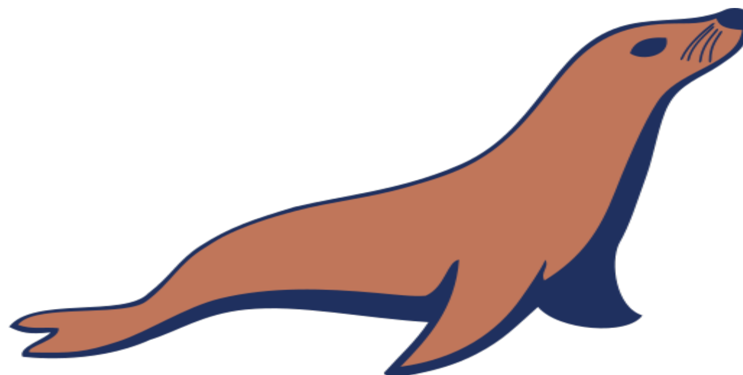
Data Modelling Tools

AUTM08016

Topic 8
Database Replication



Dr Diarmuid Ó Briain

Version 1.0 [01 January 2024]

TUS

Ollscoil Teicneolaíochta na Sionainne:
Lár Tíre, An tIarthar Láir

Technological University of the Shannon:
Midlands Midwest

Copyright © 2024 C²S Consulting

Licenced under the EUPL, Version 1.2 or – as soon they will be approved by the European Commission - subsequent versions of the EUPL (the "Licence");

You may not use this work except in compliance with the Licence.

You may obtain a copy of the Licence at:

https://joinup.ec.europa.eu/sites/default/files/custom-page/attachment/eupl_v1.2_en.pdf

Unless required by applicable law or agreed to in writing, software distributed under the Licence is distributed on an "AS IS" basis, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the Licence for the specific language governing permissions and limitations under the Licence.

Dr Diarmuid Ó Briain

Linux Version

```
~$ lsb_release -a | grep Description
Description:   Ubuntu 22.04.3 LTS
```

Apache2 Version

```
~$ apache2 -v
Server version: Apache/2.4.52 (Ubuntu)
Server built:   2023-10-26T13:44:44
```

MariaDB Version

```
~$ mariadb --version
mariadb  Ver 15.1 Distrib 10.6.12-MariaDB, for debian-linux-gnu
(x86_64) using  EditLine wrapper
```

python version

```
~$ python3 --version
Python 3.10.12
```

Table of Contents

1. Introduction 5
1.1 Objectives 5
2. MariaDB Connector/Python 6
2.1 Default Data Generator 7
3. Simple MariaDB Connector module 10
4. Custom Database interface 12
4.1 Python Virtual Environment 12
4.2 Virtual Environment modules 13
4.3 How the Database connector module works 14
4.4 Login to the application 19
4.5 Read from the database 20
4.6 Write to the Database 21
4.7 Delete from the Database 22
5. Moving the application into production on Apache2 24
5.1 Apache2 site configuration file 24
5.2 Install the WSGI and enable it for Apache2 25
5.3 Launch Apache2 Server 25
5.4 Test the new service in production 26
6. Laboratory #1 27
6.1 Create a custom interface to the counties database 27

Table of Figures

Figure 1: MariaDB Connector/Python......6
Figure 2: Basic query via MariaDB Connector/Python......6
Figure 3: mariadb connector module......10
Figure 4: Testing mariadb_conn.py......11
Figure 5: Python sys.path......12
Figure 6: Connect to MariaDB Database......14
Figure 7: Select function......14
Figure 8: Insert function......15
Figure 9: Delete function......15
Figure 10: Browse to development webserver......17
Figure 11: How index.html is built......18
Figure 12: Root of custom application......19
Figure 13: Read from the Database......20
Figure 14: Write to the database......21
Figure 15: View the newly added user......22
Figure 16: Delete from the database......23
Figure 17: Read to confirm the delete......23
Figure 18: View service from another workstation......26

1. Introduction

MariaDB database replication is a powerful feature that allows for the synchronisation of data between multiple servers. This can be used to achieve several benefits, such as, high availability, data redundancy and load balancing. In this topic a number of replication mechanisms will be discussed and an example will be demonstrated to expand on the work of the earlier topics.

1.1 Objectives

By the end of this topic the learner will be able to

- Develop a replication database server with its own read-only front-end webpage.

2. Distributed Databases

A distributed database is a database in which storage devices are not all attached to a common processing unit. The data is stored across several databases generally at different sites each managed by a Relational Database Management System (RDBMS) that can run independently.

These can be managed by a Distributed Database Management System. These databases can be stored on multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers. The location of data and degree of individual sites impact query optimisation, concurrency control and recovery.

Distributed data transactions, like other transactions, are governed the Atomicity, Consistency, Isolation, Durability (ACID) properties that guarantee that database transactions are processed reliably.

- **Atomicity** requires that each transaction is "all or nothing": if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged.
- **Consistency** ensures that any transaction will bring the database from one valid state to another.
- **Isolation** ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed one after the other.
- **Durability** means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors.

2.1 Distributed Data Independence

The distribution of portions of the database to various locations should be invisible to users of the database. Existing applications should continue to operate successfully:

- when a distributed version of the DBMS is first introduced; and
- when existing distributed data are redistributed around the system.

2.2 Distributed Transaction Atomicity

Users should be able to write transactions that access and update data at several sites. Transactions are atomic, all changes persist if the transaction commits, or roll-back if transaction aborts.

2.3 Distributed Databases on slow networks

If sites are connected by slow networks, the Independence and Atomicity properties are hard to support efficiently. Users have to be aware of where data is located because Distributed Data Independence and Distributed Transaction Atomicity are not supported. In situations where an organisation have globally distributed sites, these properties may not even be desirable due to the administrative overhead of making locations of data transparent.

2.4 Types of Distributed Databases

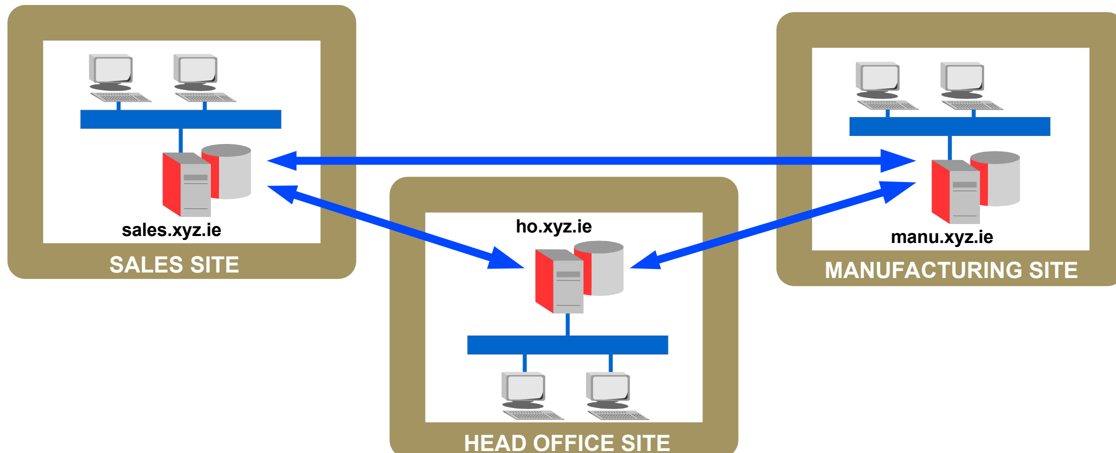


Figure 1: Homogeneous - distributed databases

2.4.1 Homogeneous

A homogeneous distributed database system is a network of two or more databases that reside on one or more machines. In Figure 1 there are three databases: ***ho***, ***manu***, and ***sales*** connected as a distributed database. An application can simultaneously access or modify the data in several databases in a single distributed environment.

