# MySQL

- MySQL Optimisation, Performance, and Logging
- Corruption and Repairs
- User and Database Management
- Binary Logging
- Database Monitoring (New Relic)
- MySQL Encryption
- MySQL Remote Access
- Storage Engines
- MySQL Replication
- XtraBackup
- MySQL Backups - mysqldump and xtrabackup

# MySQL Optimisation, Performance, and Logging

==========

# MySQL Optimisation

==========

## MySQL Tuner Script

```
wget -O mysqltuner.pl mysqltuner.pl && perl mysqltuner.pl
```

---

## Check MySQL default config files

```
/usr/sbin/mysqld --verbose --help | grep -A 1 "Default options"
```

---

## MySQL Variables

| | |
|---|---|
| innodb_buffer_pool_size | Specifies the size of the memory buffer used by InnoDB to cache data and indexes. Setting this appropriately can significantly improve read/write performance for InnoDB tables.<br><br>***Recommendation:*** *For dedicated MySQL servers, set to 70-80% of available memory.* |
| query_cache_size | Controls the size of the query cache, which stores results of SELECT queries to speed up subsequent identical queries. Effective use depends on query patterns and workload.<br><br>***Recommendation:*** *Enable and configure it based on the workload if beneficial; otherwise, keep it disabled.* |

| | |
|---|---|
| key_buffer_size | Sets the size of the buffer used for index blocks for MyISAM tables. Adjusting this can enhance performance for MyISAM-based applications.<br><br>***Recommendation:*** *Set according to the size of your MyISAM indexes; for mixed storage engines, allocate more memory to InnoDB buffer pool.* |
| max_connections | Limits the number of simultaneous client connections allowed to the MySQL server. Properly setting this prevents resource exhaustion and maintains server stability. |
| tmp_table_size =<br>max_heap_table_size = | Define the maximum size of temporary tables created in memory. Optimizing these can reduce disk I/O and improve query performance. |
| sort_buffer_size=256K | Buffer size for sorts. |
| max_connections=151 | Maximum number of simultaneous client connections.<br><br>***Recommendation:*** *Increase based on expected load and server capacity.* |
| innodb_log_file_size | Size of each log file in the log group.<br><br>***Recommendation:*** *Increase to improve performance, particularly for write-heavy applications.* |
| innodb_log_buffer_size | Size of the buffer for log data before it is written to disk.<br><br>***Recommendation:*** *Increase for high transaction throughput to reduce disk I/O.* |

# Logging

| | |
|---|---|
| slow_query_log | Enables logging of queries that exceed a certain execution time.<br><br>***Recommendation:*** *Enable and use for identifying slow queries; configure* `long_query_time` *to set the threshold.* |
| log_error = /var/log/mysql.log | Path to the error log file. |
| general_log=/var/log/mysql.log | Enables logging of all queries.<br><br>***Recommendation:*** *Use with caution due to potential performance impact and disk consumption; useful for debugging.* |

==========

## MyISAM v InnoDB

**InnoDB has row-level locking.** MyISAM only has full table-level locking.

- **InnoDB:**
  - Default storage engine in MySQL.
  - Supports ACID-compliant transactions.
  - Provides row-level locking and foreign key constraints.
  - Uses a clustered index for primary key storage.
- **MyISAM:**
  - Older storage engine, primarily used before InnoDB became the default.
  - Provides table-level locking.
  - Faster for read-heavy operations.
  - Does not support transactions or foreign keys.

