**Hands On Lab: MySQL Overview & Architecture**

**LAB 1.1: Installing MySQL on a Linux VM**

**Objective:** Install MySQL Server on a Ubuntu 20.04 LTS VM and verify the installation.

**Prerequisites:**

- Ubuntu 20.04 LTS VM with sudo access
- At least 2GB RAM, 20GB disk space
- Network connectivity

**Step-by-Step Instructions:**

1. **Update System Packages**

   sudo apt update

   sudo apt upgrade -y

2. **Install MySQL Server**

   sudo apt install -y mysql-server mysql-client

3. **Verify Installation**

   mysql --version

   mysqld --version

4. **Start MySQL Service**

   sudo systemctl start mysql

   sudo systemctl enable mysql

   sudo systemctl status mysql

5. **Connect to MySQL**

   sudo mysql -u root

6. **Run Initial Setup Commands**

   SELECT VERSION();

   SHOW VARIABLES LIKE 'datadir';

   SHOW VARIABLES LIKE 'basedir';

   SELECT USER();

7. **Secure MySQL Installation**

   sudo mysql_secure_installation

   - Set root password (if you get error in this step, you can see the resolution in Appendix A)
   - Remove anonymous users
   - Disable remote root login
   - Remove test database
   - Reload privilege tables

8. **Verify Security Settings**

   SELECT host, user FROM mysql.user;

**Hands-on Tasks:**

- Document the MySQL version and installation path
- Take screenshots of key directories: /var/lib/mysql, /etc/mysql
- List all MySQL processes running: ps aux | grep mysql
- Check MySQL port: sudo netstat -tuln | grep 3306

**Troubleshooting:**

- If service fails to start: Check /var/log/mysql/error.log
- Permission issues: Verify /var/lib/mysql ownership: ls -la /var/lib/mysql

**LAB 1.2: Exploring Storage Engines**

**Objective:** Understand different MySQL storage engines and their characteristics.

**Step-by-Step Instructions:**

1. **Connect to MySQL**

   mysql -u root -p

2. **List Available Storage Engines**

   SHOW ENGINES;

   SHOW ENGINES\G

3. **Create Sample Databases**

   CREATE DATABASE storage_engines_demo;

   USE storage_engines_demo;

4. **Create Tables with Different Storage Engines**

a) **InnoDB Table (Default)**

```
CREATE TABLE innodb_table (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100),
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

b) **MyISAM Table**

```
CREATE TABLE myisam_table (
   id INT PRIMARY KEY AUTO_INCREMENT,
   description TEXT
) ENGINE=MyISAM;
```

c) **Memory Table**

```
CREATE TABLE memory_table (
   id INT PRIMARY KEY AUTO_INCREMENT,
   session_data VARCHAR(255)
) ENGINE=MEMORY;
```

d) **Archive Table**

```
CREATE TABLE archive_table (
   id INT PRIMARY KEY AUTO_INCREMENT,
   log_data TEXT
) ENGINE=Archive;
```

5. **Insert Test Data**

```
INSERT INTO innodb_table (name) VALUES
   ('John'), ('Alice'), ('Bob');

INSERT INTO myisam_table (description) VALUES
   ('Test data 1'), ('Test data 2');

INSERT INTO memory_table (session_data) VALUES
   ('Session 1'), ('Session 2');

INSERT INTO archive_table (log_data) VALUES
   ('Log entry 1'), ('Log entry 2');
```

6. **Query Information Schema to View Engine Details**

```
SELECT TABLE_SCHEMA, TABLE_NAME, ENGINE, TABLE_TYPE
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'storage_engines_demo';
```

7. **Compare Engine Characteristics**

```
SELECT
   t.TABLE_NAME,
   t.ENGINE,
   t.ROW_FORMAT,
   t.TABLE_ROWS,
   ROUND(((t.DATA_LENGTH + t.INDEX_LENGTH) / 1024 / 1024), 2) AS size_mb
```

```
FROM INFORMATION_SCHEMA.TABLES t
WHERE t.TABLE_SCHEMA = 'storage_engines_demo';
```

**Hands-on Tasks:**

- Create tables with each engine and observe storage characteristics
- Insert sample data (100+ rows) into each table
- Compare table sizes using INFORMATION_SCHEMA
- Document the SHOW CREATE TABLE output for each engine
- Identify which engines support transactions

**Solution to Hands-on Tasks:**

Create one table per engine, insert sample data, then compare their on-disk size and behavior using SHOW TABLE STATUS or INFORMATION_SCHEMA.TABLES.

**Step 1: Create demo tables**

Use a test database so the objects are isolated.

