TY B.Tech. (CSE) – II [2022-23]

5CS372: Advanced Database System Lab.

Assignment No.8

Name : Tanaya Mukund Bhide

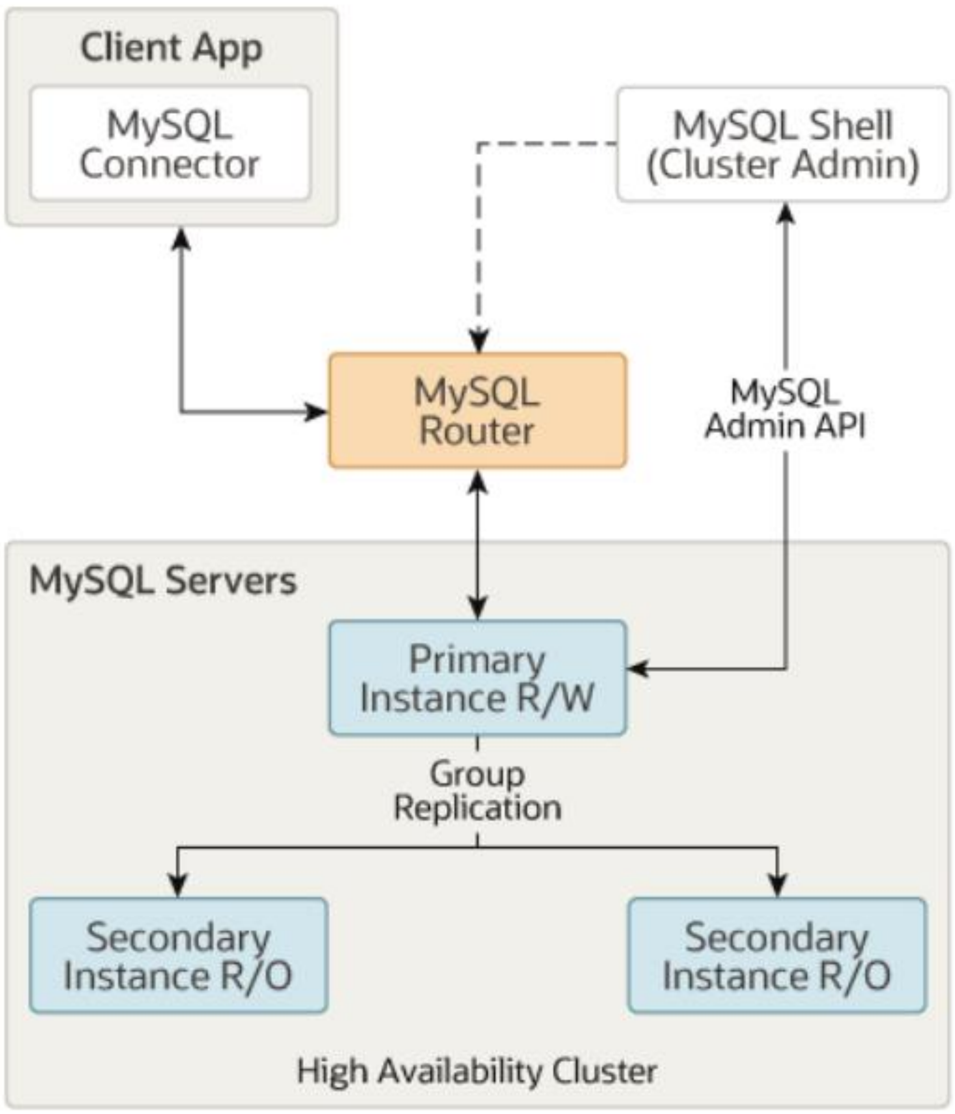
PRN : 2020BTECS00011

Batch : T5

Branch: T.Y CSE

Introduction: InnoDB Cluster is a high-availability solution provided by MySQL that allows you to easily configure and manage a cluster of MySQL instances. One of the features of InnoDB Cluster is data replication, which involves copying data from one MySQL instance to another in real-time. Replication is a key feature of InnoDB Cluster, as it helps to ensure data consistency and high availability in case of failures. In this report, we will discuss data replication using InnoDB Cluster.

Theory: InnoDB Cluster uses a master-slave replication model to replicate data from the master to one or more slave instances. The master instance is the primary source of data, and all updates and changes to the data are made on the master. The slave instances receive copies of the data from the master and apply them in real-time. This allows the slave instances to stay in sync with the master and provide high availability in case of failures.

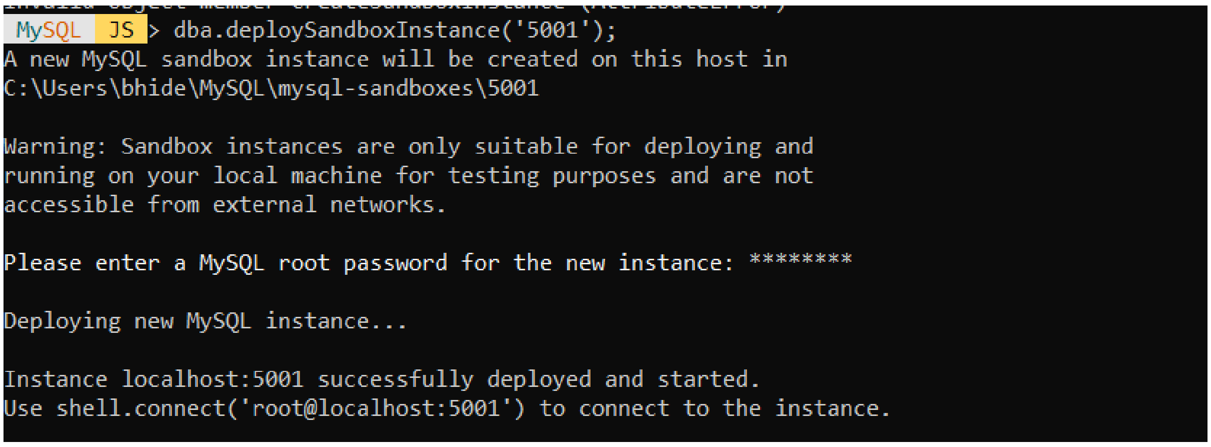

Client App
MySQL Connector
MySQL Shell (Cluster Admin)
MySQL Router
MySQL Admin API
MySQL Servers
Primary Instance R/W
Group Replication
Secondary Instance R/O
Secondary Instance R/O
High Availability Cluster

InnoDB Cluster uses Group Replication technology to manage the replication process. Group Replication is a distributed, fault-tolerant protocol that ensures that all instances in the cluster have the same data at all times. Each instance in the cluster has a role in the replication process, and the cluster works together to ensure that the data is consistent across all instances. InnoDB Cluster also provides features such as automatic failover, which allows the cluster to automatically promote a slave instance to become the new master in case of a failure. This ensures that the data remains available even if one or more instances fail. Overall, data replication using InnoDB Cluster provides a reliable and easy-tomanage solution for ensuring high availability and data consistency in a MySQL cluster.

Objective: The objective of data replication using InnoDB Cluster is to ensure high availability and data consistency in a MySQL cluster.

