MySQL Backup and Recovery - Beginner Guide

MySQL Backup and Recovery - Beginner Guide

Table of Contents

- What is Database Backup?
- Why Do We Need Backups?
- Understanding Backup Types
- Method 1: Using mysqldump
- Method 2: Using Binary Logs
- Recovery Method 1: Full Restore
- Recovery Method 2: Point-in-Time Recovery
- Best Practices
- Common Mistakes to Avoid

What is Database Backup?

A database backup is simply a **copy of your database** that you can use to restore your data if something goes wrong. Think of it like creating a save point in a video game - if you make a mistake, you can go back to that save point.

When Things Go Wrong

- Someone accidentally deletes important data
- Hardware fails (hard drive crashes)
- Software bugs corrupt your database
- Natural disasters affect your servers
- Ransomware or security breaches

Having a backup means you can get your data back!

Why Do We Need Backups?

Imagine you run an online store with customer orders, and someone accidentally runs this command:

```sql
DROP TABLE orders;  -- Oops! All orders are gone!
```

Without backup: All customer orders are permanently lost. Your business is in serious trouble.

With backup: You restore yesterday's backup and only lose a few hours of data. Crisis averted!

Key Terms to Know

- **Backup**: A copy of your database saved at a specific time
- **Recovery/Restore**: The process of bringing back your data from a backup
- **RTO (Recovery Time Objective)**: How long can your business be down?
- **RPO (Recovery Point Objective)**: How much data loss can you accept?

Example:

- If your RPO is 1 hour, you need backups at least every hour
- If your RTO is 30 minutes, you need to practice restoring quickly

---

Understanding Backup Types

Full Backup

A complete copy of your entire database at one point in time.

Pros:

- Easy to understand
- Simple to restore
- Everything is in one place

Cons:

- Takes longer for large databases
- Uses more storage space

When to use: Small to medium databases, or as a weekly/monthly base backup

Incremental Backup

Only saves the **changes** since your last backup.

Pros:

- Much faster
- Uses less storage
- Can backup more frequently

Cons:

- Requires the original full backup to work
- More complex to restore

When to use: Daily backups to capture changes between full backups

---

Method 1: Using mysqldump

`mysqldump` is the most popular and beginner-friendly backup tool for MySQL. It creates a text file containing SQL commands that can recreate your database.

Step 1: Basic Backup Command

```shell
mysqldump -u root -p my_database > backup.sql
```

Let's break this down:

- `mysqldump`: The backup program
- `-u root`: Connect as user "root"
- `-p`: Prompt for password (more secure than typing it in the command)
- `my_database`: Name of your database
- `> backup.sql`: Save the output to a file called backup.sql

Example with real database name:

```shell
mysqldump -u root -p online_store > online_store_backup.sql
```

You'll be asked for your password, and then the backup will be created!

Step 2: Backup All Databases

To backup everything at once:

```shell
mysqldump -u root -p --all-databases > all_databases_backup.sql
```

Step 3: Add Important Options

For a better backup, use these options:

```shell
mysqldump -u root -p \
  --single-transaction \
  --routines \
  --triggers \
  my_database > my_database_backup.sql
```

What these options do:

- `--single-transaction`: Creates a consistent backup without locking your database (important for live systems!)
- `--routines`: Includes stored procedures and functions

- `--triggers`: Includes triggers (automatic actions in your database)

Step 4: Compress Your Backup

Backups can be large! Save space by compressing:

```shell
mysqldump -u root -p my_database | gzip > my_database_backup.sql.gz
```

This creates a compressed `.gz` file that's much smaller!

Step 5: Add a Date to Filename

Make it easy to find backups by adding the date:

```shell
mysqldump -u root -p my_database > backup_$(date +%Y%m%d).sql
```

This creates files like: `backup_20251026.sql`

Complete Example

Here's a complete backup command you can use:

```shell
mysqldump -u root -p \
  --single-transaction \
  --routines \
  --triggers \
  --quick \
  online_store | gzip > online_store_backup_$(date +%Y%m%d).sql.gz
```

Method 2: Using Binary Logs

Binary logs record **every change** made to your database in real-time. This allows you to recover to a specific point in time, not just to when you made a backup!

What Are Binary Logs?

Think of binary logs as a detailed diary of everything that happens in your database:

- 10:00 AM - New customer registered
- 10:05 AM - Customer added item to cart
- 10:10 AM - Order completed
- 10:15 AM - Oops! Someone deleted the orders table!

With binary logs, you can "replay" all events up to 10:14 AM, before the mistake happened!

