 0), 'Entry', 'Key', 'Denied')
2. (7, 24, 80, datetime.datetime(2022, 7, 23, 0, 0), 'Entry', 'Fingerprint', 'Granted')
3. (18, 27, 73, datetime.datetime(2022, 4, 25, 0, 0), 'Entry', 'Fingerprint', 'Denied')
4. (21, 81, 42, datetime.datetime(2022, 5, 28, 0, 0), 'Entry', 'Card', 'Denied')
5. (22, 93, 10, datetime.datetime(2022, 6, 26, 0, 0), 'Entry', 'Card', 'Granted')
6. (25, 79, 93, datetime.datetime(2022, 9, 15, 0, 0), 'Exit', 'Key', 'Granted')
7. (28, 57, 13, datetime.datetime(2022, 8, 18, 0, 0), 'Exit', 'Card', 'Granted')
8. (31, 97, 15, datetime.datetime(2022, 3, 10, 0, 0), 'Entry', 'Fingerprint', 'Granted')
9. (34, 93, 86, datetime.datetime(2022, 11, 27, 0, 0), 'Entry', 'Card', 'Granted')
Displaying 10 records from access_logs_2023_partition
0. (3, 12, 13, datetime.datetime(2023, 1, 14, 0, 0), 'Entry', 'Card', 'Granted')
1. (12, 62, 16, datetime.datetime(2023, 11, 9, 0, 0), 'Entry', 'Fingerprint', 'Denied')
2. (15, 36, 57, datetime.datetime(2023, 3, 16, 0, 0), 'Entry', 'Key', 'Granted')
3. (29, 59, 47, datetime.datetime(2023, 5, 6, 0, 0), 'Exit', 'Card', 'Denied')
4. (32, 22, 9, datetime.datetime(2023, 4, 3, 0, 0), 'Exit', 'Key', 'Granted')
5. (33, 5, 61, datetime.datetime(2023, 2, 27, 0, 0), 'Exit', 'Fingerprint', 'Denied')
6. (37, 70, 65, datetime.datetime(2023, 8, 8, 0, 0), 'Entry', 'Key', 'Denied')
7. (38, 50, 96, datetime.datetime(2023, 8, 10, 0, 0), 'Entry', 'Card', 'Denied')
8. (42, 22, 98, datetime.datetime(2023, 7, 8, 0, 0), 'Entry', 'Key', 'Granted')
9. (43, 51, 62, datetime.datetime(2023, 6, 25, 0, 0), 'Entry', 'Card', 'Denied')
```