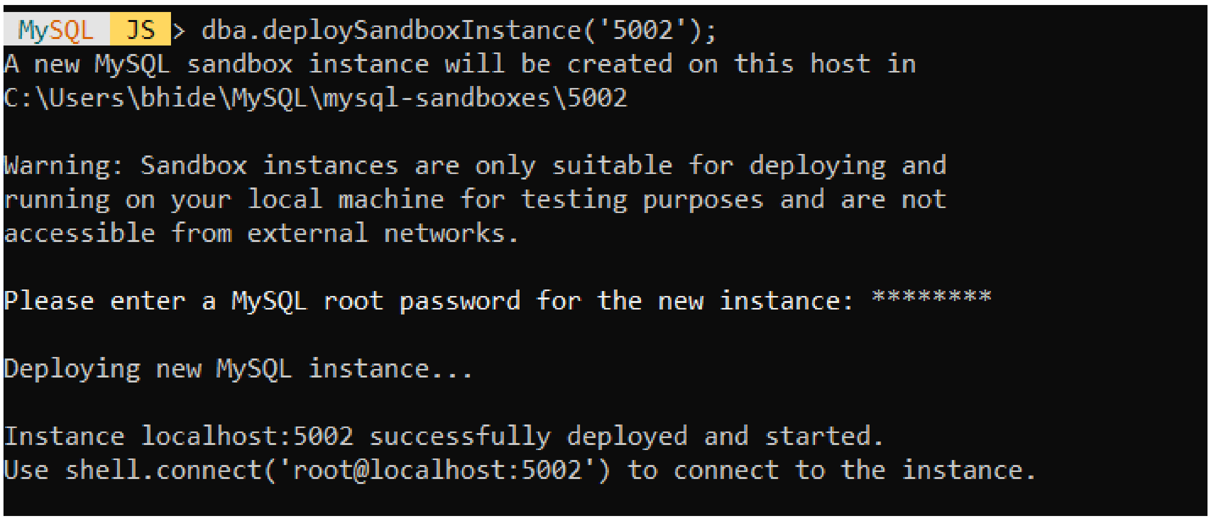
Creation of the cluster :

1. Call the dba.deploySandboxInstance method and pass in the port number 5001 as a parameter to create a new MySQL sandbox instance on port 5001.
2. Enter a MySQL root password for the new instance when prompted.
3. The method will start deploying the new MySQL instance on port 5001.
4. The new sandbox instance will be created in the directory C:\Users\bhide\MySQL\mysql-sandboxes\5001 on your local machine. This is where the MySQL server files for this instance will be stored.
5. Once the deployment is complete, you will see a message indicating that the new instance has been successfully deployed and started on localhost:5001.



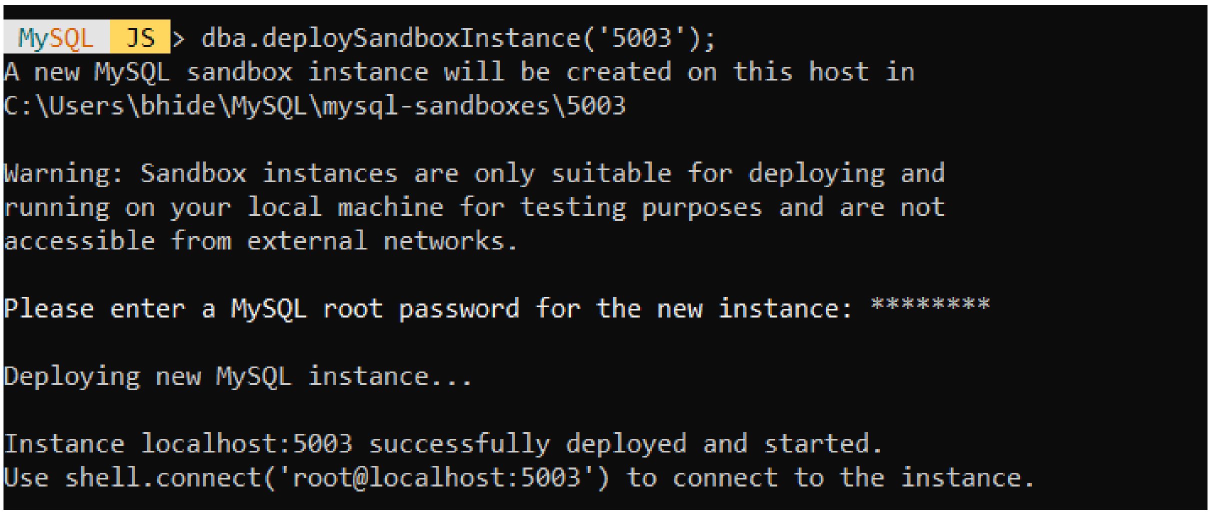
```
 MySQL  JS > dba.deploySandboxInstance('5001');
A new MySQL sandbox instance will be created on this host in
C:\Users\bhide\MySQL\mysql-sandboxes\5001

Warning: Sandbox instances are only suitable for deploying and
running on your local machine for testing purposes and are not
accessible from external networks.

Please enter a MySQL root password for the new instance: ********

Deploying new MySQL instance...

Instance localhost:5001 successfully deployed and started.
Use shell.connect('root@localhost:5001') to connect to the instance.
```

we repeat the above steps to create a second instance on port 5002, using the dba.deploySandboxInstance method with the parameter '5002'. This will create a new sandbox instance in the directory C:\Users\bhide\MySQL\mysql-sandboxes\5002 and start the instance on port 5002.



```
 MySQL  JS > dba.deploySandboxInstance('5002');
A new MySQL sandbox instance will be created on this host in
C:\Users\bhide\MySQL\mysql-sandboxes\5002

Warning: Sandbox instances are only suitable for deploying and
running on your local machine for testing purposes and are not
accessible from external networks.

Please enter a MySQL root password for the new instance: ********

Deploying new MySQL instance...

Instance localhost:5002 successfully deployed and started.
Use shell.connect('root@localhost:5002') to connect to the instance.
```

we repeat the above steps to create a second instance on port 5003, using the dba.deploySandboxInstance method with the parameter '5003'. This will create a new sandbox instance in the directory C:\Users\bhide\MySQL\mysql-sandboxes\5003 and start the instance on port 5003.



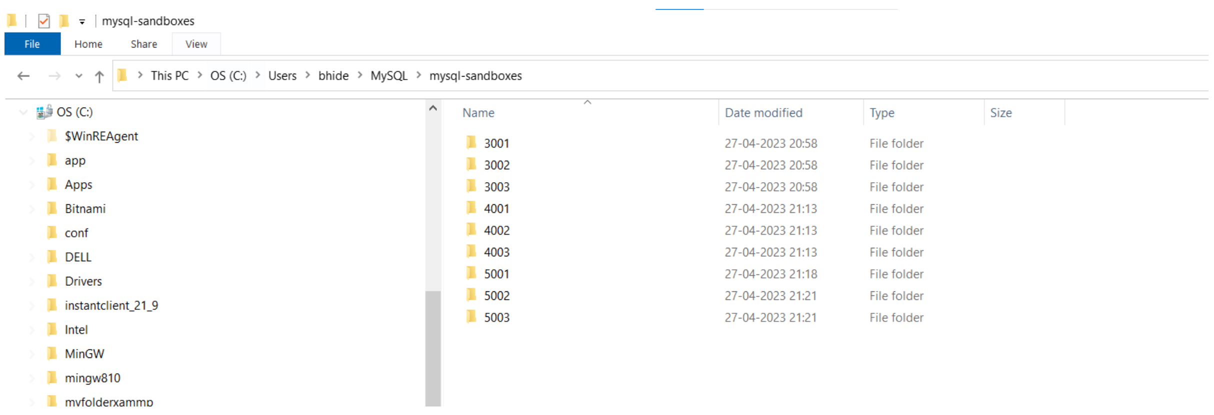
```
 MySQL  JS > dba.deploySandboxInstance('5003');
A new MySQL sandbox instance will be created on this host in
C:\Users\bhide\MySQL\mysql-sandboxes\5003

Warning: Sandbox instances are only suitable for deploying and
running on your local machine for testing purposes and are not
accessible from external networks.

Please enter a MySQL root password for the new instance: ********

Deploying new MySQL instance...

Instance localhost:5003 successfully deployed and started.
Use shell.connect('root@localhost:5003') to connect to the instance.
```

Three folders that are created in C:\Users\bhide\MySQL\mysql-sandboxes when you use the dba.deploySandboxInstance method to create a new MySQL sandbox instance:

5001: This folder contains the MySQL server files for the first MySQL sandbox instance that you created using the dba.deploySandboxInstance('5001') method. All the data and configuration files for this instance are stored in this folder.