Check table engine:

```
mysql -e "SELECT TABLE_SCHEMA as DbName ,TABLE_NAME as TableName ,ENGINE as Engine FROM information_schema.TABLES WHERE ENGINE='MyISAM' AND TABLE_SCHEMA NOT IN('mysql','information_schema','performance_schema');"
```

Generate commands to swap table engine from MyISAM to InnoDB

```
mysql -e "SELECT CONCAT('ALTER TABLE \`', TABLE_SCHEMA,'\`.\`',TABLE_NAME, '\` ENGINE = InnoDB;') FROM information_schema.TABLES WHERE ENGINE='MyISAM' AND TABLE_SCHEMA NOT IN('mysql','information_schema','performance_schema');"
```

==========

# Deadlocks

A deadlock is a situation where two or more transactions are unable to proceed because each one is waiting for a lock held by the other transaction. This creates a cycle of dependencies that cannot be resolved without external intervention.

---

## Check for deadlocks:

```
SHOW ENGINE INNODB STATUS;
```

```
SHOW STATUS LIKE 'Table_locks_%';
```

- **Table_locks_immediate**: The number of times a table lock was acquired immediately.
- **Table_locks_waited**: The number of times a table lock couldn't be acquired immediately and had to wait.

---

## Deadlock logging

This will cause MySQL to log all deadlock errors to the error log, which can then be examined for more details.

```
SET GLOBAL innodb_print_all_deadlocks = 1;
```

---

MySQL Monitoring software such as NewRelic can provide further information on deadlocks and general MySQL performance issues.

## Resolving Deadlocks

Once identified, you can resolve deadlocks by:

- **Rewriting Queries**: Optimize queries to reduce lock contention.
- **Using Consistent Locking Order**: Ensure all transactions acquire locks in a consistent order.
- **Adding Indexes**: Improve query performance and reduce the duration of locks.
- **Reducing Transaction Size**: Break large transactions into smaller ones to reduce lock time.
- **Implementing Retries**: Add retry logic in your application to handle deadlocks gracefully.

---

# Corruption and Repairs

==========

# MySQL Check

```
mysqlcheck -options database
```

## MySQL Check Options:

| | |
|---|---|
| -c | Checks tables for errors. It performs a `CHECK TABLE` operation on each table in the specified databases. |
| -r | Attempts to repair corrupted tables. It performs a `REPAIR TABLE` operation on each corrupted table. |
| -a | Analyzes tables for optimal performance. It performs an `ANALYZE TABLE` operation on each table. |
| -o | Optimizes tables to reduce fragmentation and reclaim unused space. It performs an `OPTIMIZE TABLE` operation on each table. |
| --databases db1 db2 | Checks and repairs tables in multiple databases (`db1` and `db2`). |
| -A | Checks and repairs tables in all databases on the MySQL server. |

---

## When to use mysqlcheck -r

- **Table Corruption**: Use `mysqlcheck -r` (or `mysqlcheck --repair`) when you suspect or know that one or more tables in your MySQL database are corrupted or have crashed.
- **Error Messages**: If MySQL reports errors such as "table is marked as crashed" or "table needs repair," `mysqlcheck -r` is a suitable approach to attempt repair.

## Best Practices and Considerations

1. **Backup**: Always make a backup of your databases before attempting any repairs. While `mysqlcheck -r` is generally safe, there is a small risk of data loss if the repair process encounters unexpected issues.

2. **Table Locking**: During repair, `mysqlcheck` will lock tables to ensure data consistency. Depending on the size and activity of your database, this can cause downtime or impact performance.

==========
==========

# User and Database Management

Command syntax will vary between MySQL Versions

==========

# User Management

## Show all users and hosts

```
SELECT user, host FROM mysql. user;
```

## Add User

```
CREATE USER 'name'@'localhost' IDENTIFIED BY 'password.';
```

## Granting Privileges

Available privileges:

| | |
|---|---|
| CREATE | Allows the user to create new databases and tables. |
| ALTER | Allows the user to alter (modify) existing tables. |
| DROP | Allows the user to drop (delete) databases and tables. |
| INSERT | Allows the user to insert new rows into tables. |
| UPDATE | Allows the user to update existing rows in tables. |
| DELETE | Allows the user to delete rows from tables. |
| SELECT | Allows the user to read (select) data from tables. |
| REFERENCES | Allows the user to create foreign key constraints when defining tables. |
| RELOAD | Allows the user to execute the `FLUSH` statement, which reloads various server configurations and clears or refreshes caches. |