```
Connecting to smart_building....
Connected to smart_building
Creating List partitions on Users Table
Criteria: Role of the user
Implementation: Split the Users table into three parts based on the role of the user: Admin, Manager, Employee.
List partitions on Users Table based on their roles complete
Displaying 10 records with role Admin from users_admin_partition
0. (3, 'Kenneth Martin', 'kimberlygonzalez@example.org', 'Admin', '4U3*MPwg@y', datetime.date(2018, 4, 21), datetime.date(2022, 6, 21), '(629)946-6377', '+1-296-848-0035', 'High'
)
1. (7, 'Valerie Stephenson', 'sara53@example.com', 'Admin', 'A*0Z)EJp04', datetime.date(2020, 2, 14), datetime.date(2022, 9, 22), '+1-249-590-3094x0387', '001-894-686-8210x0318',
 'Medium')
2. (10, 'Emily Pittman', 'qpatrick@example.org', 'Admin', '&+7r!FHvL*', datetime.date(2015, 1, 25), datetime.date(2021, 9, 18), '876-816-4592', '001-345-646-7291x37610', 'Low')
3. (11, 'Stephen Thomas', 'jperry@example.com', 'Admin', 'h0LQwITV(m', datetime.date(2019, 12, 20), datetime.date(2022, 1, 22), '968.908.4424x5599', '938-912-1810x411', 'Low')
4. (12, 'Timothy Hernandez', 'vanessa45@example.com', 'Admin', '+7c66W9_vG', datetime.date(2019, 9, 17), datetime.date(2022, 7, 16), '+1-827-719-5419', '(891)471-6339', 'High')
5. (14, 'Carolyn Smith', 'gilessamantha@example.net', 'Admin', 'WOJb2FMlR#', datetime.date(2018, 11, 13), datetime.date(2021, 1, 25), '8488926011', '568-308-0909x54545', 'Medium'
)
6. (17, 'Gwendolyn Bridges', 'charlesmiller@example.com', 'Admin', '+FOBtxHo94', datetime.date(2020, 12, 12), datetime.date(2021, 11, 29), '(563)989-4930', '8209631046', 'Low')
7. (18, 'Walter Herring', 'hnelson@example.net', 'Admin', '@5OJLvlC_!', datetime.date(2017, 1, 7), datetime.date(2021, 4, 29), '001-227-659-7758x4130', '001-576-630-0994x962', 'M
edium')
8. (21, 'Lisa Harris', 'jonmurphy@example.com', 'Admin', '98Q&1FfL4m', datetime.date(2017, 7, 27), datetime.date(2021, 7, 16), '3227698988', '+1-990-762-8992x94303', 'Low')
9. (22, 'Lauren Roberts', 'burtonmisty@example.com', 'Admin', '(_g4hDDyR&', datetime.date(2016, 5, 3), datetime.date(2022, 5, 7), '+1-275-702-7409', '9099298537', 'Medium')
Displaying 10 records with role Manager from users_manager_partition
0. (1, 'Scott Flowers', 'bensoneric@example.com', 'Manager', ')*ZCFsnE4w', datetime.date(2020, 5, 21), datetime.date(2021, 11, 8), '001-716-204-8662', '707-510-1788x55322', 'Medi
um')
1. (2, 'Cheyenne Carter', 'gomezbrandon@example.net', 'Manager', 'jl_6F8Qrj_', datetime.date(2018, 12, 5), datetime.date(2021, 10, 26), '(506)553-1960', '735.593.9359x554', 'Low'
)
2. (16, 'Melissa Johnson', 'floresjulie@example.org', 'Manager', '__jML67xC1', datetime.date(2017, 9, 27), datetime.date(2022, 2, 3), '284-560-1923x27066', '001-458-706-0884x8301
9', 'High')