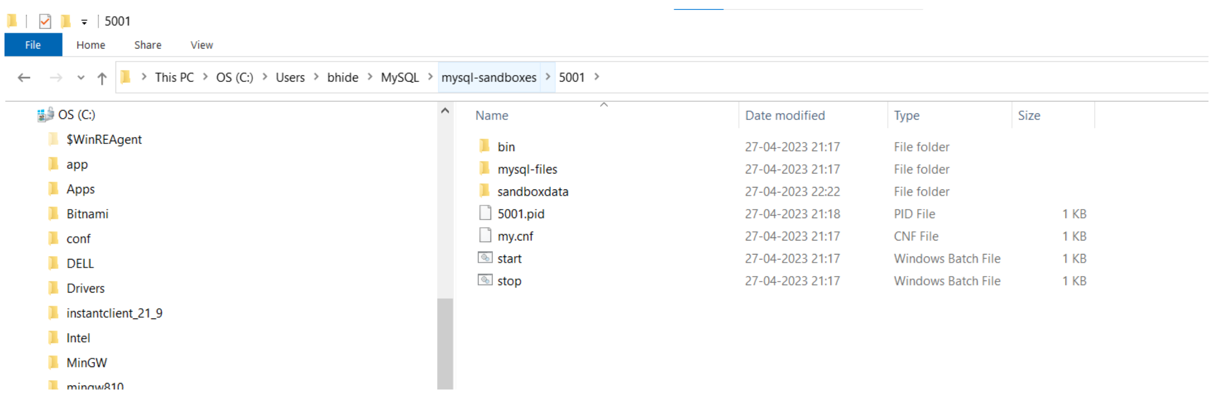
5002: This folder contains the MySQL server files for the second MySQL sandbox instance that you created using the dba.deploySandboxInstance('5002') method. Like the first folder, this folder also contains all the data and configuration files for this instance.

5003: This folder contains the MySQL server files for the second MySQL sandbox instance that you created using the dba.deploySandboxInstance('5003') method. Like the first folder, this folder also contains all the data and configuration files for this instance.



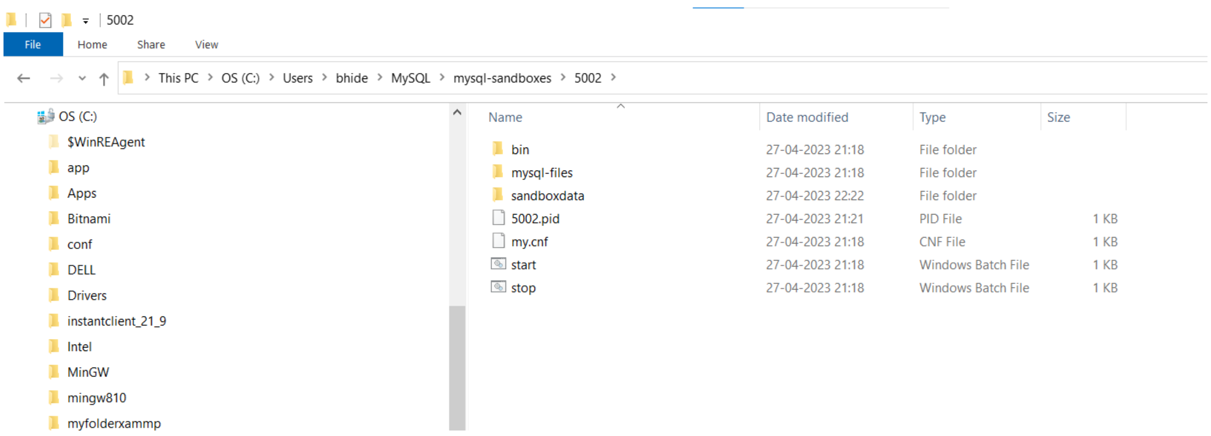
This PC > OS (C:) > Users > bhide > MySQL > mysql-sandboxes

Name	Date modified	Type	Size
3001	27-04-2023 20:58	File folder	
3002	27-04-2023 20:58	File folder	
3003	27-04-2023 20:58	File folder	
4001	27-04-2023 21:13	File folder	
4002	27-04-2023 21:13	File folder	
4003	27-04-2023 21:13	File folder	
5001	27-04-2023 21:18	File folder	
5002	27-04-2023 21:21	File folder	
5003	27-04-2023 21:21	File folder	



This PC > OS (C:) > Users > bhide > MySQL > mysql-sandboxes > 5001

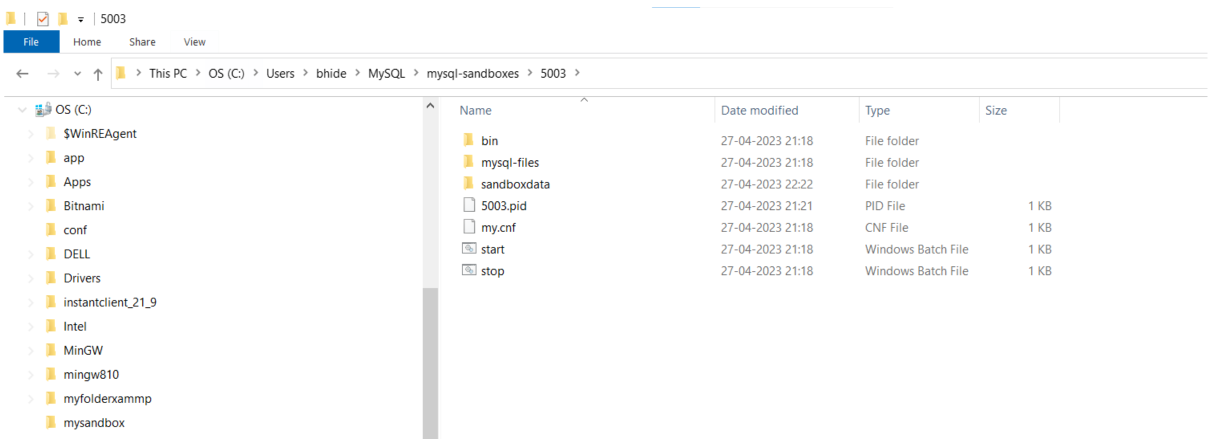
Name	Date modified	Type	Size
bin	27-04-2023 21:17	File folder	
mysql-files	27-04-2023 21:17	File folder	
sandboxdata	27-04-2023 22:22	File folder	
5001.pid	27-04-2023 21:18	PID File	1 KB
my.cnf	27-04-2023 21:17	CNF File	1 KB
start	27-04-2023 21:17	Windows Batch File	1 KB
stop	27-04-2023 21:17	Windows Batch File	1 KB



This PC > OS (C:) > Users > bhide > MySQL > mysql-sandboxes > 5002 >

OS (C:)
$WinREAgent
app
Apps
Bitnami
conf
DELL
Drivers
instantclient_21_9
Intel
MinGW
mingw810
myfolderxammp

Name	Date modified	Type	Size
bin	27-04-2023 21:18	File folder	
mysql-files	27-04-2023 21:18	File folder	
sandboxdata	27-04-2023 22:22	File folder	
5002.pid	27-04-2023 21:21	PID File	1 KB
my.cnf	27-04-2023 21:18	CNF File	1 KB
start	27-04-2023 21:18	Windows Batch File	1 KB
stop	27-04-2023 21:18	Windows Batch File	1 KB