Grant specific privilege ( or multiple comma separated):

```
GRANT PRIVILEGETYPE ON database.table TO 'username'@'host';
FLUSH PRIVILEGES;
```

Grant all:

```
GRANT ALL ON database.table TO 'username'@'host';
FLUSH PRIVILEGES;
```

### Delete a user

```
DROP USER 'name'@'host';
```

### Change user password

```
ALTER USER 'user'@'host' IDENTIFIED BY 'NewPassword';
FLUSH PRIVILEGES;
```

==========

# Database Management

### Create a database

```
CREATE DATABASE name;
```

---

### Delete a database

```
DROP DATABASE name;
```

==========

# Binary Logging

==========

# Binary Logging

==========

# What is Binary Logging?

Binary logging in MySQL is a crucial feature for data replication, backup, and recovery. It logs all changes made to the database, such as updates, deletions, and insertions, in a binary format.

## Purpose of Binary Logging

- **Replication**: Binary logs are used to replicate data from a primary server to one or more replica servers. Changes logged on the primary server are applied to the replicas to keep them synchronized.
- **Point-in-Time Recovery (PITR)**: In case of data corruption or accidental data loss, binary logs allow you to restore a backup and replay the transactions recorded in the binary logs to recover the database to a specific point in time.
- **Audit Trails**: They can be used to audit and review changes made to the database over time.

### How Binary Logging Works

- **Binary Log Files**: The changes are recorded in binary log files (with extensions like `.000001`, `.000002`, etc.) which are stored in the MySQL data directory.
- **Log Events**: Each event in the binary log represents a single database change such as a query or a transaction. These events are recorded in the order they were executed.
- **Binary Log Index File**: MySQL maintains an index file (typically `mysql-bin.index`) that lists all the binary log files and their sequence.

==========

# Binary Logging Configuration

---

---

# Enabling Binary Logging

To enable binary logging, you need to configure the MySQL server with the `log_bin` parameter in the `my.cnf` configuration file:

```
[mysqld]
log_bin = /var/log/mysql/mysql-bin.log
```

---

---

# Binary Logging Options

Bin log options that can also be set in the my.cnf file:

| | |
|---|---|
| expire_logs_days = 7 | Defines how many days to retain binary log files before automatic deletion. |
| binlog_format = format<br>format:<br>ROW<br>STATEMENT<br>MIXED | Determines the format of the binary logs. The available formats are:<br>ROW:Logs the actual changes at the row level.<br>STATEMENT: Logs each SQL statement that modifies data.<br>MIXED: Uses a combination of statement and row formats. |
| max_binlog_size = 100M | Sets the maximum size of a binary log file before a new file is created. |

---

---

# Binary Log Management

## Viewing Binary Logs

Use the `SHOW BINARY LOGS` or `SHOW MASTER LOGS` command to list the binary log files.

```
SHOW BINARY LOGS;
```

## Examining Binary Log Content

Use the `mysqlbinlog` utility to examine the content of binary log files

```
mysqlbinlog /var/log/mysql/mysql-bin.000001
```

## Purging Binary Logs

To manually delete old binary logs and free up space, use the `PURGE BINARY LOGS` statement.

```
PURGE BINARY LOGS TO 'mysql-bin.000010';
```

Or, you can purge logs older than a specific date:

```
PURGE BINARY LOGS BEFORE '2024-01-01 00:00:00';
```

==========

# Data Recovery From Binary Logs

---

> Binary Logs are just transactional changes made to a database. They don't include a full copy of the database itself. In order to replay binary logs, you'll need a backup copy of the database to replay onto. See mysql backups.