3. (19, 'Donald Miller', 'jefferytate@example.net', 'Manager', '(w54rMao@^', datetime.date(2016, 1, 2), datetime.date(2021, 4, 22), '(638)789-0822', '+1-263-687-4804x2769', 'High
')
4. (20, 'Pamela Contreras', 'qcraig@example.net', 'Manager', 'n#!s8pHsn8', datetime.date(2020, 8, 2), datetime.date(2021, 1, 10), '257-364-5205x647', '7135858266', 'High')
5. (28, 'Tiffany Sullivan', 'anne33@example.net', 'Manager', '(wJLq76@#3', datetime.date(2020, 5, 12), datetime.date(2022, 6, 11), '001-838-273-8433x502', '001-490-423-3741x13431
', 'Medium')
6. (30, 'Devon Mcintyre', 'donald09@example.org', 'Manager', '3_9vPTt3md', datetime.date(2016, 10, 9), datetime.date(2021, 7, 4), '001-751-707-2379x51490', '(500)234-2386x6393',
'High')
7. (31, 'Kelly Lynn', 'chris35@example.net', 'Manager', 'wh62Y9%fB@', datetime.date(2019, 7, 23), datetime.date(2023, 10, 18), '(766)465-6665', '7504249091', 'Low')
8. (38, 'David Wright', 'rachel77@example.net', 'Manager', 'W+$GYtQp#7', datetime.date(2017, 7, 21), datetime.date(2021, 2, 18), '+1-743-941-7159', '3805329959', 'Low')
9. (39, 'Stephanie Myers', 'mitchellvanessa@example.com', 'Manager', 'n73*AEEn_%', datetime.date(2018, 9, 2), datetime.date(2022, 7, 31), '001-276-645-0437x55376', '406-272-4900'
, 'Medium')
Displaying 10 records with role Staff from users_employee_partition
0. (4, 'Pedro Berry', 'victoria37@example.net', 'Employee', '!2N7F3K1Dw', datetime.date(2018, 4, 29), datetime.date(2021, 8, 10), '001-547-473-8059x93478', '+1-358-315-2314x544',
 'High')
1. (5, 'Rodney Park', 'peter22@example.org', 'Employee', '!T6Ohblx)@', datetime.date(2017, 7, 28), datetime.date(2023, 1, 5), '+1-833-757-9098x3430', '(584)222-0412', 'Low')
2. (6, 'Bradley Hayes', 'williamsfrank@example.com', 'Employee', '2&5fZG0pUY', datetime.date(2020, 2, 18), datetime.date(2021, 3, 6), '(895)226-9199x2263', '388-944-3623x6500', '
Low')
3. (8, 'Ryan Hampton', 'pholloway@example.net', 'Employee', '1++hk3Uj3l', datetime.date(2018, 7, 10), datetime.date(2023, 6, 11), '+1-361-281-6553x8039', '001-749-208-7679x09450'
, 'Medium')
4. (9, 'Lance Fields', 'brentballard@example.org', 'Employee', 'sw&g1Wor(I', datetime.date(2020, 6, 26), datetime.date(2023, 1, 15), '(328)452-2465x5081', '(360)985-2940', 'Low')
5. (13, 'Tina Collier', 'vincentadams@example.org', 'Employee', '#%J8VhLf+L', datetime.date(2019, 3, 5), datetime.date(2023, 10, 28), '+1-446-494-9669x06776', '669-585-8720x8254'
, 'Low')
6. (15, 'Maria Ramsey', 'llawrence@example.com', 'Employee', '5l5+CLap)#', datetime.date(2016, 6, 22), datetime.date(2023, 9, 22), '+1-516-285-7238', '9028975664', 'Low')
7. (23, 'Abigail Gentry', 'timothy81@example.com', 'Employee', '&BN&x!tzm2', datetime.date(2016, 1, 17), datetime.date(2021, 5, 11), '336-800-5629x7926', '365-476-3316', 'Medium'
)