Step 1: Enable Binary Logging

Edit your MySQL configuration file (usually `/etc/mysql/my.cnf` or `C:\ProgramData\MySQL\my.ini` on Windows):

```ini
[mysqld]
log-bin=mysql-bin
server-id=1
binlog_format=ROW
```

What these settings mean:

- `log-bin=mysql-bin`: Enable binary logging with prefix "mysql-bin"
- `server-id=1`: Give your server a unique ID
- `binlog_format=ROW`: Record actual data changes (most reliable)

Step 2: Restart MySQL

After editing the config file:

```shell
# Linux
sudo systemctl restart mysql

# Windows
# Restart MySQL service from Services panel
```

Step 3: View Your Binary Logs

Check that binary logging is working:

```sql
SHOW BINARY LOGS;
```

Output will look like:

```
+------------------+-----------+
| Log_name         | File_size |
+------------------+-----------+
| mysql-bin.000001 |      1547 |
| mysql-bin.000002 |      3894 |
+------------------+-----------+
```

Step 4: Backup Binary Logs

Binary logs are stored in your MySQL data directory. Copy them regularly:

```shell
# Find your data directory
mysql -u root -p -e "SHOW VARIABLES LIKE 'datadir';"

# Copy binary logs to backup location
cp /var/lib/mysql/mysql-bin.* /backup/binlogs/
```

Step 5: Create New Binary Log

Force MySQL to start a new binary log file (good before backups):

```sql
FLUSH LOGS;
```

Complete Backup Strategy

Combine both methods for best results:

```shell
# 1. Create a new binary log
mysql -u root -p -e "FLUSH LOGS;"

# 2. Make a full backup
mysqldump -u root -p --single-transaction \
  --master-data=2 \
  --all-databases | gzip > full_backup_$(date +%Y%m%d).sql.gz

# 3. Backup binary logs daily
cp /var/lib/mysql/mysql-bin.* /backup/binlogs/
```

The `--master-data=2` option saves the binary log position in your backup file, which helps with recovery!

Recovery Method 1: Full Restore

This is the simplest recovery method - restore everything from your backup file.

Step 1: Prepare for Restore

Important: Restoring will overwrite existing data! Make sure you want to do this.

If the database doesn't exist, create it first:

```sql
CREATE DATABASE my_database;
```

Step 2: Restore from Backup

From uncompressed backup:

```shell
mysql -u root -p my_database < backup.sql
```

From compressed backup:

```shell
gunzip < backup.sql.gz | mysql -u root -p my_database
```

Step 3: Verify the Restore

Check that your data is back:

```sql
USE my_database;
SHOW TABLES;
SELECT COUNT(*) FROM important_table;
```

Complete Example

Let's say you accidentally deleted your `online_store` database:

```shell
# Step 1: Recreate the database
mysql -u root -p -e "CREATE DATABASE online_store;"

# Step 2: Restore from yesterday's backup
gunzip < online_store_backup_20251025.sql.gz | mysql -u root -p online_store

# Step 3: Check your data
mysql -u root -p online_store -e "SELECT COUNT(*) FROM orders;"
```

Restore All Databases

If you backed up all databases:

```shell
mysql -u root -p < all_databases_backup.sql
```

No need to create databases first - the backup file contains those commands!

Monitor Restore Progress

For large databases, you can monitor progress:

```shell
# Linux - shows progress bar
pv backup.sql | mysql -u root -p my_database

# Or use verbose mode
mysql -u root -p my_database < backup.sql -v
```

Recovery Method 2: Point-in-Time Recovery

This method lets you restore to a **specific moment in time**, not just to when you made your backup. This is perfect when you know exactly when something went wrong!

Scenario

- You made a full backup yesterday at midnight
- Today at 2:30 PM, someone accidentally deleted the `orders` table
- You want to restore everything up to 2:29 PM (just before the mistake)

Step 1: Restore the Full Backup

First, restore your most recent full backup:

```shell
gunzip < full_backup_20251025.sql.gz | mysql -u root -p
```

Now your database is back to yesterday at midnight. But you're missing all of today's data!

Step 2: Find the Binary Log Position

We need to find where in the binary logs the accident happened.

View the binary log contents:

```shell
mysqlbinlog /var/lib/mysql/mysql-bin.000003 | less
```

Look for timestamps and find the problem:

```sql
#251026 14:30:15 server id 1
DROP TABLE orders;  /* This is the mistake! */
```

Note the time: **14:30:15** (2:30:15 PM)

Step 3: Apply Binary Logs Up to That Time

Apply all changes from the binary logs, but stop just before the mistake:

```shell
mysqlbinlog --stop-datetime="2025-10-26 14:30:00" \
  /var/lib/mysql/mysql-bin.000003 | mysql -u root -p
```

What this does:

- Reads all changes from the binary log
- Stops at 14:30:00 (30 seconds before the DROP TABLE)
- Applies those changes to your database

Step 4: Verify Recovery

Check that your data is correct:

```sql
USE online_store;
SHOW TABLES;  -- orders table should be there!
SELECT MAX(order_date) FROM orders;  -- Should show recent orders
```