**1. Identify the Binary Logs to Use**

Determine the range of binary logs that contain the transactions you need to replay. You can list all binary logs with:

```
SHOW BINARY LOGS;
```

**2. Obtain a Backup**

Obtain the newest backup/dump of the database prior to the time that we need to recover to.

**3. Import the backup into MySQL**

```
mysql -u root -p < /path/to/backup.sql
```

**4. Apply Binary Logs**

A. Identify the start and end points within the binary logs. You might need to apply all transactions or up to a specific point in time or transaction.

```
mysqlbinlog --start-datetime="2024-06-28 10:00:00" --stop-datetime="2024-06-29 14:00:00"
/path/to/mysql-bin.000001 /path/to/mysql-bin.000002 | mysql -u root -p
```

==========
==========

# Database Monitoring (New Relic)

==========

## What is New Relic?

In the context of MySQL, New Relic provides monitoring and performance management capabilities through its Application Performance Monitoring (APM) and Infrastructure Monitoring features.

- **Monitor Query Performance**: Track the execution time, throughput, and response times of MySQL queries. Identify slow queries and potential bottlenecks affecting application performance.
- **Transaction Tracing**: Trace individual transactions from your application through to MySQL database interactions. This helps pinpoint which parts of your application are causing database load or delays.
- **Database Errors**: Capture and alert on MySQL database errors that occur during application operations.

## Benefits of Using New Relic with MySQL

- **Performance Optimization**: Identify and resolve MySQL query performance issues that impact application responsiveness and user experience.
- **Troubleshooting**: Quickly diagnose and troubleshoot database-related problems such as slow queries or database errors affecting application functionality.
- **Capacity Planning**: Forecast MySQL server resource requirements based on historical performance data and trends monitored by New Relic.
- **End-to-End Visibility**: Gain comprehensive insights into the entire application stack, from front-end application performance to backend MySQL database operations.

---

https://www.ans.co.uk/docs/monitoring/installnewrelic/

# MySQL Encryption

## Encryption Types in MySQL

1. **Data-at-Rest Encryption**:
   - Tablespace Encryption: Encrypts the entire tablespace, including the InnoDB tables.
   - Column-Level Encryption: Encrypts specific columns in a table.
2. **Data-in-Transit Encryption**:
   - SSL/TLS Encryption: Encrypts data transmitted between MySQL clients and servers using SSL/TLS.

==========

## Data-at-Rest Encryption

> MyISAM does not support encryption natively. Tables will need to be converted to InnoDB before encryption is implemented.

### 1. Enable the Keyring Plugin

The keyring plugin is used to store and manage encryption keys securely within MySQL.

**Install the Keyring Plugin**

- If you are using MySQL 5.7 or later, the keyring_file plugin is included by default.
- Add the following lines to the MySQL configuration file

```
[mysqld]
early-plugin-load = keyring_file.so
keyring_file_data = /var/lib/mysql-keyring/keyring
```

**Create the directory for the keyring file if it doesn't exist**

```
sudo mkdir /var/lib/mysql-keyring
sudo chown mysql:mysql /var/lib/mysql-keyring
```

Restart the MySQL server to load the plugin

**Verify the Keyring Plugin is Enabled:**

```
SHOW PLUGINS;
```

## 2. Enable InnoDB Tablespace Encryption

**Enable InnoDB File-Per-Table**:

Ensure that `innodb_file_per_table` is enabled, which is the default setting in MySQL 5.6 and later.

```
[mysqld]
innodb_file_per_table = 1
```

**Enable InnoDB Encryption**:

```
[mysqld]
innodb_encrypt_tables = ON
innodb_encrypt_log = ON
innodb_encryption_threads = 4
```

**Restart MySQL.**

```
systemctl restart mysql
```

## 3. Encrypt Existing Tables

**Encrypt a Specific Table**

```
ALTER TABLE mytable ENCRYPTION='Y';
```

## 4. Verify Encryption

**Check Encryption Status**

You can verify if a table is encrypted by querying the `information_schema.tables` table:

```
SELECT table_schema, table_name, create_options
FROM information_schema.tables
WHERE create_options LIKE '%ENCRYPTION="Y"%';
```

---

# Binary Log Encryption

You can replay unencrypted binary logs onto encrypted tables.