For a client application, the location and platform of the databases are transparent. You can also create synonyms for remote objects in the distributed system so that users can access them with the same syntax as local objects. For example, A single query from a sales client on the local database at the sales office can retrieve data from the pricing table on the local database and the products table on the database at the manufacturing site.

```
MariaDB [(none)]> SELECT a.Pricing, b.Products
                -> FROM  Sales a INNER JOIN Manufacturing b
                -> ON a.Product_no = b.Product_no
                -> WHERE a.Product_no = 23456;
```

In this way, a distributed system gives the appearance of native data access. Users on `mfg` do not have to know that the data they access resides on remote databases.

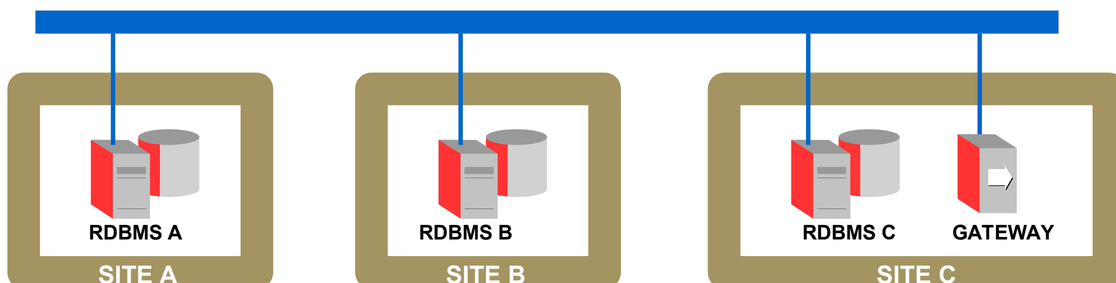


Figure 2: Hetrogeneous - distributed databases

2.4.2 Heterogeneous

Different sites run different RDBMSs separately and are connected somehow to enable access to data from multiple sites. In the example illustrated in Figure 2, Site A running MySQL, Site B, Microsoft SQL and Site C, Oracle 12c. Different sites using different schema and software. Differences in schema can be a major problem for query processing and transaction processing. Sites may not be aware of each other and may provide only limited facilities for cooperation in transaction processing. Different nodes may have different hardware, software and data structures at various nodes or locations are also incompatible.

Gateway protocols allow applications to connect to multiple Database servers or other gateways across a network, using a number of communications protocols and middleware Application Program Interfaces (APIs) to establish a distributed processing and distributed database environment. Examples: Open Database Connectivity (ODBC), Java Database Connectivity (JDBC) and Object Linking and Embedding, Database (OLE-DB).

2.5 Distributed Database Architectures

2.5.1 Client – Server Architecture

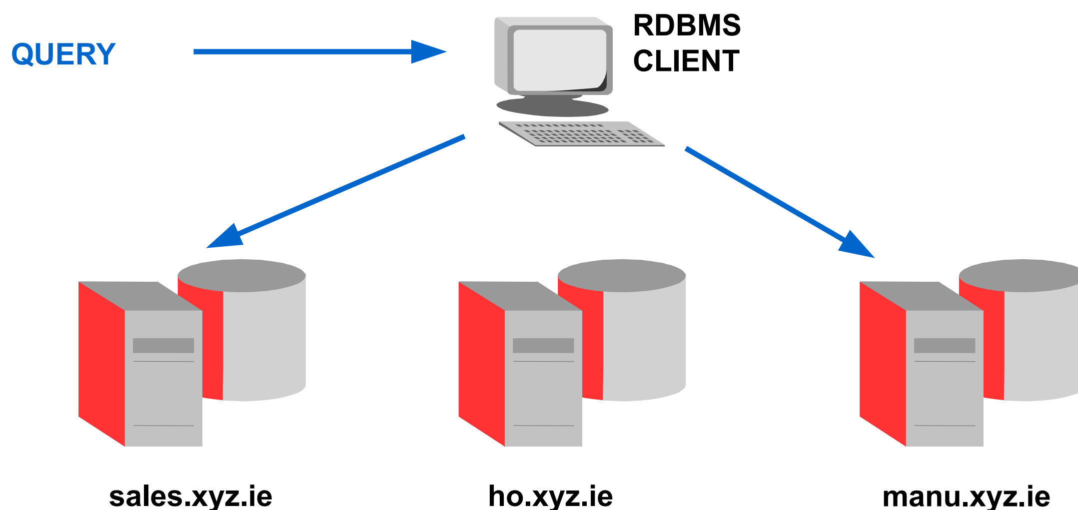


Figure 3: Client - Server Architecture

A system that has one or more client process and one or more server processes. Client sends a query to a server, and the server processes the query returning the result to the client.

A client connects directly to a database server. A direct connection occurs when a client connects to a server and accesses information from a database contained on that server. i.e. a connection to the sales database to access the Pricing table on this database can be achieved by issuing one of the following SQL Queries:

```
MariaDB [(none)]> SELECT * FROM Pricing;
MariaDB [(none)]> SELECT * FROM Pricing where Product_no = '23456';
```

This query is direct because an object on a remote database is not being accessed.