Complete Point-in-Time Recovery Example

```shell
# 1. Restore full backup from midnight
echo "Restoring full backup..."
gunzip < full_backup_20251025.sql.gz | mysql -u root -p

# 2. Apply binary logs until just before accident
echo "Applying changes up to 2:30 PM..."
mysqlbinlog --stop-datetime="2025-10-26 14:30:00" \
  /var/lib/mysql/mysql-bin.000003 | mysql -u root -p

# 3. Verify
mysql -u root -p -e "USE online_store; SHOW TABLES;"
```

Using Positions Instead of Time

If you know the exact position in the binary log:

```shell
mysqlbinlog --stop-position=12345 \
  /var/lib/mysql/mysql-bin.000003 | mysql -u root -p
```

Find positions using:

```shell
mysqlbinlog /var/lib/mysql/mysql-bin.000003 | grep -n "DROP TABLE"
```

Best Practices

1. The 3-2-1 Backup Rule

Remember this simple rule:

- Keep **3** copies of your data (original + 2 backups)
- Store on **2** different types of media (e.g., local disk + cloud)
- Keep **1** copy offsite (different location)

Example:

- Original: Production database server
- Copy 1: Backup on same server
- Copy 2: Backup on external drive
- Copy 3: Backup in cloud storage (AWS S3, Google Drive, etc.)

2. Create a Backup Schedule

For small databases (< 10 GB):

- **Daily**: Full backup using mysqldump
- **Keep**: Last 7 daily backups

For larger databases:

- **Weekly**: Full backup (Sundays)
- **Daily**: Binary log backup
- **Keep**: Last 4 weekly backups + all binary logs

3. Automate Your Backups

Create a simple backup script:

```shell
#!/bin/bash
# backup_mysql.sh

# Configuration
DATE=$(date +%Y%m%d)
BACKUP_DIR="/backup/mysql"
DATABASE="online_store"

# Create backup directory if it doesn't exist
mkdir -p $BACKUP_DIR

# Perform backup
echo "Starting backup at $(date)"
mysqldump -u root -p'YourPassword' \
  --single-transaction \
  --routines \
  --triggers \
  $DATABASE | gzip > $BACKUP_DIR/${DATABASE}_${DATE}.sql.gz

# Check if backup was successful
if [ $? -eq 0 ]; then
    echo "Backup completed successfully!"
else
    echo "Backup failed!" | mail -s "Backup Failed" admin@example.com
fi

# Delete backups older than 7 days
find $BACKUP_DIR -name "*.sql.gz" -mtime +7 -delete

echo "Backup finished at $(date)"
```

Make it executable and schedule it:

```shell
chmod +x backup_mysql.sh

# Add to cron (Linux) - runs every day at 2 AM
crontab -e
# Add this line:
0 2 * * * /path/to/backup_mysql.sh >> /var/log/mysql_backup.log 2>&1
```

Windows users: Use Task Scheduler to run this script daily.

4. Test Your Backups Regularly

Never trust an untested backup!

Create a monthly reminder to:

```shell
# 1. Restore to a test database
gunzip < latest_backup.sql.gz | mysql -u root -p test_database

# 2. Verify data
mysql -u root -p test_database -e "SELECT COUNT(*) FROM important_table;"

# 3. Test point-in-time recovery
# Try restoring to a specific time

# 4. Document how long it took
echo "Restore test completed in X minutes" >> restore_test_log.txt
```

5. Secure Your Backups

Protect your backup files:

```shell
# Set proper file permissions (Linux)
chmod 600 backup.sql.gz
chown mysql:mysql backup.sql.gz

# Encrypt sensitive backups
mysqldump -u root -p my_database | \
  openssl enc -aes-256-cbc -salt -out backup_encrypted.sql.enc

# To decrypt and restore:
openssl enc -d -aes-256-cbc -in backup_encrypted.sql.enc | \
  mysql -u root -p my_database
```

Store credentials securely:

Instead of putting passwords in scripts, use a config file:

```shell
# Create ~/.my.cnf
[client]
user=root
password=YourSecurePassword

# Protect it
chmod 600 ~/.my.cnf

# Now you can backup without exposing password
mysqldump --defaults-file=~/.my.cnf my_database > backup.sql
```

6. Document Your Process

Create a simple document with:

Backup Information:

- Location of backups
- Backup schedule
- Retention policy (how long to keep backups)
- Who is responsible for backups

Recovery Procedures:

- Step-by-step restore instructions
- Emergency contacts
- Estimated recovery time