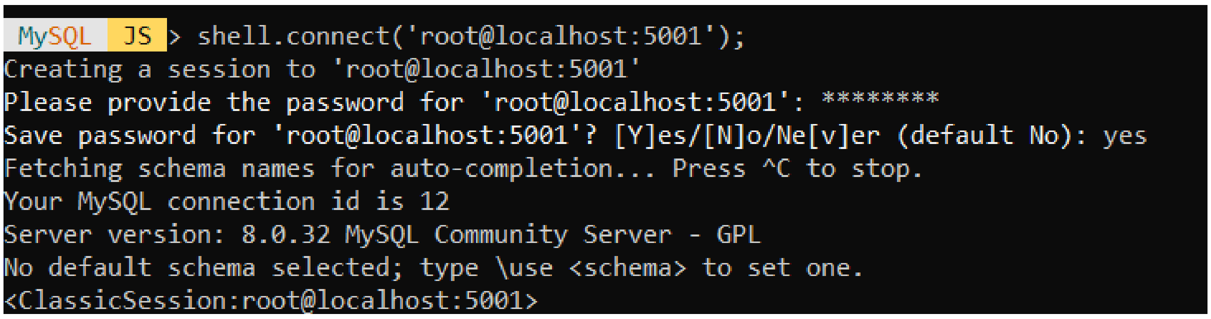


This PC > OS (C:) > Users > bhide > MySQL > mysql-sandboxes > 5003 >

OS (C:)
$WinREAgent
app
Apps
Bitnami
conf
DELL
Drivers
instantclient_21_9
Intel
MinGW
mingw810
myfolderxammp
mysandbox

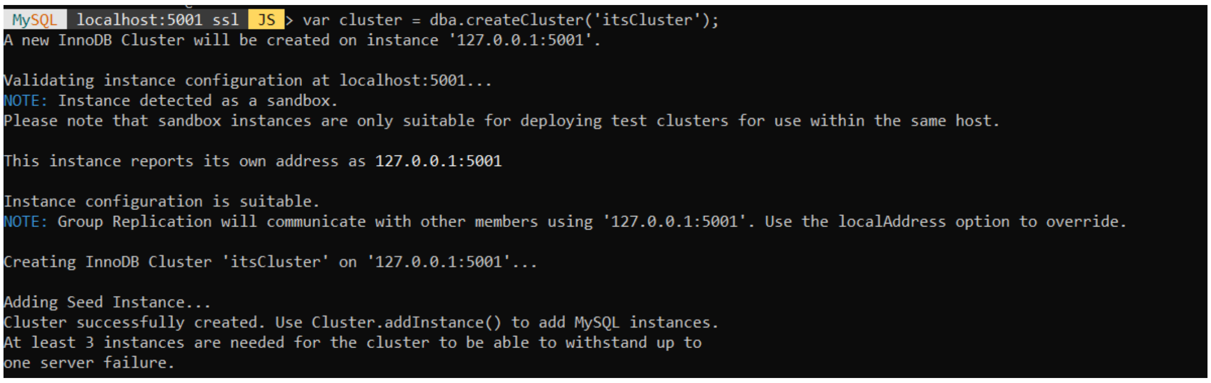
Name	Date modified	Type	Size
bin	27-04-2023 21:18	File folder	
mysql-files	27-04-2023 21:18	File folder	
sandboxdata	27-04-2023 22:22	File folder	
5003.pid	27-04-2023 21:21	PID File	1 KB
my.cnf	27-04-2023 21:18	CNF File	1 KB
start	27-04-2023 21:18	Windows Batch File	1 KB
stop	27-04-2023 21:18	Windows Batch File	1 KB

MySQL JS shell command shell.connect('root@localhost:5001');which connects to the MySQL sandbox instance running on port 5001 that you created using the dba.deploySandboxInstance method earlier.



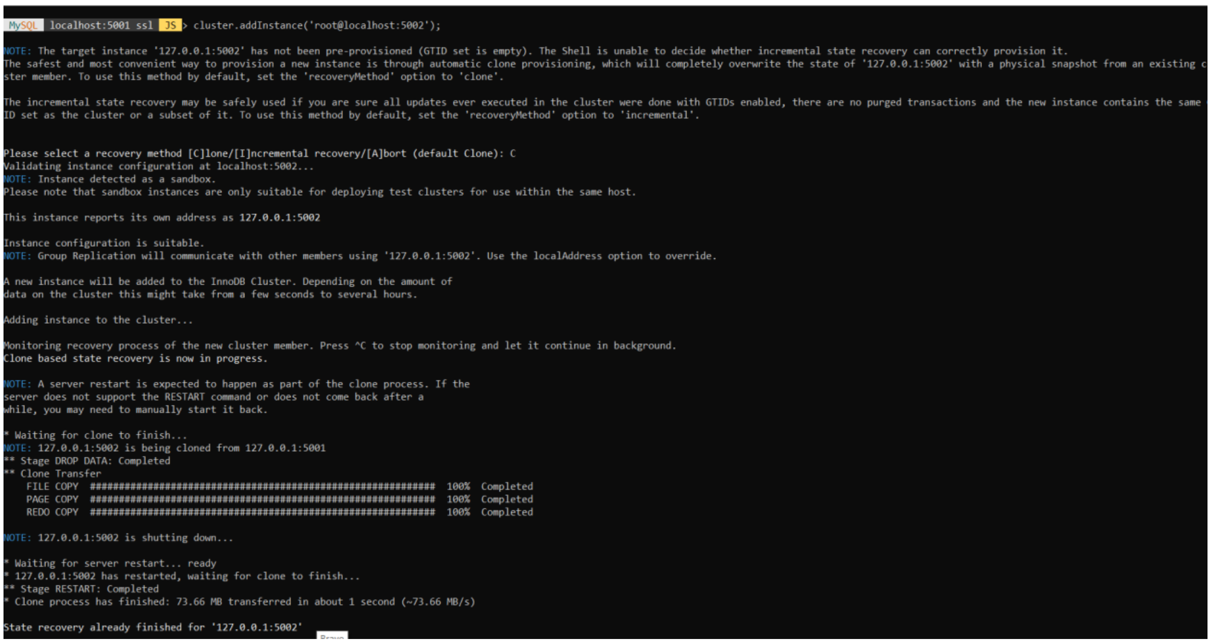
```
 MySQL  JS > shell.connect('root@localhost:5001');
Creating a session to 'root@localhost:5001'
Please provide the password for 'root@localhost:5001': ********
Save password for 'root@localhost:5001'? [Y]es/[N]o/Ne[v]er (default No): yes
Fetching schema names for auto-completion... Press ^C to stop.
Your MySQL connection id is 12
Server version: 8.0.32 MySQL Community Server - GPL
No default schema selected; type \use <schema> to set one.
<ClassicSession:root@localhost:5001>
```

Using the MySQL command, a new InnoDB cluster named 'itsCluster' is created on a MySQL instance running on localhost:5001. The instance configuration is validated, and a seed instance is added to the cluster. The message indicates that the cluster is successfully created and that you can use the Cluster.addInstance() method to add more MySQL instances to the cluster.