```
CREATE DATABASE IF NOT EXISTS engine_demo;
USE engine_demo;
```

Create the same logical table with different engines.

```
CREATE TABLE customer_innodb (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100),
  email VARCHAR(200),
  created_at DATETIME
) ENGINE = InnoDB;

CREATE TABLE customer_myisam (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100),
  email VARCHAR(200),
```

```
  created_at DATETIME
) ENGINE = MyISAM;

CREATE TABLE customer_memory (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100),
  email VARCHAR(200),
  created_at DATETIME
) ENGINE = MEMORY;

CREATE TABLE customer_archive (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(100),
  email VARCHAR(200),
  created_at DATETIME
) ENGINE = ARCHIVE;
```

**Step 2: Insert identical data**

Populate each table with the same rows to make comparisons meaningful.

```
INSERT INTO customer_innodb (name, email, created_at)
SELECT CONCAT('User', n),
      CONCAT('user', n, '@example.com'),
      NOW()
FROM (
  SELECT 1 AS n UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5
  UNION ALL SELECT 6 UNION ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9 UNION ALL SELECT 10
) AS t;
```

```
INSERT INTO customer_myisam  (name, email, created_at)
SELECT name, email, created_at FROM customer_innodb;

INSERT INTO customer_memory  (name, email, created_at)
SELECT name, email, created_at FROM customer_innodb;

INSERT INTO customer_archive (name, email, created_at)
SELECT name, email, created_at FROM customer_innodb;
```

Use one base INSERT and then copy that data into the other engine tables so all four have identical rows.

1. Insert into the InnoDB table

```
USE engine_demo;

INSERT INTO customer_innodb (name, email, created_at)
VALUES
 ('Alice',  'alice@example.com',  NOW()),
 ('Bob',    'bob@example.com',    NOW()),
 ('Charlie','charlie@example.com',NOW()),
 ('David',  'david@example.com',  NOW()),
 ('Emma',   'emma@example.com',   NOW()),
 ('Frank',  'frank@example.com',  NOW()),
 ('Grace',  'grace@example.com',  NOW()),
 ('Helen',  'helen@example.com',  NOW()),
 ('Ivan',   'ivan@example.com',   NOW()),
 ('Julia',  'julia@example.com',  NOW());
```

This uses a single INSERT with multiple value lists, which is the recommended pattern for inserting many rows efficiently.

2. Copy the same data to other engines

Now clone rows from customer_innodb into the MyISAM, MEMORY, and ARCHIVE tables with INSERT … SELECT.

```
INSERT INTO customer_myisam (name, email, created_at)

SELECT name, email, created_at

FROM customer_innodb;


INSERT INTO customer_memory (name, email, created_at)

SELECT name, email, created_at

FROM customer_innodb;


INSERT INTO customer_archive (name, email, created_at)

SELECT name, email, created_at

FROM customer_innodb;
```

INSERT INTO … SELECT copies column values directly from the source table and is a standard way to bulk-populate another table with the same structure.

3. Quick verification

```
SELECT 'innodb'  AS engine_name, COUNT(*) AS rows FROM customer_innodb

UNION ALL

SELECT 'myisam', COUNT(*) FROM customer_myisam

UNION ALL

SELECT 'memory', COUNT(*) FROM customer_memory

UNION ALL

SELECT 'archive',COUNT(*) FROM customer_archive;
```

All four tables should show the same row count (10 rows in this example), ensuring a fair comparison of storage characteristics.

For a stronger effect, scale this up to a few thousand rows using a numbers helper or stored procedure.

**Step 3: Observe storage with SHOW TABLE STATUS**

Check per-table size and characteristics.

SHOW TABLE STATUS FROM engine_demo LIKE 'customer_%'\G

Key columns to discuss with participants:

- Engine: Confirms storage engine used.
- Rows: Estimated number of rows (MyISAM is often exact; InnoDB is approximate).
- Data_length:
    - MyISAM: size of the data file in bytes.
    - InnoDB: approximate space allocated for the clustered index (data) in bytes.
- Index_length:
    - MyISAM: size of the index file.
    - InnoDB: approximate space for secondary indexes.
- Row_format, Data_free: Observe fragmentation and free space especially for InnoDB/MyISAM.

**Prompt questions for learners:**

- Compare Data_length and Index_length between InnoDB and MyISAM for the same row count.
- Check how MEMORY table reports size (entirely in RAM, still exposed via metadata).
- Note ARCHIVE's compact storage and append-only nature (no index_length by default).

**Step 4: Observe via INFORMATION_SCHEMA.TABLES**

Use a single query to get sizes in MB for each engine.