2.5.2 Collaborated Server Architecture

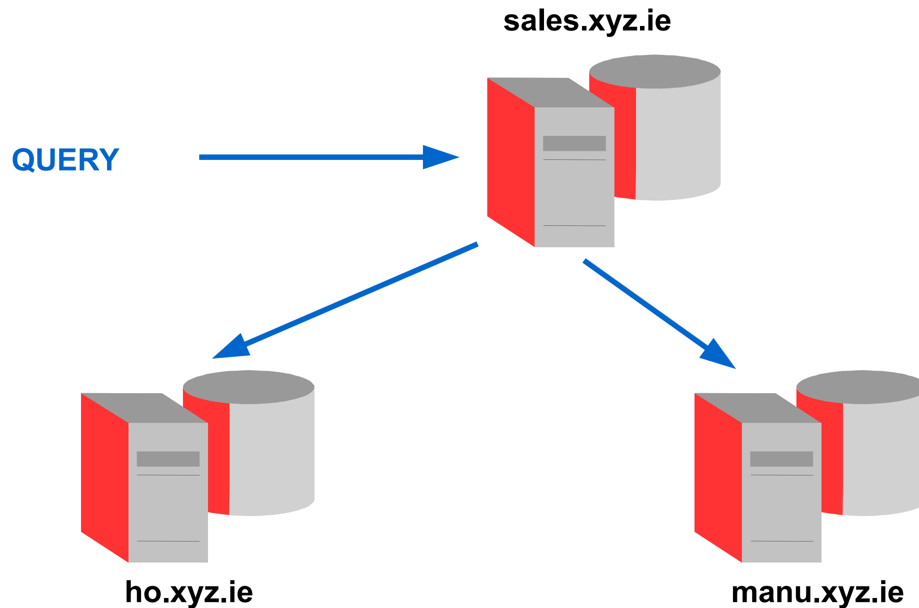


Figure 4: Collaborated Server Architecture

A collaborated connection occurs when a client connects to a server and then accesses information contained in a database on a different server. For example, a connection to the sales database that requires access to the Product table on the remote Manufacturing database can be achieved by issuing the SQL Query:

```
sql> SELECT * FROM Product@manu.xyz.ie;
```

This query is indirect because the object being accessed is not on the database that is currently connected.

A query requires access to data at other servers, it generates sub queries to be executed by other servers and pieces the results together to answer the original query.

```
MariaDB [(none)]> SELECT a.Pricing, b.Products
                   -> FROM  Sales a INNER JOIN Manufacturing b
                   -> ON a.Product_no = b.Product_no
                   -> WHERE a.Product_no = 23456;
```

2.5.3 Middleware

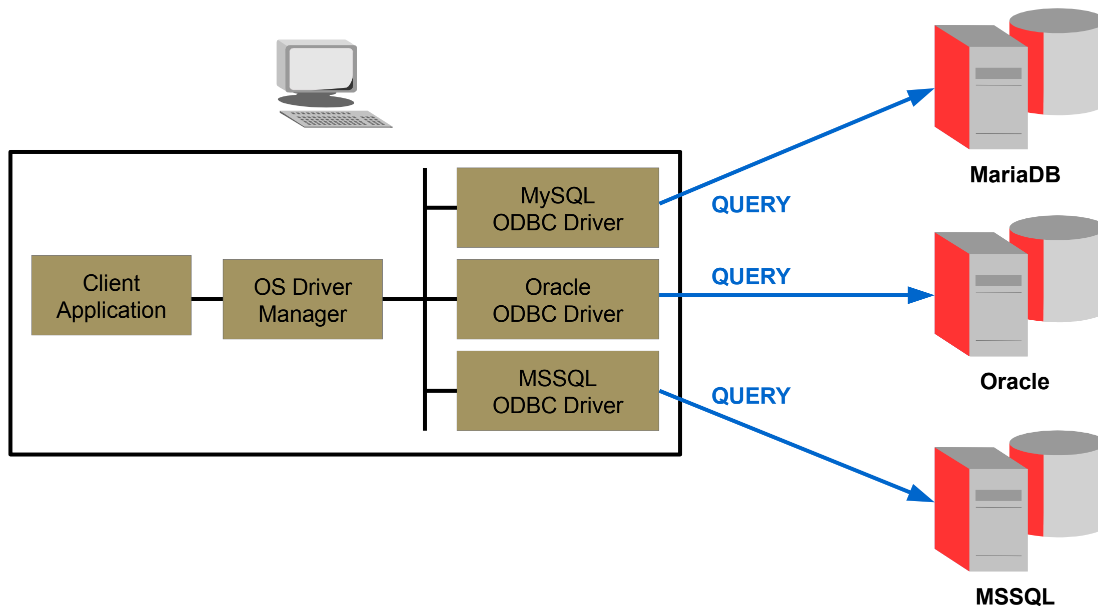


Figure 5: Middleware

SQL-oriented Data Access Middleware is a software layer between applications and database servers. Middleware has the capability to translate generic SQL into the SQL specific to the database.

- Open Database Connectivity (**ODBC**) – supported by most database vendors.
- Object Linking and Embedding Database (**OLE-DB**) - Microsoft enhancement of ODBC.
- Java Database Connectivity (**JDBC**) - Special Java classes that allow Java applications/applets to connect to databases.
- Common Object Request Broker Architecture (**CORBA**) – specification of object-oriented middleware.
- Distributed Component Object Model (**DCOM**) – Microsoft's version of CORBA – not as robust as CORBA over multiple platforms.

Considering ODBC further, this is an Application Programming Interface (API) that provides a common language for application programs to access and process SQL databases independent of the particular RDBMS that is accessed. It requires the following parameters:

- ODBC driver needed.
- Back-end server name.
- Database name.
- User ID and Password.

2.6 High Availability Cluster Architecture

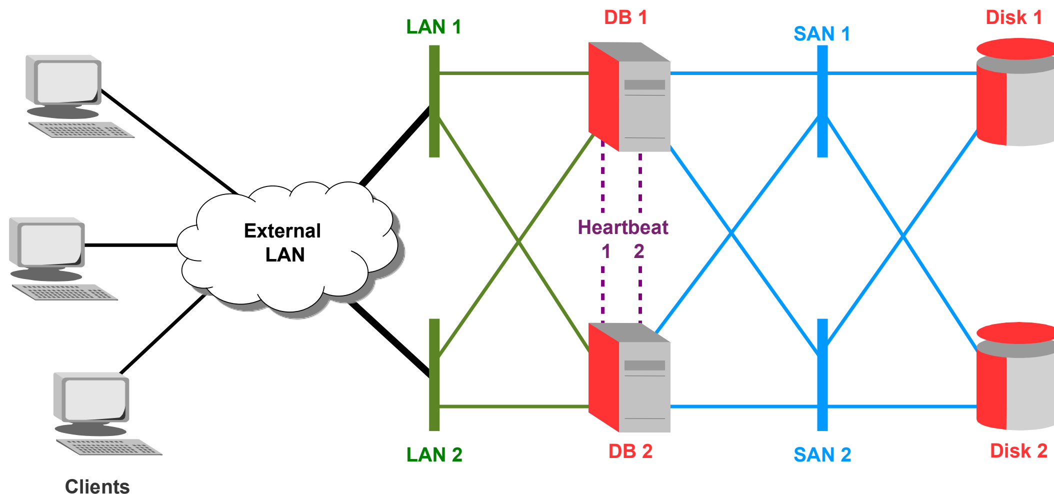


Figure 6: HA Cluster

High-availability (HA) clusters are groups of servers that support server applications that can be reliably utilised with a minimum of down-time. They operate by harnessing redundant servers in groups or clusters that provide continued service when system components fail. Without clustering, if a server running a database crashes, the application will be unavailable until the crashed server is fixed. HA clustering remedies this situation by detecting hardware/software faults, and immediately restarting the application on another system without requiring administrative intervention, a process called fail-over.