## Enable Binary Log Encryption

Add the following configuration to your `my.cnf`

```
[mysqld]
binlog_encryption = ON
```

## Verify Binary Log Encryption

```
SHOW VARIABLES LIKE 'binlog_encryption';
```

---

# Replaying encrypted binary logs

Replaying encrypted binary logs involves ensuring that the encrypted logs are decrypted and applied correctly on the MySQL server.

**Use mysqlbinlog to Read Encrypted Binary Logs**

The `mysqlbinlog` utility will handle the decryption transparently if the keyring plugin is properly configured.

```
mysqlbinlog /path/to/binlog.000001 | mysql -u username -p
```

The process for replaying binary logs whether encrypted or not is largely the same, providing that the keyring plugin is enabled. See here for more info.

==========

# Data-in-Transit Encryption

Data-in-transit encryption refers to the protection of data as it moves between systems, such as between a client and a server, or between servers. This type of encryption ensures that data remains confidential and integral during transmission, preventing unauthorized access and tampering.

Data-in-transit encryption typically uses Transport Layer Security (TLS) or its predecessor, Secure Sockets Layer (SSL), to secure communications.

# Implementing Data-in-Transit Encryption in MySQL

**1. Generate or Obtain Certificates and Keys**:

Generate self-signed certificates using OpenSSL or obtain them from a trusted CA. Certificate needs to cover the MySQL server hostname.

Example command to generate a self-signed certificate using OpenSSL:

```
openssl req -newkey rsa:2048 -nodes -keyout server-key.pem -x509 -days 365 -out server-cert.pem
```

**2. Configure MySQL Server**

Edit the MySQL configuration file (`my.cnf` or `my.ini`) to include the paths to the certificates and keys.

```
[mysqld]
ssl-ca = /path/to/ca-cert.pem
ssl-cert = /path/to/server-cert.pem
ssl-key = /path/to/server-key.pem
```

> To force the use of SSL when connecting to a MySQL server, you can add the below to the my.cnf:
> `require_secure_transport = ON`

**3. Restart MySQL Server**

```
systemctl restart mysql
```

**4. Verify SSL/TLS Configuration**

Verify that SSL/TLS is enabled on the server.

```
SHOW VARIABLES LIKE '%ssl%';
```

==========

# MySQL Remote Access

==========

To configure remote access for MySQL, you need to ensure that MySQL is configured to accept remote connections and that your firewall and MySQL user permissions are set up correctly.

## Configure MySQL Server to Allow Remote Connections

### 1. Bind Address

The `bind-address` option in the MySQL configuration file specifies which network interfaces MySQL will listen on for incoming connections. By default, this is set to bind to 127.0.0.1 (localhost), meaning that only local connections can be made.

The bind address can be set to 0.0.0.0 to listen on all interfaces.

```
bind-address = 0.0.0.0
```

You can also set the bind address to the IP of a specific interface, should you wish to limit where MySQL traffic comes from.

### 2. Ensure port 3306 is open inbound on firewall

Check for limitations on port 3306 inbound traffic on local/hardware firewalls.

### 3. MySQL User Configuration

**Create new user:**

You can either create a new user with the desired host or alter an existing user's host if necessary. However, creating a new entry is often the safest and most organized method.

**Create a New Entry for the User**:

```
CREATE USER 'example_user'@'203.0.113.1' IDENTIFIED BY 'user_password';
GRANT ALL PRIVILEGES ON database_name.* TO 'example_user'@'203.0.113.1';
FLUSH PRIVILEGES;
```

**Alter existing user:**

```
RENAME USER 'example_user'@'localhost' TO 'example_user'@'203.0.113.1';
FLUSH PRIVILEGES;
```

==========
==========

# Storage Engines

MySQL supports multiple storage engines, with InnoDB and MyISAM being the most commonly used.