8. (25, 'Nancy Smith', 'rodriguezamanda@example.org', 'Employee', 'e!bpS4Fv^!', datetime.date(2016, 10, 19), datetime.date(2021, 9, 20), '(778)365-4946x5718', '715-399-0049x732',
 'Medium')
9. (29, 'Samantha Goodwin', 'craig22@example.com', 'Employee', 'wu0HnK&d@1', datetime.date(2020, 2, 19), datetime.date(2023, 8, 31), '(246)403-4961x023', '5083777489', 'Low')
```

Replication

MongoDB master-slave replication is a fundamental feature designed to enhance data availability, fault tolerance, and read scalability in MongoDB deployments. In this replication model, there are two types of nodes: the master and the slave (or secondary).

Master Node:

The master node is the primary node that manages write operations. All write operations, such as inserts, updates, and deletes, are initially performed on the master node. It is responsible for processing these write operations and then replicating the changes to one or more slave nodes.

Slave Nodes:

Slave nodes, also known as secondary nodes, replicate data from the master node. They serve as read replicas and can be used for querying data. However, they do not handle write operations directly. Instead, they apply changes received from the master node to keep their data synchronized.

Replication Process:

1. **Oplog (Operation Log):** MongoDB uses an operation log, known as the oplog, to record all write operations that occur on the master node. The oplog is a capped collection that stores a chronological record of the write operations.

2. **Replication Flow:**

    - The master node processes a write operation and records it in its oplog.

    - The slave nodes continuously pull the oplog from the master and apply the recorded operations to replicate the changes locally.

    - This ensures that the data on the slave nodes stays synchronized with the master.

Use Cases and Benefits:

1. **High Availability:** If the master node fails, one of the slave nodes can be promoted to become the new master, ensuring continuous availability of the database.

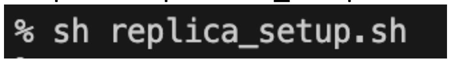
2. **Read Scalability:** Since slave nodes can handle read operations, distributing read traffic across multiple nodes can improve overall system performance.

3. **Backup:** Slave nodes can be used for backup purposes, providing a consistent snapshot of the data.

4. **Geographic Distribution:** Replication can be used to create copies of data in different geographic locations, improving data locality and reducing latency for users in different regions.

The setup for master-slave replication in mongodb is as follows:

1. We start three mongod processes, each on a different port and with a different db path, but all using the same replica set name "Smart_Building".
   Note: The command for this setup is in replication_setup.sh.

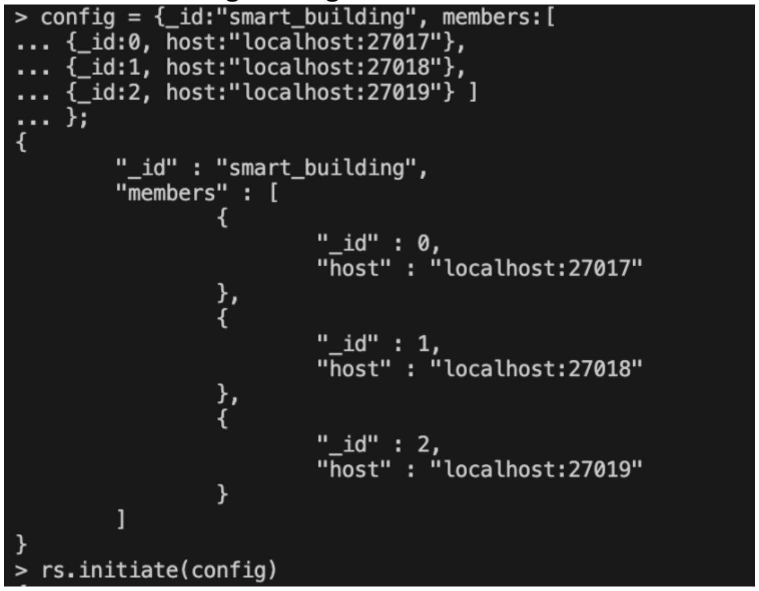