Example documentation:

```
MySQL Backup Information
_____________________

Database: online_store
Backup Location: /backup/mysql
Cloud Backup: AWS S3 bucket "mycompany-backups"

Schedule:
- Daily backups at 2:00 AM
- Keep 7 daily backups
- Monthly full backups kept for 1 year

To Restore:
1. Find latest backup in /backup/mysql
2. Run: gunzip < backup.sql.gz | mysql -u root -p online_store
3. Verify: mysql -u root -p -e "USE online_store; SHOW TABLES;"

Emergency Contact: DBA Team - dba@company.com - 555-0123
Estimated Recovery Time: 30 minutes
```

7. Monitor Your Backups

Set up alerts for:

- Failed backups
- Backup files not created on schedule
- Backup files that are too small (might be incomplete)
- Disk space running low

Simple monitoring script:

```shell
#!/bin/bash
# check_backup.sh

BACKUP_FILE="/backup/mysql/online_store_$(date +%Y%m%d).sql.gz"
MIN_SIZE=1000000  # Minimum 1 MB

if [ -f "$BACKUP_FILE" ]; then
    SIZE=$(stat -f%z "$BACKUP_FILE" 2>/dev/null || stat -c%s "$BACKUP_FILE")
    if [ $SIZE -gt $MIN_SIZE ]; then
        echo "Backup OK: $BACKUP_FILE ($SIZE bytes)"
    else
        echo "WARNING: Backup file too small!"
        # Send alert email
    fi
else
    echo "ERROR: Backup file not found!"
    # Send alert email
fi
```

Common Mistakes to Avoid

❌ Mistake 1: Only Keeping One Backup

Problem: If that backup is corrupted, you have nothing.

Solution: Keep multiple backups (3-2-1 rule)

❌ Mistake 2: Never Testing Restores

Problem: You don't know if your backup works until it's too late.

Solution: Test restore monthly

❌ Mistake 3: Storing Backups on Same Server

Problem: If the server dies, you lose backups too.

Solution: Always have offsite backups

❌ Mistake 4: Putting Passwords in Scripts

Problem: Security risk - anyone can read the password.

Solution: Use configuration files with restricted permissions

❌ Mistake 5: Not Enabling Binary Logs

Problem: Can only restore to backup time, not any specific moment.

Solution: Enable binary logging for point-in-time recovery

❌ Mistake 6: Ignoring Backup Failures

Problem: Backups fail silently, no one notices until needed.

Solution: Set up monitoring and alerts

❌ Mistake 7: No Documentation

Problem: During emergency, people don't know how to restore.

Solution: Write clear, step-by-step recovery instructions

Quick Reference

Backup Cheat Sheet

```shell
# Simple full backup
mysqldump -u root -p my_database > backup.sql

# Better full backup (compressed, with all features)
mysqldump -u root -p --single-transaction --routines --triggers \
  my_database | gzip > backup_$(date +%Y%m%d).sql.gz

# Backup all databases
mysqldump -u root -p --all-databases > all_backup.sql

# Enable binary logs (in my.cnf)
[mysqld]
log-bin=mysql-bin
server-id=1

# View binary logs
SHOW BINARY LOGS;
```

Restore Cheat Sheet

```shell
# Simple restore
mysql -u root -p my_database < backup.sql

# Restore compressed backup
gunzip < backup.sql.gz | mysql -u root -p my_database

# Point-in-time recovery
# 1. Restore full backup
gunzip < full_backup.sql.gz | mysql -u root -p
# 2. Apply binary logs to specific time
mysqlbinlog --stop-datetime="2025-10-26 14:30:00" \
  mysql-bin.000003 | mysql -u root -p
```

Practice Exercise

Try this on a test database to learn:

1. **Create a test database:**

```sql
CREATE DATABASE practice_backup;
USE practice_backup;
CREATE TABLE users (id INT, name VARCHAR(50));
INSERT INTO users VALUES (1, 'Alice'), (2, 'Bob');
```

2. **Make a backup:**

```shell
mysqldump -u root -p practice_backup > backup1.sql
```

3. **Add more data:**

```sql
INSERT INTO users VALUES (3, 'Charlie');
```

4. **Make another backup:**

```shell
mysqldump -u root -p practice_backup > backup2.sql
```

5. **Simulate disaster:**

```sql
DROP TABLE users;  -- Oops!
```

6. **Restore:**

```shell
mysql -u root -p practice_backup < backup2.sql
```

7. **Verify:**

```sql
USE practice_backup;
SELECT * FROM users;  -- Should see Alice, Bob, and Charlie
```

Congratulations! You've performed your first backup and recovery!

Summary

Remember these key points:

1. **Always have backups** - Use mysqldump for simple, reliable backups
2. **Enable binary logs** - Allows recovery to any point in time
3. **Follow 3-2-1 rule** - Multiple copies in different locations
4. **Test regularly** - Practice restoring before you really need it
5. **Automate everything** - Use scripts and scheduling
6. **Monitor and alert** - Know when backups fail
7. **Document your process** - Make recovery easy during emergencies

> **Most Important:** The best backup is the one you can successfully restore. Test your backups!