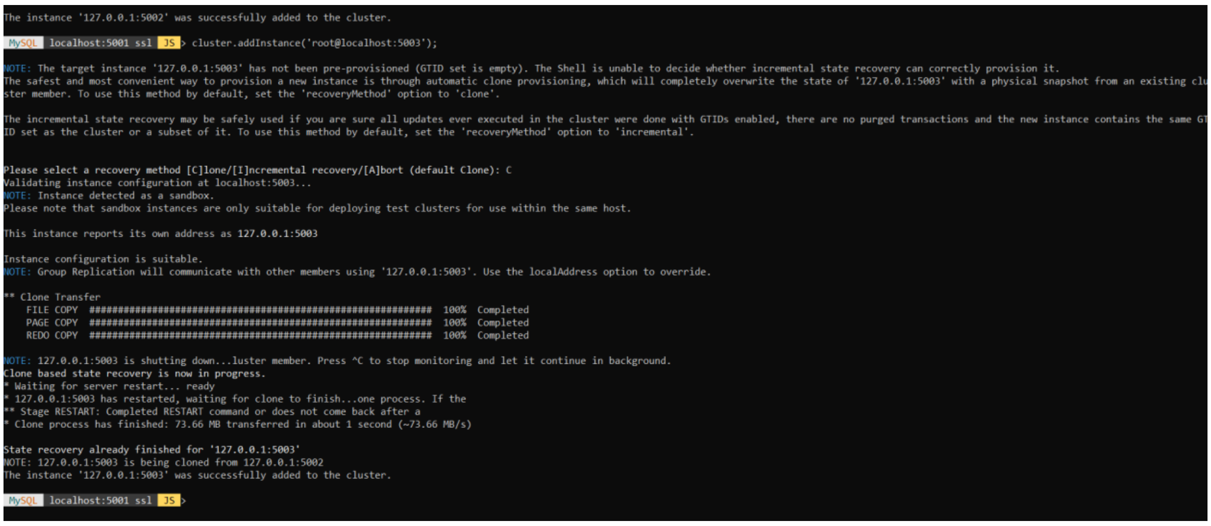
```
 MySQL  localhost:5001 ssl  JS > var cluster = dba.createCluster('itsCluster');
A new InnoDB Cluster will be created on instance '127.0.0.1:5001'.

Validating instance configuration at localhost:5001...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:5001

Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:5001'. Use the localAddress option to override.

Creating InnoDB Cluster 'itsCluster' on '127.0.0.1:5001'...

Adding Seed Instance...
Cluster successfully created. Use Cluster.addInstance() to add MySQL instances.
At least 3 instances are needed for the cluster to be able to withstand up to
one server failure.
```

A new instance is being added to the existing InnoDB Cluster with the address 'root@localhost:5002'. The Shell first asks for the recovery method to be used and then validates the configuration of the target instance at localhost:5002. It checks if the instance is suitable and able to communicate with other members of the cluster. Then, it starts the process of adding the new instance to the cluster using clone-based state recovery. The clone process may take some time depending on the amount of data on the cluster. The Shell monitors the recovery process and waits for it to finish. Once the clone process is complete, the new instance will be fully provisioned and added to the cluster.

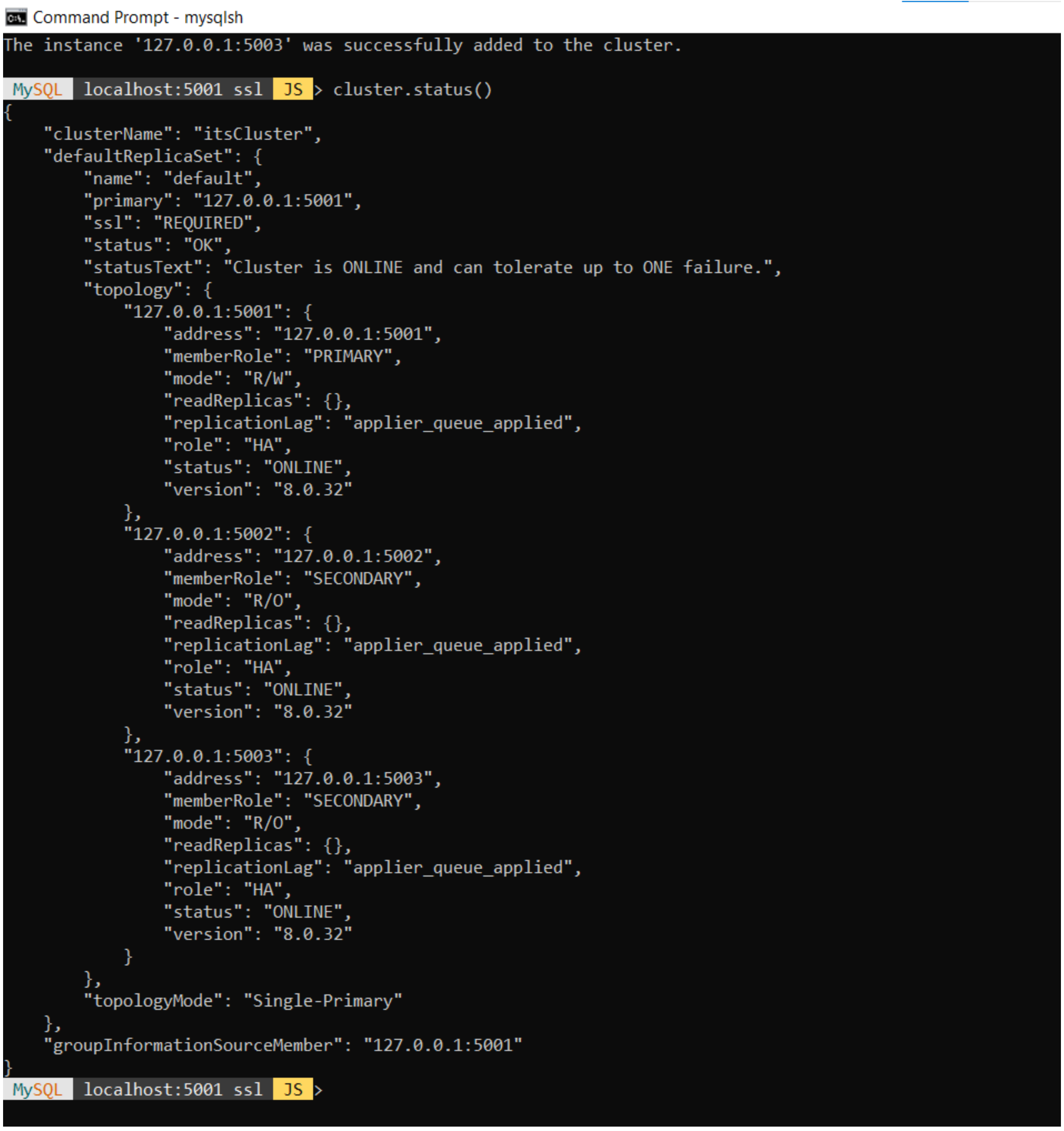


```
MySQL  localhost:5001 ssl  JS > cluster.addInstance('root@localhost:5002');

NOTE: The target instance '127.0.0.1:5002' has not been pre-provisioned (GTID set is empty). The Shell is unable to decide whether incremental state recovery can correctly provision it.
The safest and most convenient way to provision a new instance is through automatic clone provisioning, which will completely overwrite the state of '127.0.0.1:5002' with a physical snapshot from an existing c
ster member. To use this method by default, set the 'recoveryMethod' option to 'clone'.

The incremental state recovery may be safely used if you are sure all updates ever executed in the cluster were done with GTIDs enabled, there are no purged transactions and the new instance contains the same
ID set as the cluster or a subset of it. To use this method by default, set the 'recoveryMethod' option to 'incremental'.


Please select a recovery method [C]lone/[I]ncremental recovery/[A]bort (default Clone): C
Validating instance configuration at localhost:5002...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:5002

Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:5002'. Use the localAddress option to override.

A new instance will be added to the InnoDB Cluster. Depending on the amount of
data on the cluster this might take from a few seconds to several hours.

Adding instance to the cluster...

Monitoring recovery process of the new cluster member. Press ^C to stop monitoring and let it continue in background.
Clone based state recovery is now in progress.

NOTE: A server restart is expected to happen as part of the clone process. If the
server does not support the RESTART command or does not come back after a
while, you may need to manually start it back.

* Waiting for clone to finish...
NOTE: 127.0.0.1:5002 is being cloned from 127.0.0.1:5001
** Stage DROP DATA: Completed
** Clone Transfer
    FILE COPY  ############################################################  100%  Completed
    PAGE COPY  ############################################################  100%  Completed
    REDO COPY  ############################################################  100%  Completed

NOTE: 127.0.0.1:5002 is shutting down...

* Waiting for server restart... ready
* 127.0.0.1:5002 has restarted, waiting for clone to finish...
** Stage RESTART: Completed
* Clone process has finished: 73.66 MB transferred in about 1 second (~73.66 MB/s)

State recovery already finished for '127.0.0.1:5002'
```