```
% sh replica_setup.sh
```

2. To Initialize the replica set, begin the mongo server by opening a new terminal and running mongo.

```
% mongo
```

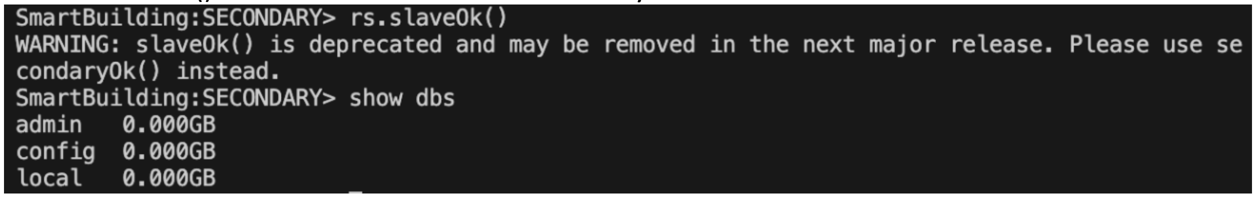
3. Now, run the initiate command to get things started:



```
> config = {_id:"smart_building", members:[
... {_id:0, host:"localhost:27017"},
... {_id:1, host:"localhost:27018"},
... {_id:2, host:"localhost:27019"} ]
... };
{
        "_id" : "smart_building",
        "members" : [
                {
                        "_id" : 0,
                        "host" : "localhost:27017"
                },
                {
                        "_id" : 1,
                        "host" : "localhost:27018"
                },
                {
                        "_id" : 2,
                        "host" : "localhost:27019"
                }
        ]
}
> rs.initiate(config)
```

   The three mongod servers we started earlier will now coordinate and come online as a replica set.

4. Run rs.slaveOk() method in each of the secondary servers.

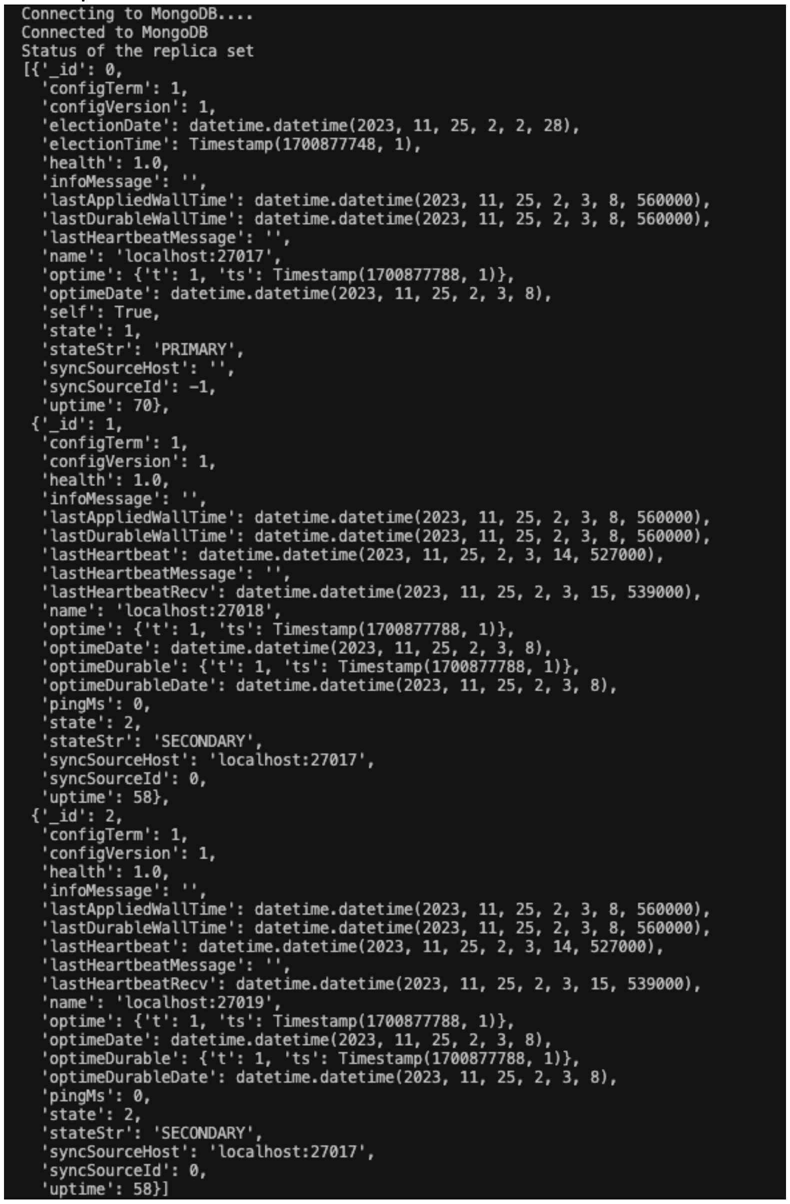


```
SmartBuilding:SECONDARY> rs.slaveOk()
WARNING: slaveOk() is deprecated and may be removed in the next major release. Please use se
condaryOk() instead.
SmartBuilding:SECONDARY> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

The replica set is ready for usage in the Master-Slave mode of replication.