---

## MyISAM

1. **Table-Level Locking:**
   - **Description:**
     - MyISAM uses table-level locking, which means that the entire table is locked for the duration of a read or write operation.
   - **Types of Locks:**
     - **Read Lock (Shared Lock):**
       - Multiple read operations can occur simultaneously.
       - Write operations are blocked until all read locks are released.
     - **Write Lock (Exclusive Lock):**
       - Only one write operation can occur at a time.
       - All other read and write operations are blocked until the write lock is released.
   - **Implications:**
     - **Advantages:**
       - Simpler to implement and manage.
       - Can be faster for read-heavy workloads with fewer write operations.
     - **Disadvantages:**
       - Poor concurrency for write-heavy workloads.
       - Write operations can block reads, leading to potential bottlenecks.

---

## InnoDB

1. **Row-Level Locking:**
   - **Description:**
     - InnoDB uses row-level locking, which means that only the specific rows involved in a transaction are locked.
   - **Types of Locks:**
     - **Shared Lock (S Lock):**
       - Allows multiple transactions to read the same rows simultaneously.
       - Prevents other transactions from writing to the locked rows.
     - **Exclusive Lock (X Lock):**

        - Prevents other transactions from reading or writing to the locked rows.
    - **Intent Locks:**
        - **Intent Shared Lock (IS Lock):**
            - Indicates that a transaction intends to read rows in a table.
            - Compatible with other IS locks and S locks, but not with IX or X locks.
        - **Intent Exclusive Lock (IX Lock):**
            - Indicates that a transaction intends to write rows in a table.
            - Compatible with other IX locks, but not with S or X locks.
- **Implications:**
    - **Advantages:**
        - Higher concurrency as only specific rows are locked.
        - Better performance for mixed read/write workloads.
        - Supports transactions and ACID compliance.
    - **Disadvantages:**
        - More complex locking mechanism.
        - Slightly higher overhead compared to table-level locking.

==========
==========

# MySQL Replication

MySQL replication is a process that allows data from one MySQL database server (the primary or source server) to be copied automatically to one or more MySQL database servers (replica or slave servers). Replication is used to improve data availability, load balancing, and for backup and failover purposes.

---

# Types of MySQL Replication

**Asynchronous Replication:**

- Description**:** The primary server sends events to the replica, but does not wait for the replica to acknowledge receipt.
- Use Cases: High performance, low latency applications where some delay in data consistency is acceptable.

**Semi-Synchronous Replication:**

- Description: The primary server waits for at least one replica to acknowledge that it has received the events, but not necessarily that it has applied them.
- Use Cases: Balances performance with increased data safety compared to asynchronous replication.

**Synchronous Replication (Group Replication):**

- Description: All replicas must acknowledge receipt and application of events before the primary server considers the transaction complete.
- Use Cases: High availability and data consistency scenarios where strong data guarantees are required.

==========

# Replication Setups

> See here for more info, and the ANS KB for setup. The below example is from my own solution.

# Master - Master

In a Master-Master replication setup, two MySQL servers are configured to replicate data to each other. Both servers can accept write operations, and changes are propagated bidirectionally.

Both servers act as master and slave to each other, allowing writes on both nodes.

If one server fails, the other can continue to serve both read and write requests.

Data is duplicated on both servers, providing a backup in case one fails.

**Advantages:**

1. Increased Availability:
   - Both servers can handle writes, providing high availability.
2. Load Balancing:
   - Distribute read and write operations across both servers.
3. Redundancy:
   - Data is replicated on both servers, providing a failover option.

**Disadvantages:**

1. Complexity:
   - More complex to set up and maintain compared to Master-Slave.
2. Conflict Resolution:
   - Potential for conflicts if the same data is modified on both servers simultaneously.
3. Data Consistency:
   - Possible data inconsistency issues if replication lag occurs.