HA cluster implementations attempt to build redundancy into a cluster to eliminate single points of failure, including multiple network connections and data storage which is redundantly connected via Storage Area Networks (SAN).

HA clusters usually use a heartbeat private network connection which is used to monitor the health and status of each node in the cluster. One subtle but serious condition all clustering software must be able to handle is split-brain, which occurs when all of the private links go down simultaneously, but the cluster nodes are still running. If that happens, each node in the cluster may mistakenly decide that every other node has gone down and attempt to start services that other nodes are still running. Having duplicate instances of services may cause data corruption on the shared storage.

Figure 6 demonstrates the most common arrangement for a HA cluster. This is called a two-node cluster. This is is the minimum required to provide redundancy, but many clusters consist of many more, sometimes dozens of nodes. Configurations can sometimes be categorised into one of the following models:

Active/active — Traffic intended for the failed node is either passed onto an existing node or load balanced across the remaining nodes. This is usually only possible when the nodes utilise a homogeneous software configuration.

Active/passive (N+1) — Provides a fully redundant instance of each node, which is only brought online when its associated primary node fails. This configuration typically requires the most extra hardware.

N+M — In cases where a single cluster is managing many services, having only one dedicated fail-over node may not offer sufficient redundancy. In such cases, more than one (M) standby servers are included and available. The number of standby servers is a trade-off between cost and reliability requirements.

N-to-1 — Allows the fail-over standby node to become the active one temporarily, until the original node can be restored or brought back online, at which point the services or instances must be failed-back to it in order to restore high availability.

N-to-N — A combination of active/active and N+M clusters, N to N clusters redistribute the services, instances or connections from the failed node among the remaining active nodes, thus eliminating (as with active/active) the need for a 'standby' node, but introducing a need for extra capacity on all active nodes.

2.7 Storing data

1	Student ID	Name	Town
2	K001234	Lovelace	Nottingham
3	K001235	Babbage	London
4	K001236	Menabrea	Turin

Figure 7: Database relations

Relations are stored across several sites and to reduce message-passing costs a relation maybe fragmented across many sites. Fragmentation brakes a relation down into smaller relations and stores the fragments at different sites.

2.7.1 Horizontal Fragmentation (HF)

If the relation is divided such that a subset of rows is stored at SITE_A and another subset is stored at SITE_B and even more stored at SITE_C etc..., this is called Horizontal Fragmentation (HF), and the original relation is obtained by taking the union of all the sets.

2.7.2 Vertical Fragmentation (VF)

If the Student relation is divided in two sets say SET_A and SET_B such that half of the attributes of Student is in SET_A and the other half is in SET_B. This is called Vertical Fragmentation (VF) as a relation is fragmented along columns. The original relation is obtained by natural join of all the sets. For natural join we require at least one Primary Key attribute which is common to all the sets.

2.8 Replication

Figure 8: Replication

Replication is the storing several copies of a relation or fragment. The entire relation can be stored at one or more sites. This is done to either:

- **High Availability (HA)** – If a site contains replicated data goes down, then we can use another site.
- **Faster Query Evaluation** – Queries are executed faster by using local copy of a relation instead of going to a remote site.

2.8.1 Synchronous/Asynchronous replication

There are two types of replication are Synchronous and Asynchronous replication.

Synchronous replication writes the data to both the primary and to the secondary site database at the same time. In doing this, the data remains completely current and identical. The process works quickly and there is a extremely small margin of error. Because of this, it is ideal for disaster recovery and is the method preferred for projects that require absolutely no data loss. A high speed SAN is required to support such replication

Asynchronous replication also writes data to both a primary and secondary site, however with this process there is a delay when data is copied from one to another. This approach to data backup is often termed, "*Store and Forward*". With this type of replication the data first writes to the primary database and then commits the data for replication to a secondary source: either memory or disk-based. Finally, the data copies at scheduled intervals to the target. This method can work over longer distances and over slow speed links than synchronous replication, so at times it may be the only viable option.

Type of Replication	Synchronous	Asynchronous
Recovery Point Objective	Zero	15 minutes to a few hours
Distance Limitations	Best if both SANs are in the same datacentre.	Anywhere with a good data connection.
Cost	Most expensive type of SAN solution.	Not as expensive as Synchronous but more expensive than basic SANs.

Figure 9: Summary of replication types

3. Distributed Database Security

Part of planning for a distributed relational database involves the decisions to be made about securing distributed data.

These decisions include:

- What systems should be made accessible to users in other locations and which users in other locations should have access to those systems.
- How tightly controlled access to those systems should be. For example, should a user password be required when a conversation is started by a remote user?
- Is it required that passwords flow over the wire in encrypted form?
- Is it required that a user profile under which a client job runs be mapped to a different user identification or password based on the name of the relational database being connected to?
- What data should be made accessible to users in other locations and which users in other locations should have access to that data.
- What actions those users should be allowed to take on the data.
- Whether authorisation to data should be centrally controlled or locally controlled.

3.1 MariaDB Security

When thinking about security within a MariaDB server the following topics come to the fore.

3.1.1 General factors that affect security

These include choosing good passwords, not granting unnecessary privileges to users, ensuring application security by preventing SQL injections and data corruption, and others.

3.1.2 Security of the installation itself

The data files, log files, and the all the application files of the installation should be protected to ensure that they are not readable or writeable by unauthorised parties.

3.1.3 Access control and security within the database system

Access control for database users and databases granted with access to the databases, views and stored programs in use within the database.

3.1.4 Network security of MariaDB and the system.

The security is related to the grants for individual users, but it may also be beneficial to restrict MariaDB so that it is available only locally on the MariaDB server host, or to a limited set of other hosts.

3.1.5 Secure Sockets Layer (SSL)

MariaDB supports secure (encrypted) connections between MariaDB clients and the server using the Secure Sockets Layer (SSL) protocol. Secure connections are based on the OpenSSL API and are available through the MariaDB C API. Replication uses the C API, so secure connections can be used between master and slave servers.

To check whether a MariaDB daemon supports SSL, examine the value of the **`have_ssl`** system variable:

```
MariaDB [(none)]> SHOW VARIABLES LIKE 'have_ssl';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_ssl      | YES   |
+---------------+-------+
```

If the value is **`YES`**, the server supports SSL connections. If the value is **`DISABLED`**, the server is capable of supporting SSL connections but was not started with the appropriate **`--ssl-xxx`** options to enable them to be used.

3.1.6 Backups

Ensure that there are adequate and appropriate backups of the database files, configuration and log files. A recovery solution needs to be in place and tested so a failed database can be successfully recovered from the information in the backups.