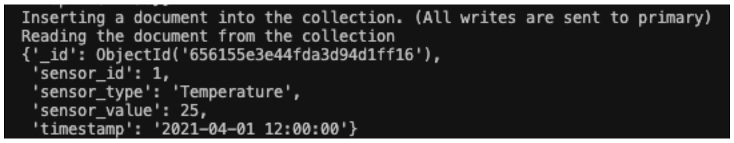
Failover Scenario Screenshots

1. Initial Status of the Replica Set:



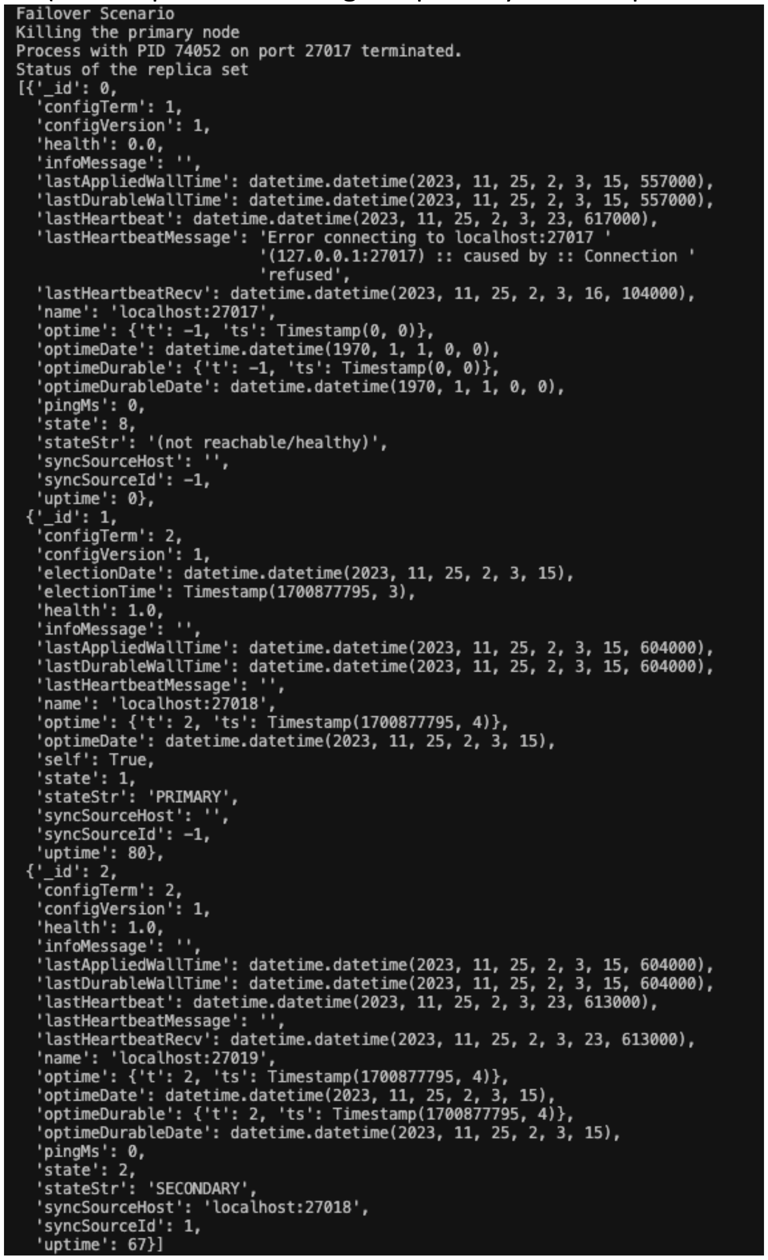
```
Connecting to MongoDB....
Connected to MongoDB
Status of the replica set
[{'_id': 0,
  'configTerm': 1,
  'configVersion': 1,
  'electionDate': datetime.datetime(2023, 11, 25, 2, 2, 28),
  'electionTime': Timestamp(1700877748, 1),
  'health': 1.0,
  'infoMessage': '',
  'lastAppliedWallTime': datetime.datetime(2023, 11, 25, 2, 3, 8, 560000),
  'lastDurableWallTime': datetime.datetime(2023, 11, 25, 2, 3, 8, 560000),
  'lastHeartbeatMessage': '',
  'name': 'localhost:27017',
  'optime': {'t': 1, 'ts': Timestamp(1700877788, 1)},
  'optimeDate': datetime.datetime(2023, 11, 25, 2, 3, 8),
  'self': True,
  'state': 1,
  'stateStr': 'PRIMARY',
  'syncSourceHost': '',
  'syncSourceId': -1,
  'uptime': 70},
 {'_id': 1,
  'configTerm': 1,
  'configVersion': 1,
  'health': 1.0,
  'infoMessage': '',
  'lastAppliedWallTime': datetime.datetime(2023, 11, 25, 2, 3, 8, 560000),
  'lastDurableWallTime': datetime.datetime(2023, 11, 25, 2, 3, 8, 560000),
  'lastHeartbeat': datetime.datetime(2023, 11, 25, 2, 3, 14, 527000),
  'lastHeartbeatMessage': '',
  'lastHeartbeatRecv': datetime.datetime(2023, 11, 25, 2, 3, 15, 539000),