---

# Example Configuration

Master - Slave (asynchronous) configuration.

## Configure Server A (master)

### 1. Set the `my.cnf` file for Server A

```
[mysqld]
server_id = 1
log_bin = /var/log/mysql/mysql-bin.log
```

### 2. Create a user for replication on Server A

```
CREATE USER 'replication'@'%' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%';
FLUSH PRIVILEGES;
```

> As you can see here `CREATE USER 'replication'@'%' IDENTIFIED BY 'password';`, I've set the host as wildcard (%), You'll ideally want to create 2 users here, one with the internal IP of Server A, and one with the internal IP of server B (internal IF on same network). For example:
>
> ```
> CREATE USER 'replication'@'server_a_ip' IDENTIFIED BY 'password';
> CREATE USER 'replication'@'server_b_ip' IDENTIFIED BY 'password';
> ```

### 3. Lock all tables to get MySQL into a consistent state on server A

```
FLUSH TABLES WITH READ LOCK;
SHOW MASTER STATUS;
```

Note down the `File` and `Position` values. You will use these in Server A's configuration.

### 4. Create a database dump

For replication to work, we'll first need to import a dump of the locked databases from the master onto the slave.

```
mysqldump -u root -p --all-databases --master-data > dbdump.sql
```

| | |
|---|---|
| --master-data | **Adds `CHANGE MASTER TO` Statement:** At the beginning of the dump file (`dbdump.sql` in your case), `--master-data` adds a comment containing the `CHANGE MASTER TO` statement. This statement includes the following information:<br>• `MASTER_LOG_FILE`: The name of the binary log file that was being written to when the dump was initiated.<br>• `MASTER_LOG_POS`: The position within that binary log file where the dump started.<br>• This information helps in setting up a slave server to start replication from the exact point where the dump was taken, ensuring consistency between the master and the slave. |

## Configure Server B (Slave)

### 1. Set the my.cnf file for Server B

```
[mysqld]
server_id = 2
log_bin = /var/log/mysql/mysql-bin.log
```

**2. Import database dump from Server A**

```
mysql -u root -p < dbdump.sql
```

**3. Configure replication on Server B using the below MySQL command:**

```
CHANGE MASTER TO
    MASTER_HOST='server_A_ip',
    MASTER_USER='repl',
    MASTER_PASSWORD='password',
    MASTER_LOG_FILE='mysql-bin.000001',  -- use the File value from Server A
    MASTER_LOG_POS=120;                  -- use the Position value from Server A
START SLAVE;
```

**3. Verify Replication Status:**

Ensure that both servers are replication to each other without errors. You should see `Slave_IO_Running` and `Slave_SQL_Running` set to `Yes` on both servers.

```
SHOW SLAVE STATUS\G;
```

- Ensure `Slave_IO_Running` and `Slave_SQL_Running` are both set to `Yes`.
- Check for errors in the `Last_IO_Error` and `Last_SQL_Error` fields.

4. Unlock the tables on both servers

```
UNLOCK TABLES;
```

> Ensure to test once set up.

---

# Master - Slave

In a Master-Slave replication setup, one MySQL server (the master) is responsible for all write operations, and one or more slave servers replicate the data from the master. Slave servers are typically read-only.

Data flows from the master to the slave(s) only.

Slaves can handle read operations, offloading the master.

**Advantages:**

1. **Simplicity:**
   - Easier to set up and maintain compared to Master-Master replication.
2. **Read Scalability:**
   - Offload read operations to slaves, improving performance.
3. **Data Safety:**
   - Slaves can be used for backups and disaster recovery without affecting the master.

**Disadvantages:**

1. **Single Point of Failure:**
   - The master is a single point of failure for write operations.
2. **Replication Lag:**
   - Potential for replication lag, which can lead to stale reads on slaves.
3. **Limited Write Scalability:**
   - All writes are handled by the master, which can become a bottleneck.