```
SELECT
  table_name,
  engine,
```

```
  table_rows,
  ROUND(data_length / 1024 / 1024, 2) AS data_mb,
  ROUND(index_length / 1024 / 1024, 2) AS index_mb,
  ROUND((data_length + index_length) / 1024 / 1024, 2) AS total_mb
FROM information_schema.tables
WHERE table_schema = 'engine_demo'
  AND table_name LIKE 'customer_%'
ORDER BY total_mb DESC;
```

Discuss with the group:

- Which engine uses more total space for the same data set and why (e.g., InnoDB overhead vs MyISAM)?
- MEMORY table will show size but is volatile and lost on restart.
- ARCHIVE table typically has small index_length but may have more compressed data_length for large, append-only workloads.

**Step 5: Optional extensions**

To deepen the lab:

- Add indexes only on some engines and re-run the size query to show impact of INDEX_LENGTH.
- Delete a large percentage of rows from the InnoDB/MyISAM tables, then re-check Data_length and Data_free to show fragmentation and need for OPTIMIZE TABLE or ALTER TABLE for space reclamation.
- Restart MySQL and show that MEMORY table data is gone while others persist, tying storage characteristics to durability guarantees.

**Assessment Questions:**

1. Which engine is the default in MySQL 8.0?
2. Which engine stores data in memory?
3. Which engine is read-only but compresses well?
4. Which engine supports foreign keys?

APPENDIX - A

1. ... Failed! Error: SET PASSWORD has no significance for user 'root'@'localhost' as the authentication method used doesn't store authentication data in the MySQL server. Please consider using ALTER USER instead if you want to change authentication parameters.

   This error is **expected behavior in newer MySQL versions** and is not a failure in the usual sense 👍
   Let me explain **why it happens** and **how to fix it correctly**.

---

### 🔍 Why this error occurs

Your MySQL **root user** is authenticated using an **external authentication plugin**, most commonly:

- auth_socket (Linux / Ubuntu)
- caching_sha2_password managed externally

When this is the case:

- SET PASSWORD **does nothing**
- MySQL does **not store the password internally**
- Hence the warning/error:

*SET PASSWORD has no significance… authentication method doesn't store authentication data*

This is very common in:

- MySQL 8.x
- Ubuntu/Debian installations
- Docker-based MySQL setups

---

### ✅ Correct ways to resolve it

### ✅ Option 1: Change password using ALTER USER (Recommended)

If you want MySQL to manage the password internally:

ALTER USER 'root'@'localhost'

IDENTIFIED WITH mysql_native_password

BY 'NewStrongPassword!';

Then refresh privileges:

FLUSH PRIVILEGES;

✓ This **switches authentication** to a password-based mechanism
✓ Fixes the error permanently

---

✅ **Option 2: Keep auth_socket (Passwordless root login)**

If you want root to log in **only via OS user** (common in servers):

sudo mysql

Check authentication method:

SELECT user, host, plugin FROM mysql.user WHERE user='root';

If plugin = auth_socket, then:

- Password changes are irrelevant
- This is **secure and valid**
- No action needed

---

✅ **Option 3: Create a separate admin user (Best practice)**

**Recommended for production systems**

CREATE USER 'dbadmin'@'localhost'

IDENTIFIED BY 'StrongPassword!';

GRANT ALL PRIVILEGES ON *.* TO 'dbadmin'@'localhost'

WITH GRANT OPTION;

FLUSH PRIVILEGES;

Use dbadmin instead of root.

---

### 🔐 Which option should you choose?

| Scenario | Best Option |
| --- | --- |
| Local development | Option 1 |
| Production server | Option 3 |
| Linux-only admin access | Option 2 |
| Docker / CI pipelines | Option 1 or 3 |

---

### 🧪 Verify the fix

mysql -u root -p

or

SHOW GRANTS FOR 'root'@'localhost';

---

### 🚨 Important Note

**⚠️ Avoid using root with passwords in production**
Create dedicated users with least privileges instead.