  'name': 'localhost:27018',
  'optime': {'t': 1, 'ts': Timestamp(1700877788, 1)},
  'optimeDate': datetime.datetime(2023, 11, 25, 2, 3, 8),
  'optimeDurable': {'t': 1, 'ts': Timestamp(1700877788, 1)},
  'optimeDurableDate': datetime.datetime(2023, 11, 25, 2, 3, 8),
  'pingMs': 0,
  'state': 2,
  'stateStr': 'SECONDARY',
  'syncSourceHost': 'localhost:27017',
  'syncSourceId': 0,
  'uptime': 58},
 {'_id': 2,
  'configTerm': 1,
  'configVersion': 1,
  'health': 1.0,
  'infoMessage': '',
  'lastAppliedWallTime': datetime.datetime(2023, 11, 25, 2, 3, 8, 560000),
  'lastDurableWallTime': datetime.datetime(2023, 11, 25, 2, 3, 8, 560000),
  'lastHeartbeat': datetime.datetime(2023, 11, 25, 2, 3, 14, 527000),
  'lastHeartbeatMessage': '',
  'lastHeartbeatRecv': datetime.datetime(2023, 11, 25, 2, 3, 15, 539000),
  'name': 'localhost:27019',
  'optime': {'t': 1, 'ts': Timestamp(1700877788, 1)},
  'optimeDate': datetime.datetime(2023, 11, 25, 2, 3, 8),
  'optimeDurable': {'t': 1, 'ts': Timestamp(1700877788, 1)},
  'optimeDurableDate': datetime.datetime(2023, 11, 25, 2, 3, 8),
  'pingMs': 0,
  'state': 2,
  'stateStr': 'SECONDARY',
  'syncSourceHost': 'localhost:27017',
  'syncSourceId': 0,
  'uptime': 58}]
```