A new instance is being added to the existing InnoDB Cluster with the address 'root@localhost:5002'. The Shell first asks for the recovery method to be used and then validates the configuration of the target instance at localhost:5003. It checks if the instance is suitable and able to communicate with other members of the cluster. Then, it starts the process of adding the new instance to the cluster using clone-based state recovery. The clone process may take some time depending on the amount of data on the cluster. The Shell monitors the recovery process and waits for it to finish. Once the clone process is complete, the new instance will be fully provisioned and added to the cluster.



```
The instance '127.0.0.1:5002' was successfully added to the cluster.

 MySQL  localhost:5001 ssl  JS > cluster.addInstance('root@localhost:5003');

NOTE: The target instance '127.0.0.1:5003' has not been pre-provisioned (GTID set is empty). The Shell is unable to decide whether incremental state recovery can correctly provision it.
The safest and most convenient way to provision a new instance is through automatic clone provisioning, which will completely overwrite the state of '127.0.0.1:5003' with a physical snapshot from an existing clu
ster member. To use this method by default, set the 'recoveryMethod' option to 'clone'.

The incremental state recovery may be safely used if you are sure all updates ever executed in the cluster were done with GTIDs enabled, there are no purged transactions and the new instance contains the same GT
ID set as the cluster or a subset of it. To use this method by default, set the 'recoveryMethod' option to 'incremental'.


Please select a recovery method [C]lone/[I]ncremental recovery/[A]bort (default Clone): C
Validating instance configuration at localhost:5003...
NOTE: Instance detected as a sandbox.
Please note that sandbox instances are only suitable for deploying test clusters for use within the same host.

This instance reports its own address as 127.0.0.1:5003

Instance configuration is suitable.
NOTE: Group Replication will communicate with other members using '127.0.0.1:5003'. Use the localAddress option to override.

** Clone Transfer
    FILE COPY  ############################################################  100%  Completed
    PAGE COPY  ############################################################  100%  Completed
    REDO COPY  ############################################################  100%  Completed

NOTE: 127.0.0.1:5003 is shutting down...luster member. Press ^C to stop monitoring and let it continue in background.
Clone based state recovery is now in progress.
* Waiting for server restart... ready
* 127.0.0.1:5003 has restarted, waiting for clone to finish...one process. If the
** Stage RESTART: Completed RESTART command or does not come back after a
* Clone process has finished: 73.66 MB transferred in about 1 second (~73.66 MB/s)

State recovery already finished for '127.0.0.1:5003'
NOTE: 127.0.0.1:5003 is being cloned from 127.0.0.1:5002
The instance '127.0.0.1:5003' was successfully added to the cluster.

 MySQL  localhost:5001 ssl  JS >
```



```
Command Prompt - mysqlsh
The instance '127.0.0.1:5003' was successfully added to the cluster.

 MySQL  localhost:5001 ssl  JS > cluster.status()
{
    "clusterName": "itsCluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "127.0.0.1:5001",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "127.0.0.1:5001": {
                "address": "127.0.0.1:5001",
                "memberRole": "PRIMARY",
                "mode": "R/W",
                "readReplicas": {},
                "replicationLag": "applier_queue_applied",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:5002": {
                "address": "127.0.0.1:5002",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLag": "applier_queue_applied",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            },
            "127.0.0.1:5003": {
                "address": "127.0.0.1:5003",
                "memberRole": "SECONDARY",
                "mode": "R/O",
                "readReplicas": {},
                "replicationLag": "applier_queue_applied",
                "role": "HA",
                "status": "ONLINE",
                "version": "8.0.32"
            }
        },
        "topologyMode": "Single-Primary"
    },
    "groupInformationSourceMember": "127.0.0.1:5001"
}
 MySQL  localhost:5001 ssl  JS >
```

The cluster is named "itsCluster" and consists of three members, each running on a different port, namely 5001, 5002, and 5003.

The status indicates that the cluster is currently online and can tolerate one failure without losing its functionality. The default replica set in the cluster is named "default," and its primary node is the one running on port 5001.

Each member of the replica set has its own address and role within the cluster. The member running on port 5001 is the primary node, and the ones on 5002 and 5003 are secondary nodes.

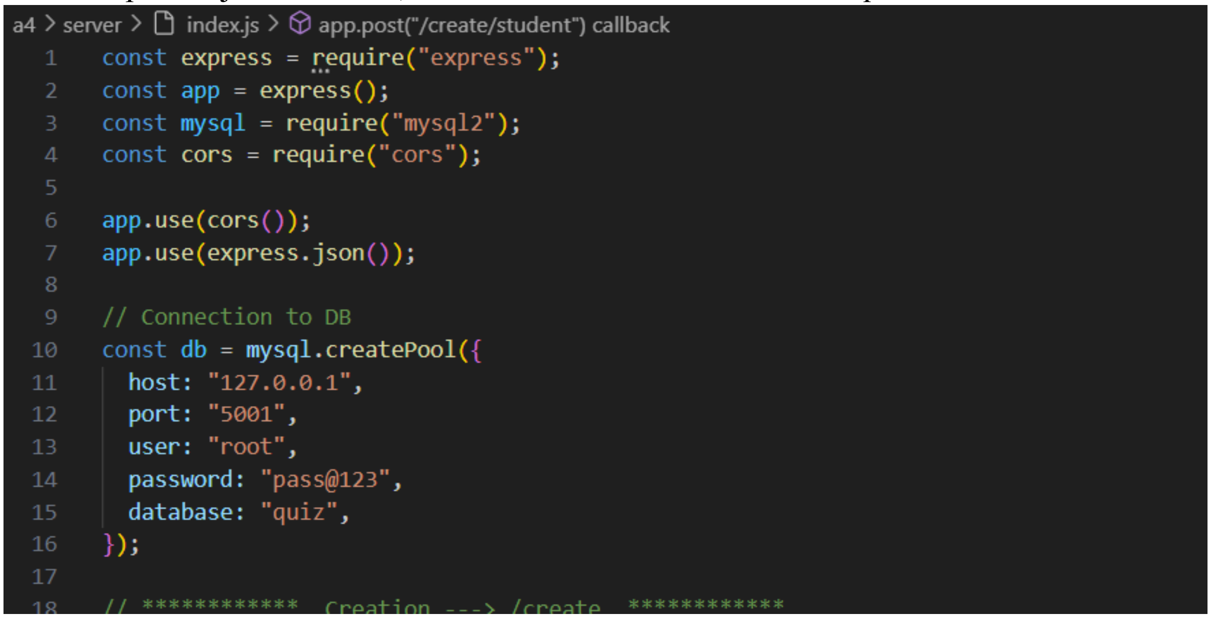
I Have set up a MySQL cluster with three nodes, and each node is running on a different port. The node1 is running on port 5001, node2 is running on port 5002, and node3 is running on port 5003.

This setup allows the nodes to communicate with each other and work together as a cluster. Each node in the cluster can perform read and write operations, depending on its role, which can be either a primary node or a secondary node. The primary node is responsible for handling write operations, while the secondary nodes handle read operations.