3.2 Securing Distributed Databases

Enterprise class distributed databases support all of the security features that are available with a non-distributed database environment in a distributed database systems, including:

- Password authentication for users and roles.
- Some types of external authentication for users and roles including:
    - Kerberos for connected user links.
    - Distributed Computing Environment (DCE) for connected user links.
- Login packet encryption for client-to-server and server-to-server connections.

3.2.1 Kerberos

Kerberos is a network authentication protocol which works on the basis of 'tickets' to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner. It's design is aimed primarily at a client–server model and it provides mutual authentication, both the user and the server verify each other's identity. Kerberos protocol messages are protected against eavesdropping and replay attacks.

Kerberos builds on symmetric key cryptography and requires a trusted third party, and optionally may use public-key cryptography during certain phases of authentication. Kerberos uses TCP port 88 by default.

The three elements of Kerberos comprise the Key Distribution Centre (KDC), the client user and the server with the desired service to access, i.e. the Distributed Database. The KDC is performs two service functions: the Authentication Service (AS) and the Ticket-Granting Service (TGS).

3.2.2 Distributed Computing Environment (DCE)

DCE from the Open Software Foundation (OSF) is a set of integrated network services that work across multiple systems to provide a distributed environment. The network services include Remote Procedure Calls (RPC), directory service, security service, threads, distributed file service, disk-less support, and distributed time service.

DCE is the middleware between distributed applications and the system/network services and is based on a client/server model. By using the services and tools that DCE provides, users can create, use, and maintain distributed applications that run across a heterogeneous environment.

4. Database Replication

4.1 Installation of software

Follow the same process as used on the server in Workshop 2/3, which now becomes the Primary (Master) Server build a Replica (Slave) Server:

- `mariadb-server`
- `mariadb-client`
- `libmariadb3`
- `libmariadb-dev`
- `openssh-server`
- `apache2`

4.2 MariaDB Database Replication

Considering a MariaDB database replication setup and how to get everything working smoothly again after a server crash, or for whatever reason it is necessary to have a replica database that maintains its mirror with the primary database.

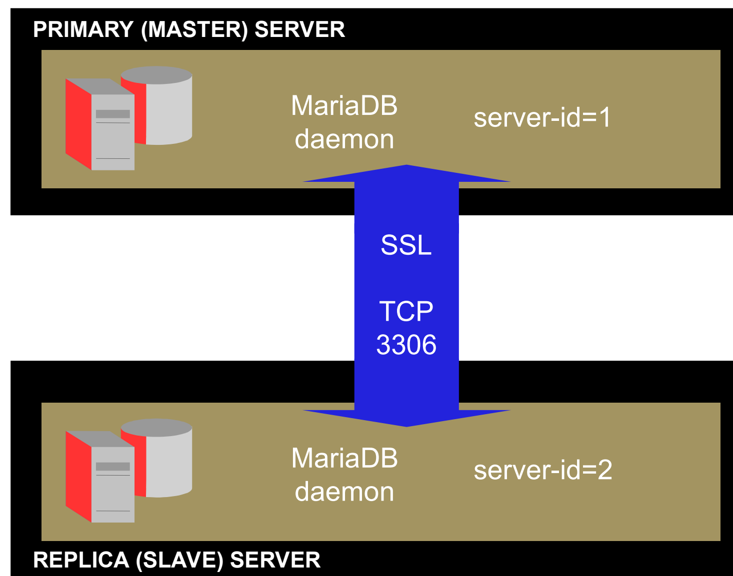


Figure 10: Database replication

In Figure 10 there are two MariaDB servers, one which is the Primary (Master) MariaDB daemon and the other which is the Replica (Slave) MariaDB daemon. The Master maintains a log of every action performed on it in a binary file and the Slave monitors the binary log on the Master and whenever a new event happens, it will copy the event in the Processor database. Let the database **`Eng`** created on the Raspberry Pi be the Primary (Master). Install MariaDB on a second computer on the network. For the purpose of this workshop a Virtual Machine (VM) on a computer.

4.3 Test network

For the purpose of demonstration, the following network, in Figure 11 will be used.

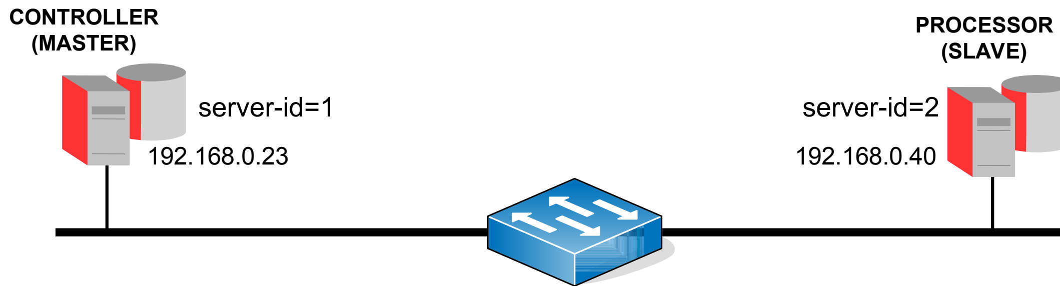


Figure 11: Database Replication Network

4.4 Configuring the Primary Server

4.4.1 Replication user on Primary Server

Create a user on the Primary (Master) server that the Replica (Slave) can connect as. In this case I will use the username '**replication_user**'. Log into MariaDB as **root** and create the user:

```
[Primary]~$ sudo mysql -u root

MariaDB [(none)]> GRANT REPLICATION SLAVE ON *.* TO
                  -> 'replication_user'@'%'
                  -> IDENTIFIED BY 'replication_pass';
Query OK, 0 rows affected (0.002 sec)
```

Reload the privileges from the grant tables in the MariaDB database.

```
MariaDB [(none)]> FLUSH PRIVILEGES;

MariaDB [(none)]> SELECT User, Host FROM mysql.user;
+------------------+-----------+
| User             | Host      |
+------------------+-----------+
| replication_user | %         |
| admin            | localhost |
| enguser          | localhost |
| mariadb.sys      | localhost |
| mysql            | localhost |
| phpmyadmin       | localhost |
| root             | localhost |
+------------------+-----------+
7 rows in set (0.005 sec)
```

4.4.2 Primary Server Binary log

Stop the Primary (Master) Server.

```
[Primary]~$ sudo systemctl stop mariadb.service
```

Edit the configuration by adding the following section:

```
[Primary]~$ cd /etc/mysql/mariadb.conf.d
[Primary]/etc/mysql/mariadb.conf.d$ sudo vi 50-server.cnf
~~~~
[mariadb]
log_bin
server_id=1
log_basename=master1
binlog_format=mixed
binlog_do_db=Eng
bind_address=0.0.0.0
~~~~
:wq!
```

- **log-bin** - Enable binary logging
- **server_id** - Unique Server ID
- **log-basename** - Unique name for replication logs
- **binlog-format** - Statement-Based, Row-Based or Mixed Logging
- **binlog_do_db** - allow master to write statements and transactions affecting databases that match a specified name into its binary log
- **bind-address** - An IP address or 0.0.0.0 as a wildcard.