2. Inserting document into collection



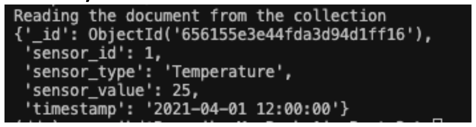
```
Inserting a document into the collection. (All writes are sent to primary)
Reading the document from the collection
{'_id': ObjectId('656155e3e44fda3d94d1ff16'),
 'sensor_id': 1,
 'sensor_type': 'Temperature',
 'sensor_value': 25,
 'timestamp': '2021-04-01 12:00:00'}
```

3. Failover Scenario: (Kill the process running the primary node at port 27017)



```
Failover Scenario
Killing the primary node
Process with PID 74052 on port 27017 terminated.
Status of the replica set
[{'_id': 0,
  'configTerm': 1,
  'configVersion': 1,
  'health': 0.0,
  'infoMessage': '',
  'lastAppliedWallTime': datetime.datetime(2023, 11, 25, 2, 3, 15, 557000),
  'lastDurableWallTime': datetime.datetime(2023, 11, 25, 2, 3, 15, 557000),
  'lastHeartbeat': datetime.datetime(2023, 11, 25, 2, 3, 23, 617000),
  'lastHeartbeatMessage': 'Error connecting to localhost:27017 '
                          '(127.0.0.1:27017) :: caused by :: Connection '
                          'refused',
  'lastHeartbeatRecv': datetime.datetime(2023, 11, 25, 2, 3, 16, 104000),
  'name': 'localhost:27017',
  'optime': {'t': -1, 'ts': Timestamp(0, 0)},
  'optimeDate': datetime.datetime(1970, 1, 1, 0, 0),
  'optimeDurable': {'t': -1, 'ts': Timestamp(0, 0)},
  'optimeDurableDate': datetime.datetime(1970, 1, 1, 0, 0),
  'pingMs': 0,
  'state': 8,
  'stateStr': '(not reachable/healthy)',
  'syncSourceHost': '',
  'syncSourceId': -1,
  'uptime': 0},
 {'_id': 1,
  'configTerm': 2,
  'configVersion': 1,
  'electionDate': datetime.datetime(2023, 11, 25, 2, 3, 15),
  'electionTime': Timestamp(1700877795, 3),
  'health': 1.0,
  'infoMessage': '',
  'lastAppliedWallTime': datetime.datetime(2023, 11, 25, 2, 3, 15, 604000),
  'lastDurableWallTime': datetime.datetime(2023, 11, 25, 2, 3, 15, 604000),
  'lastHeartbeatMessage': '',
  'name': 'localhost:27018',
  'optime': {'t': 2, 'ts': Timestamp(1700877795, 4)},
  'optimeDate': datetime.datetime(2023, 11, 25, 2, 3, 15),
  'self': True,
  'state': 1,
  'stateStr': 'PRIMARY',
  'syncSourceHost': '',
  'syncSourceId': -1,
  'uptime': 80},
 {'_id': 2,
  'configTerm': 2,
  'configVersion': 1,
  'health': 1.0,
  'infoMessage': '',
  'lastAppliedWallTime': datetime.datetime(2023, 11, 25, 2, 3, 15, 604000),
  'lastDurableWallTime': datetime.datetime(2023, 11, 25, 2, 3, 15, 604000),
  'lastHeartbeat': datetime.datetime(2023, 11, 25, 2, 3, 23, 613000),
  'lastHeartbeatMessage': '',
  'lastHeartbeatRecv': datetime.datetime(2023, 11, 25, 2, 3, 23, 613000),
  'name': 'localhost:27019',
  'optime': {'t': 2, 'ts': Timestamp(1700877795, 4)},
  'optimeDate': datetime.datetime(2023, 11, 25, 2, 3, 15),
  'optimeDurable': {'t': 2, 'ts': Timestamp(1700877795, 4)},
  'optimeDurableDate': datetime.datetime(2023, 11, 25, 2, 3, 15),
  'pingMs': 0,
  'state': 2,
  'stateStr': 'SECONDARY',
  'syncSourceHost': 'localhost:27018',
  'syncSourceId': 1,
  'uptime': 67}]
```

4. Reading documents after primary failure:



```
Reading the document from the collection
{'_id': ObjectId('656155e3e44fda3d94d1ff16'),
 'sensor_id': 1,
 'sensor_type': 'Temperature',
 'sensor_value': 25,
 'timestamp': '2021-04-01 12:00:00'}
```

Thus, the document is read even after the primary fails ensuring data availability during failure. By default, the nodes are configured to elect a new primary when the current primary fails. All subsequent writes will be sent to the new primary node in the replica set.

Compiled By:
Distributed Nerds
Ahraz Rizvi
Keshava Rajavaram
Poorvik Dharmendra
Sahara Abdi