In a MySQL cluster, having multiple nodes improves the availability and reliability of the database system. If one node fails, the other nodes can continue to operate, ensuring that the database remains available to users. The replication between the nodes also provides data redundancy, which can help prevent data loss in case of hardware or software failures.

This establishes a connection to a MySQL database from a Node.js application named "quiz app".

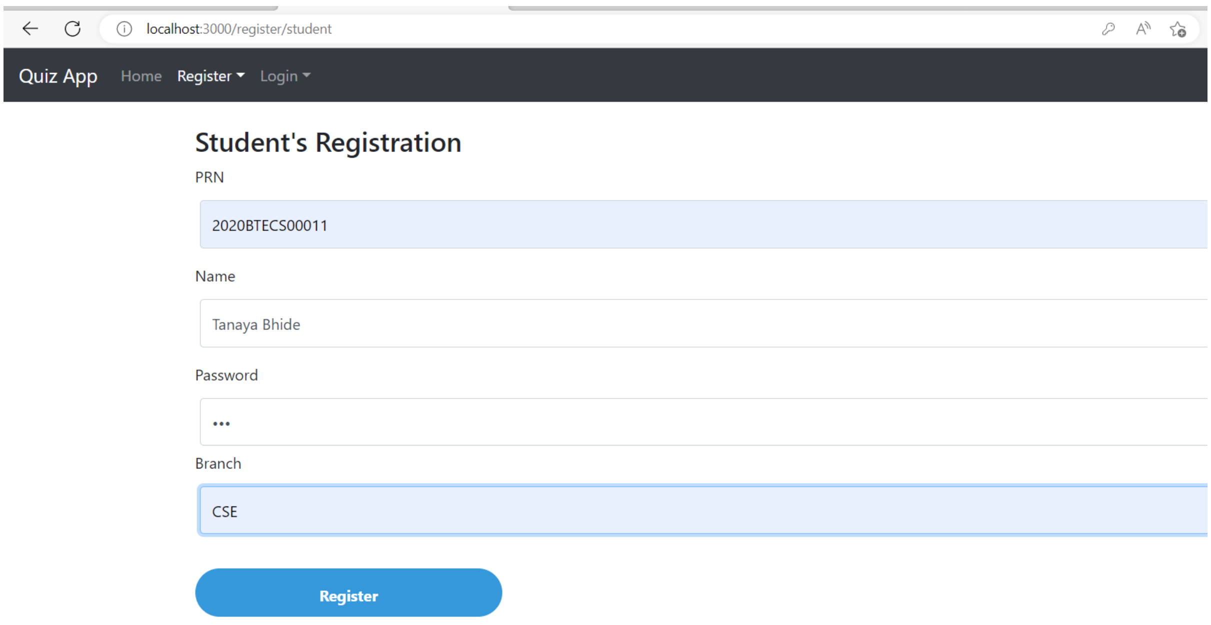
Once the pool object is created, it can be used to execute database queries and transactions.


a4 > server > index.js > app.post("/create/student") callback

```
const express = require("express");
const app = express();
const mysql = require("mysql2");
const cors = require("cors");

app.use(cors());
app.use(express.json());

// Connection to DB
const db = mysql.createPool({
  host: "127.0.0.1",
  port: "5001",
  user: "root",
  password: "pass@123",
  database: "quiz",
});

// ************  Creation ---> /create  ************
```

When a student successfully registers in the application, their registration information is stored in a MySQL database named "node1" as well as in two other databases named "node2" and "node3". These three databases are part of a cluster, which means that they are interconnected and can share data with each other.

The student registration data is stored in a table named "student" in all three databases, and the values entered during registration can be viewed in each of the databases. This means that the registration information is replicated across the cluster and can be accessed from any of the nodes.



localhost:3000/register/student

Quiz App Home Register Login

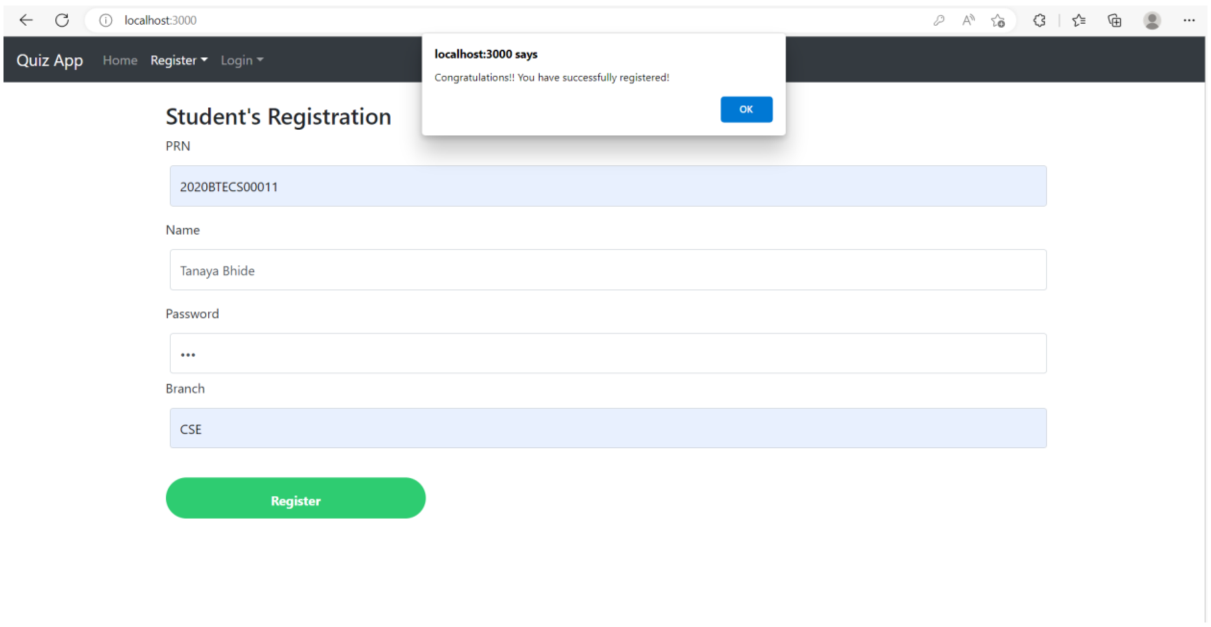
Student's Registration

PRN: 2020BTECS00011

Name: Tanaya Bhide

Password: •••

Branch: CSE



localhost:3000

Quiz App Home Register Login

localhost:3000 says

Congratulations!! You have successfully registered!