Start the Primary (Master) Server

```
[Primary]~$ sudo systemctl start mariadb.service
```

Confirm the configuration changes were made.

```
[Primary]~$ sudo mysql -u root

MariaDB [(none)]> SHOW VARIABLES LIKE 'bind%';
+---------------+---------+
| Variable_name | Value   |
+---------------+---------+
| bind_address  | 0.0.0.0 |
+---------------+---------+
1 row in set (0.008 sec)
```

```
MariaDB [(none)]> SHOW VARIABLES LIKE 'binlog_format';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | MIXED |
+---------------+-------+
1 row in set (0.009 sec)


MariaDB [(none)]> SHOW VARIABLES LIKE 'server%';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 1     |
+---------------+-------+
1 row in set (0.009 sec)
```

4.4.3 Clean the logs on Primary and restart the server

Determine the log directory for the database.

```
MariaDB [(none)]> SHOW VARIABLES LIKE 'log_bin_basename';
+------------------+----------------------------+
| Variable_name    | Value                      |
+------------------+----------------------------+
| log_bin_basename | /var/lib/mysql/master1-bin |
+------------------+----------------------------+
1 row in set (0.008 sec)
```

Clear out any logs from **/var/log/mysql** as indicated by the previous command:

```
[Primary]~$ sudo rm /var/log/mysql/*
```

Restart the MariaDB Service and confirm status.

```
[Primary]~$ sudo systemctl restart mariadb.service

[Primary]~$ sudo systemctl status mariadb.service
● mariadb.service - MariaDB 10.3.29 database server
   Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; vendor preset:
   Active: active (running) since Tue 2021-08-31 17:48:20 IST; 4s ago
     Docs: man:mysqld(8)
           https://mariadb.com/kb/en/library/systemd/
  Process: 19333 ExecStartPre=/usr/bin/install -m 755 -o mysql -g root -d /var/r
  Process: 19334 ExecStartPre=/bin/sh -c systemctl unset-environment _WSREP_STAR
  Process: 19336 ExecStartPre=/bin/sh -c [ ! -e /usr/bin/galera_recovery ] && VA
  Process: 19433 ExecStartPost=/bin/sh -c systemctl unset-environment _WSREP_STA
  Process: 19435 ExecStartPost=/etc/mysql/debian-start (code=exited, status=0/SU
 Main PID: 19401 (mysqld)
   Status: "Taking your SQL requests now..."
    Tasks: 32 (limit: 2059)
   CGroup: /system.slice/mariadb.service
           └─19401 /usr/sbin/mysqld
```

4.4.4 Lock the Primary while configuring the Replica Database

Prevent any new data being added on the Primary (Master) Server. Log into MariaDB and flush tables with a read lock. Note the **File** and **Position** details.

```
[Primary]~$ sudo mysql -u root

MariaDB [(none)]> FLUSH TABLES WITH READ LOCK;
Query OK, 0 rows affected (0.003 sec)

MariaDB [(none)]> SHOW BINLOG STATUS;
+--------------------+----------+--------------+------------------+
| File               | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+--------------------+----------+--------------+------------------+
| master1-bin.000002 |      330 | Eng          |                  |
+--------------------+----------+--------------+------------------+
1 row in set (0.001 sec)
```

4.4.5 Dump a copy of the Primary Database

Dump the existing data from the Primary (Master) Server to a file and SFTP the file over to the Replica (Slave) Server.

```
[Primary]~$ sudo mysqldump -u root Eng > Eng.sql

[Primary]~$ ls
Eng.sql
```

4.5 Configuring the Replica Server

4.5.1 Confirm connection to Primary Server over the network

Before configuring the Replica (Slave) Server, confirm that from the Replica (Slave) Server it is possible to connect to the Primary (Master) Server with the user configured for replication.

```
[Replica]~$ sudo mysql -h 192.168.0.23 -u replication_user -p
Enter password: replication_pass
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 32
Server version: 10.6.12-MariaDB-0ubuntu0.22.04.1-log Ubuntu 22.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]> QUIT;
bye
```

4.5.2 Create a local version of the database on the Replica Server

Create a local version of the database on the Replica (Slave), create a local user with rights to the local version of the database also.

```
MariaDB [(none)]> CREATE DATABASE Eng;
Query OK, 1 row affected (0.000 sec)

MariaDB [(none)]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| Eng                |
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.000 sec)

MariaDB [(none)]> CREATE USER 'replicauser'@'localhost' IDENTIFIED BY
'replicapass';
Query OK, 0 rows affected (0.013 sec)

MariaDB [(none)]> GRANT ALL ON Eng.* TO 'replicauser'@'localhost';
Query OK, 0 rows affected (0.12 sec)

MariaDB [(none)]> QUIT;
Bye
```

4.5.3 MariaDB Replica Server configuration file

Stop the Replica (Slave) server.

```
[Replica]~$ sudo systemctl stop mariadb.service
```

Edit the configuration by adding the following section:

```
[Replica]~$ cd /etc/mysql/mariadb.conf.d
/etc/mysql/mariadb.conf.d$ sudo vi 50-server.cnf
~~~~
~~~~
[mariadb]
server_id = 2
replicate_do_db = Eng
~~~~
~~~~
:wq!
```

- **`server_id`** - Unique Server ID
- **`replicate_do_db`** – Database to replicate

4.5.4 Clean the logs on the Replica Server before starting

Clear any replication logs from **/var/lib/mysql**.

```
[Replica]~$ sudo rm /var/log/mysql/*

[Replica]~$ sudo systemctl restart mariadb.service

[Replica]~$ sudo systemctl status mariadb.service
● mariadb.service - MariaDB 10.6.12 database server
     Loaded: loaded (/lib/systemd/system/mariadb.service; enabled; vendor prese>
     Active: active (running) since Fri 2023-12-29 08:09:27 GMT; 5s ago
       Docs: man:mariadbd(8)
             https://mariadb.com/kb/en/library/systemd/
    Process: 3401 ExecStartPre=/usr/bin/install -m 755 -o mysql -g root -d /var>
    Process: 3402 ExecStartPre=/bin/sh -c systemctl unset-environment _WSREP_ST>
    Process: 3404 ExecStartPre=/bin/sh -c [ ! -e /usr/bin/galera_recovery ] && >
    Process: 3475 ExecStartPost=/bin/sh -c systemctl unset-environment _WSREP_S>
    Process: 3477 ExecStartPost=/etc/mysql/debian-start (code=exited, status=0/>
   Main PID: 3463 (mariadbd)
     Status: "Taking your SQL requests now..."
      Tasks: 12 (limit: 14110)
     Memory: 67.3M
        CPU: 252ms
     CGroup: /system.slice/mariadb.service
             └─3463 /usr/sbin/mariadbd
```