==========
==========

# XtraBackup

# MySQL Backups - mysqldump and xtrabackup

==========

# Percona XtraBackup

Percona XtraBackup is a widely used open-source tool for MySQL and MariaDB database backups. It is developed by Percona and is specifically designed to perform hot backups of MySQL databases without interrupting database operations.

---

## Key Features of XtraBackup:

| | |
|---|---|
| Hot Backups | XtraBackup allows you to take backups while the database is running and serving requests, minimizing downtime and maintaining data availability. |
| Consistency | It ensures that backups are consistent, capturing all changes up to the point when the backup process begins. |
| Incremental Backups | XtraBackup supports incremental backups, which only include changes made since the last backup. This feature reduces backup time and storage requirements. |
| Compressed Backups | Backups can be compressed to save disk space and reduce storage costs. |
| Encrypted Backups | XtraBackup supports encryption, which helps secure backup data both at rest and in transit. |
| Streaming Backups | The tool can stream backups to another server, useful for remote storage or cloud integration. |

---

## Percona XtraBackup vs mysqldump & bin logs

| **Category** | **Percona XtraBackup** | **mysqldump and binary logging** |
|---|---|---|

| | | |
|---|---|---|
| Performance | **Non-Blocking Backups:** XtraBackup performs hot backups without locking tables, meaning the database remains available for reads and writes during the backup process. This is crucial for maintaining application availability.<br>**Incremental Backups:** XtraBackup supports incremental backups, which capture only the changes since the last backup. This reduces the time and storage space needed for backups compared to full dumps every time. | **Blocking:** `mysqldump` can lock tables (depending on options used), which can cause significant downtime, especially on large databases.<br>**Performance Impact:** `mysqldump` can put a considerable load on the server, impacting performance. |
| Scalability | **Efficient for Large Databases:** XtraBackup is designed to handle large datasets efficiently. It directly copies the data files, making it much faster and less resource-intensive than logical backups.<br>**Optimized Storage:** Incremental and compressed backups help in managing storage effectively. | **Time-Consuming:** `mysqldump` can be very slow for large databases because it converts the entire database to SQL statements.<br>**Storage Intensive:** Frequent full dumps consume a lot of storage space. |

| | | |
|---|---|---|
| Consistency | **Point-in-Time Recovery:** XtraBackup captures a consistent snapshot of the database and applies the transaction logs to ensure data consistency. This is crucial for point-in-time recovery.<br>**Consistency Across Storage Engines:** XtraBackup supports InnoDB and other storage engines, ensuring that backups are consistent across different types of tables. | **Potential Inconsistencies:** With `mysqldump`, obtaining a consistent snapshot requires careful management of transactions and locks, which can be complex and error-prone.<br>**Manual Effort:** Ensuring consistent backups with binary logs often requires combining multiple tools and scripts, adding to administrative overhead. |
| Features | **Compression and Encryption:** XtraBackup supports compressed and encrypted backups out of the box, providing security and storage efficiency.<br>**Streaming:** Backups can be streamed to another server, facilitating remote storage and integration with cloud services. | **Lack of Built-in Compression/Encryption:** `mysqldump` does not provide native support for compression or encryption. These need to be managed separately.<br>**Complexity:** Handling streaming, compression, and encryption typically requires additional scripts and tools, increasing complexity. |
| Recovery Speed | **Fast Recovery:** Restoring from XtraBackup is generally faster because it involves copying data files back to the data directory and applying logs, rather than replaying SQL statements.<br>**Prepared Backups:** XtraBackup's `--prepare` phase ensures that the backups are ready to be restored quickly and efficiently. | **Slower Recovery:** Restoring from `mysqldump` involves replaying all SQL statements, which can be very slow for large datasets.<br>**Manual Log Replay:** Combining binary logs with a `mysqldump` backup for point-in-time recovery can be complex and time-consuming. |

---