Student's Registration

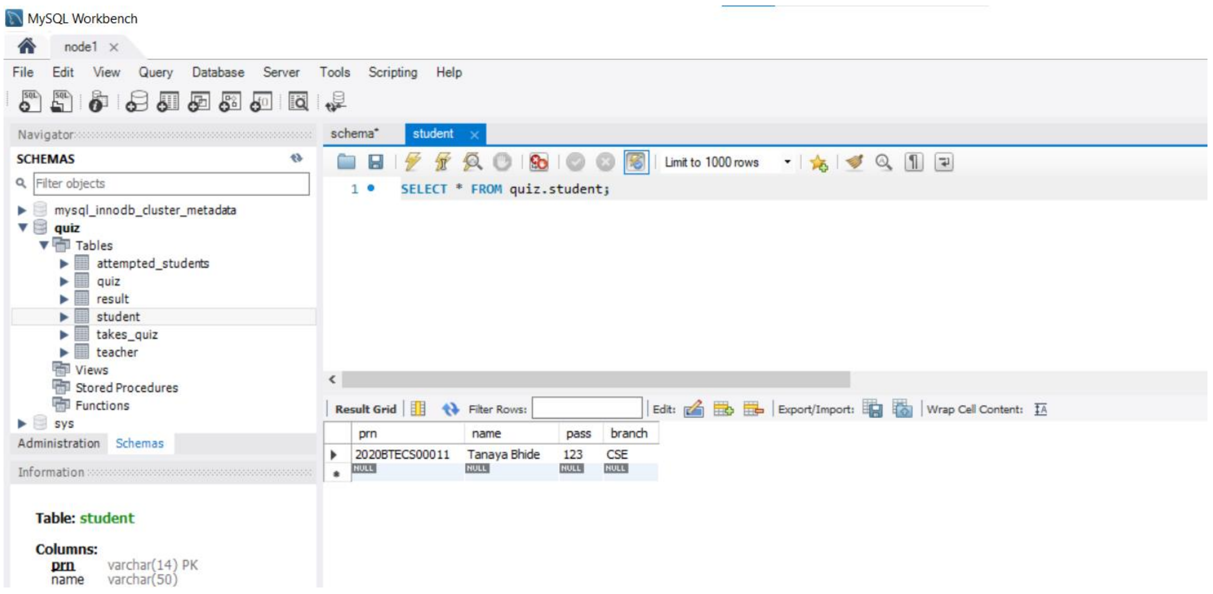
PRN: 2020BTECS00011

Name: Tanaya Bhide

Password: •••

Branch: CSE

Node 1 :



```
mysql_innodb_cluster_metadata
quiz
  Tables
    attempted_students
    quiz
    result
    student
    takes_quiz
    teacher
  Views
  Stored Procedures
  Functions
sys
```

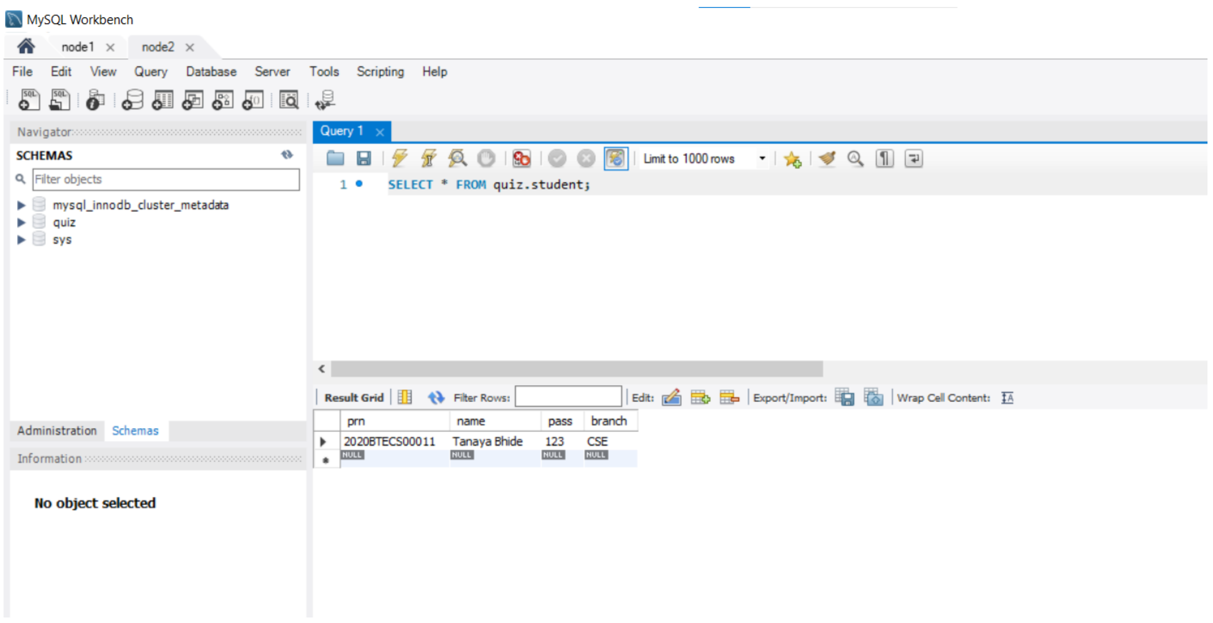
```
SELECT * FROM quiz.student;
```

prn	name	pass	branch
2020BTECS00011	Tanaya Bhide	123	CSE
NULL	NULL	NULL	NULL

Table: student

Columns:
prn varchar(14) PK
name varchar(50)

Node 2 :



```
mysql_innodb_cluster_metadata
quiz
sys
```

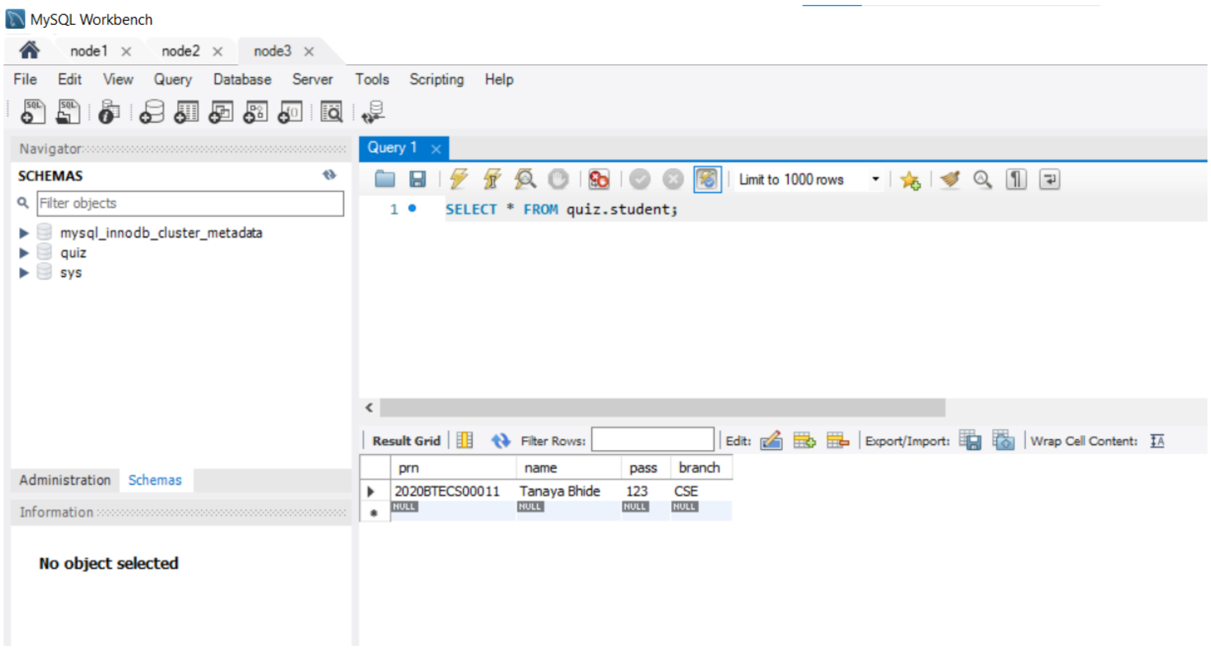
```
SELECT * FROM quiz.student;
```

prn	name	pass	branch
2020BTECS00011	Tanaya Bhide	123	CSE
NULL	NULL	NULL	NULL

No object selected

Node 3 :

‘



```
SELECT * FROM quiz.student;
```

prn	name	pass	branch
2020BTECS00011	Tanaya Bhide	123	CSE
NULL	NULL	NULL	NULL

Secondary nodes are typically not allowed to perform write operations directly. The primary node is responsible for handling all write requests and maintaining the consistency of data across the cluster.

When a write operation is performed on the primary node, the changes are replicated to the secondary nodes, which update their copies of the data accordingly. This process ensures that all nodes have consistent and up-to-date copies of the data.