4.5.5 Confirm configuration

Confirm the configuration.

```
MariaDB [(none)]> SHOW VARIABLES LIKE 'server_id';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| server_id     | 2     |
+---------------+-------+
1 row in set (0.002 sec)

MariaDB [(none)]> SHOW VARIABLES LIKE 'replicate_do_db%';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| replicate_do_db | Eng   |
+-----------------+-------+
1 row in set (0.002 sec)
```

4.5.6 Getting the current data onto the Replica Database

Connect to the Primary (Master) using a file transfer program like Secure File Transfer Protocol (SFTP) and copy the **Eng.sql** dump file created earlier.

```
[Replica]~$ sftp ada@192.168.0.23
ada@192.168.0.23's password: ada_pass
Connected to 192.168.0.23.
sftp> ls
Eng.sql

sftp> get Eng.sql
Fetching /home/ada/Eng.sql to Eng.sql
/home/ada/Eng.sql                    100% 2807   440.0KB/s   00:00
sftp> quit
```

On the Replica (Slave) Server import the database dump file from the Primary (Master) server.

```
[Replica]~$ ls
Eng.sql

[Replica]~$ sudo mysql –u root Eng < ./Eng.sql
```

Confirm that the tables have been imported.

```
[Replica]~$ mysql –u root –p

MariaDB [(none)]> USE Eng;
Database changed

MariaDB [Eng]> SHOW TABLES;
+----------------+
| Tables_in_Eng |
+----------------+
| EngHobbies    |
| EngProject    |
+----------------+
2 rows in set (0.001 sec)

MariaDB [Eng]> SELECT Firstname, Lastname FROM EngProject;
+-----------+----------+
| Firstname | Lastname |
+-----------+----------+
| Ada       | Lovelace |
+-----------+----------+
1 row in set (0.001 sec)
```

4.5.7 Matching Replica Server with the Primary Server

On the Replica (Slave) Server and define the Primary (Master) using the information retained from from configuring it.

```
MariaDB [(none)]> CHANGE MASTER TO MASTER_HOST='192.168.0.23',
                   -> MASTER_USER='replication_user',
                   -> MASTER_PASSWORD='replication_pass',
                   -> MASTER_LOG_FILE='master1-bin.000002',
                   -> MASTER_LOG_POS=330;
Query OK, 0 rows affected (0.11 sec)
```

Start Replica (Slave).

```
MariaDB [(none)]> START SLAVE;
Query OK, 0 rows affected (0.003 sec)
```

4.5.8 Unlock the Primary Database

Unlock the Primary (Master).

```
[Primary]~$ sudo mysql -u root

MariaDB [(none)]> UNLOCK TABLES;
Query OK, 0 rows affected (0.001 sec)
```

On the replica database, confirm Replica (Slave) is connected to Primary (Master) by reviewing the status.

```
MariaDB [(none)]> SHOW REPLICA STATUS \G;
*************************** 1. row ***************************
                Slave_IO_State: Waiting for master to send event
                   Master_Host: 192.168.0.23
                   Master_User: replication_user
                   Master_Port: 3306
                 Connect_Retry: 60
               Master_Log_File: master1-bin.000003
           Read_Master_Log_Pos: 330
                Relay_Log_File: mysqld-relay-bin.000003
                 Relay_Log_Pos: 631
         Relay_Master_Log_File: master1-bin.000003
              Slave_IO_Running: Yes
             Slave_SQL_Running: Yes
               Replicate_Do_DB: Eng
           Replicate_Ignore_DB:
            Replicate_Do_Table:
        Replicate_Ignore_Table:
       Replicate_Wild_Do_Table:
   Replicate_Wild_Ignore_Table:
                    Last_Errno: 0
                    Last_Error:
                  Skip_Counter: 0
           Exec_Master_Log_Pos: 330
               Relay_Log_Space: 1242
               Until_Condition: None
                Until_Log_File:
```

```
                  Until_Log_Pos: 0
             Master_SSL_Allowed: No
             Master_SSL_CA_File:
             Master_SSL_CA_Path:
                Master_SSL_Cert:
              Master_SSL_Cipher:
                 Master_SSL_Key:
          Seconds_Behind_Master: 0
  Master_SSL_Verify_Server_Cert: No
                  Last_IO_Errno: 0
                  Last_IO_Error:
                 Last_SQL_Errno: 0
                 Last_SQL_Error:
    Replicate_Ignore_Server_Ids:
               Master_Server_Id: 1
                 Master_SSL_Crl:
             Master_SSL_Crlpath:
                     Using_Gtid: No
                    Gtid_IO_Pos:
        Replicate_Do_Domain_Ids:
    Replicate_Ignore_Domain_Ids:
                  Parallel_Mode: optimistic
                      SQL_Delay: 0
            SQL_Remaining_Delay: NULL
        Slave_SQL_Running_State: Slave has read all relay log; waiting
for more updates
               Slave_DDL_Groups: 0
Slave_Non_Transactional_Groups: 0
    Slave_Transactional_Groups: 0
1 row in set (0.001 sec)
```

Confirm the connection from the Replica (Slave) on the Primary (Master)

```
[Primary]~$ ss -at '( sport = :3306 )'
State    Recv-Q  Send-Q     Local Address:Port      Peer Address:Port   Process
LISTEN   0       80              0.0.0.0:mysql           0.0.0.0:*
ESTAB    0       0          192.168.0.23:mysql      192.168.0.40:34364
```

Using the browser tool created, add a user to the Primary (Master) **`EngProject`** table.

Figure 12: Add a new user to the Primary database

Confirm the user has been added to the Primary (Master) **`EngProject`** table and has been replicated to the **`EngProject`** table on the Replica (Slave) database.

```
[Primary]~$ mysql -u enguser -p
Enter password: newengpass

MariaDB [None]> USE Eng;
MariaDB [Eng]> SELECT * FROM EngProject;
+------------+-----------+-----------+----------+-------------------+------------+
| Student_no | Username  | FirstName | LastName | Email             | Role       |
+------------+-----------+-----------+----------+-------------------+------------+
|          1 | alovelace | Ada       | Lovelace | ada@lovelace.com  | Programmer |
|          2 | lmenabrea | Luigi     | Menabrea | luigi@menabrea.it | Politician |
|          3 | equinn    | Edel      | Quinn    | edel@quinn.net    | Nurse      |
|          4 | vcunnane  | Vincent   | Cunnane  | vc@tus.ie         | Professor  |
+------------+-----------+-----------+----------+-------------------+------------+
4 rows in set (0.000 sec)
```

```
[Replica]~$ mysql –u enguser –p
Enter password: newengpass

MariaDB [None]> USE Eng;
MariaDB [Eng]> SELECT * FROM EngProject;
+------------+-----------+-----------+----------+--------------------+------------+
| Student_no | Username  | FirstName | LastName | Email              | Role       |
+------------+-----------+-----------+----------+--------------------+------------+
|          1 | alovelace | Ada       | Lovelace | ada@lovelace.com   | Programmer |
|          2 | lmenabrea | Luigi     | Menabrea | luigi@menabrea.it  | Politician |
|          3 | equinn    | Edel      | Quinn    | edel@quinn.net     | Nurse      |
|          4 | vcunnane  | Vincent   | Cunnane  | vc@tus.ie          | Professor  |
+------------+-----------+-----------+----------+--------------------+------------+
4 rows in set (0.001 sec)
```

4.6 Recovery in case of failure

If for some reason the database replication were to fail or crash. It can be difficult for replication to restart. Stop both databases, clear the logs as shown on the both servers. Restart the Primary (Master) and then restart the Replica (Slave) database.

4.7 Custom tool to view the Replica database

Add a new user to the Replica (Slave) database with access to the **Eng** database.

```
[Replica]~$ mysql –u root

MariaDB [(none)]> CREATE USER 'enguser'@'localhost' IDENTIFIED BY
'engpass';
Query OK, 0 rows affected (0.010 sec)

MariaDB [(none)]> GRANT ALL ON Eng.* TO 'enguser'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

Create a Python Virtual Environment and make it active.

```
~$ python3 –m venv ~/.venv
~$ source ~/.venv/bin/activate
(.venv)[Replica]~$
```

Determine the directory on the path to host **mariadb_conn.py** and copy it in.

```
(.venv)[Replica]~$ python3
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import sys
>>> sys.path
['', '/usr/lib/python310.zip', '/usr/lib/python3.10',
'/usr/lib/python3.10/lib-dynload',
'/home/labuser/.venv/lib/python3.10/site-packages']
>>> quit()
(.venv)[Replica]~$ cp ~/web_repl/tools/mariadb_conn.py
/home/labuser/.venv/lib/python3.10/site-packages
```

Install the **mariadb Connector/Python** module, the **Flask** framework and **yaml**.

```
(.venv)[Replica]~$ python3 -m pip install flask mariadb pyyaml
```

Deactivate from the virtual environment.

```
(.venv)[Replica]~$ deactivate
[Replica]~$
```

Move the **web_repl** to the root of the Apache2 webserver.

```
[Replica]~$ sudo cp -r ~/web_repl /var/www/html/web
[Replica]~$ sudo chown -R www-data: /var/www/html/web
```

Give the website user access to the group hosting the virtual environment and vica versa.

```
~$ sudo usermod -a -G ada www-data
~$ sudo usermod -a -G www-data ada

~$ cat /etc/group | grep ^www
www-data:x:33:ada
~$ cat /etc/group | grep ^ada
ada:x:1000:www-data
```

Create an Apache2 site configuration file by replacing the default file, this points to the Web Server Gateway Interface (WSGI), **`app.wsgi`** and the directory for the Python app.

```
[Replica]~$ cd /etc/apache2/sites-available/

/etc/apache2/sites-available$ sudo mv 000-default.conf 000-default.conf.orig

/etc/apache2/sites-available$ cat <<EOM | sudo tee 000-default.conf

<VirtualHost *:80>
        ServerAdmin webmaster@localhost

        WSGIScriptAlias / /var/www/html/web/app.wsgi
        <Directory /var/www/html/web>
            Order allow,deny
            Allow from all
        </Directory>

        ErrorLog /error.log
        CustomLog /access.log combined

</VirtualHost>
EOM
```

Update the path in the **`app.wsgi`** file to reflect the path noted above.

```
[Replica]~$ sudo vi /var/www/html/web/app.wsgi
import sys

sys.path.insert(0, "/home/labuser/.venv/lib/python3.10/site-packages")
sys.path.insert(0, "/var/www/html/web")

from init import app as application
```

Install the Apache2 WSGI library and, using a2enmod enable the library module within the apache2 configuration.

```
[Replica]~$ sudo apt install apache2-utils libapache2-mod-wsgi-py3
[Replica]~$ sudo a2enmod wsgi
Enabling module wsgi.
```

Enable and start the Apache2 server.

```
[Replica]~$ sudo systemctl enable apache2
[Replica]~$ sudo systemctl start apache2
```

Browse to the IP address of the Replica (Slave) Server. Login with the **`enguser`** credentials and the updated replica can be observed in Figure 13.

`http://<Replica Server URL or address>`

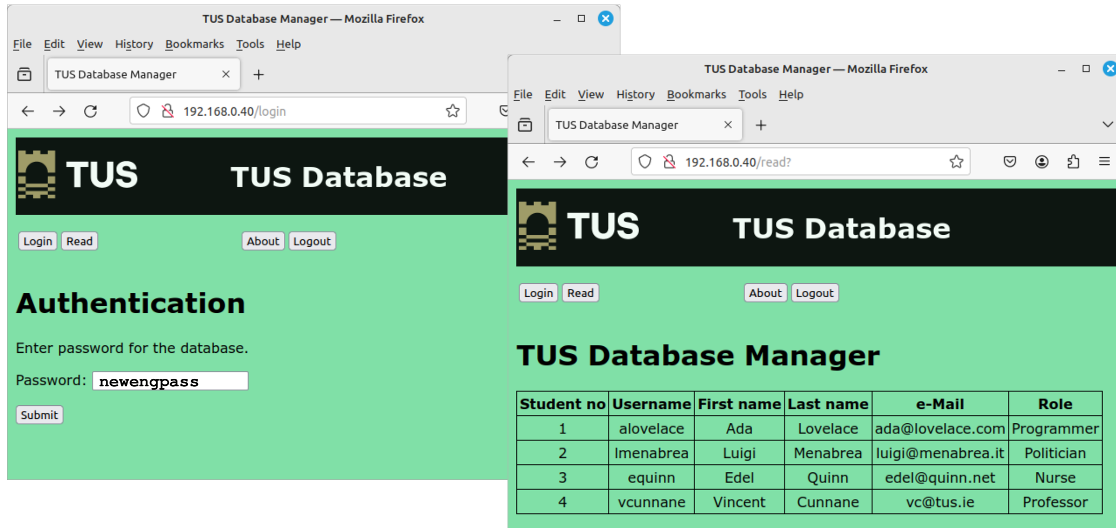


Figure 13: Select data from Replica Database

5. Laboratory #1

5.1 Create a Replica database using MariaDB

- Create a replica address database using MariaDB on a different server that mirrors the database created in the previous topic.
- Establish replication to this database to the other database such that this is the Replica (Slave) and the database from the previous topic is the Primary (Master).
- Document each stage

5.2 Create a custom interface to the database

- Create a web based interface to the database built in 5.1 that has read-only access to the data.
- Document each stage.

Notes:

This page is intentionally blank