MySQL Cheatsheet

Essential commands for database management and SQL operations

This cheatsheet provides a quick reference to fundamental MySQL commands, syntax, and advanced features, ideal for both beginners and experienced developers for efficient database operations and data management.

Database Connection
Connect to MySQL servers

Data Querying
Select and filter data

Data Manipulation
Insert, update, and delete records

Database Schema
Create and manage tables

Performance Optimization
Indexes and query optimization

Database Connection & Management

Connect to Server: `mysql -u username -p`

Connect to MySQL server using command line.

```
# Connect with username and password prompt
mysql -u root -p
# Connect to specific database
mysql -u username -p database_name
# Connect to remote server
mysql -h hostname -u username -p
# Connect with port specification
mysql -h hostname -P 3306 -u username -p database_name
```

Database Operations: `CREATE` / `DROP` / `USE`

Manage databases on the server.

```
# Create a new database
CREATE DATABASE company_db;
# List all databases
SHOW DATABASES;
# Select a database to use
USE company_db;
# Drop a database (delete permanently)
DROP DATABASE old_database;
```

Export Data: `mysqldump`

Backup database data to SQL file.

```
# Export entire database
mysqldump -u username -p database_name > backup.sql
# Export specific table
mysqldump -u username -p database_name table_name > table_backup.sql
# Export with structure only
mysqldump -u username -p --no-data database_name > structure.sql
```

Import Data: `mysql < file.sql`

Import SQL file into MySQL database.

```
# Import SQL file into database
mysql -u username -p database_name < backup.sql
# Import without specifying database (if included in file)
mysql -u username -p < full_backup.sql
```

User Management: `CREATE USER` / `GRANT`

Manage database users and permissions.

```
# Create new user
CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'password';
# Grant all privileges
GRANT ALL PRIVILEGES ON database_name.* TO 'user'@'localhost';
# Grant specific privileges
GRANT SELECT, INSERT, UPDATE ON table_name TO 'user'@'localhost';
# Apply privilege changes
FLUSH PRIVILEGES;
```

Show Server Info: `SHOW STATUS` / `SHOW VARIABLES`

Display server configuration and status.

```
# Show server status
SHOW STATUS;
# Show configuration variables
SHOW VARIABLES;
# Show current processes
SHOW PROCESSLIST;
```

Table Structure & Schema

Create and manage database tables and their structure.

01

Table Creation: `CREATE TABLE`

Create new tables with specified columns and data types.

```
# Create table with various data types
CREATE TABLE users (
   id INT AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(50) NOT NULL,
   email VARCHAR(100) UNIQUE,
   age INT,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
# Create table with foreign key
CREATE TABLE orders (
   order_id INT AUTO_INCREMENT PRIMARY KEY,
   user_id INT,
   FOREIGN KEY (user_id) REFERENCES users(id)
);
```

02

Table Information: `DESCRIBE` / `SHOW`

View table structure and database contents.

```
# Show table structure
DESCRIBE users;
# Alternative syntax
SHOW COLUMNS FROM users;
# List all tables
SHOW TABLES;
# Show CREATE statement for table
SHOW CREATE TABLE users;
```

03

Modify Tables: `ALTER TABLE`

Change existing table structure, add or drop columns.

```
# Add new column
ALTER TABLE users ADD COLUMN phone VARCHAR(20);
# Drop column
ALTER TABLE users DROP COLUMN age;
# Modify column type
ALTER TABLE users MODIFY COLUMN username VARCHAR(100);
# Rename column
ALTER TABLE users CHANGE old_name new_name VARCHAR(50);
```

Data Manipulation & CRUD Operations

Insert Data: `INSERT INTO`

Add new records to tables.

```
# Insert single record
INSERT INTO users (username, email, age)
VALUES ('john_doe', 'john@email.com', 25);
# Insert multiple records
INSERT INTO users (username, email, age) VALUES
('alice', 'alice@email.com', 30),
('bob', 'bob@email.com', 28);
# Insert from another table
INSERT INTO users_backup SELECT * FROM users;
```

Update Data: `UPDATE`

Modify existing records in tables.

```
# Update specific record
UPDATE users SET age = 26 WHERE username = 'john_doe';
# Update multiple columns
UPDATE users SET age = 31, email = 'alice_new@email.com'
WHERE username = 'alice';
# Update with calculation
UPDATE products SET price = price * 1.1 WHERE category = 'electronics';
```

Delete Data: `DELETE` / `TRUNCATE`

Remove records from tables.

```
# Delete specific records
DELETE FROM users WHERE age < 18;
# Delete all records (keep structure)
DELETE FROM users;
# Delete all records (faster, resets AUTO_INCREMENT)
TRUNCATE TABLE users;
# Delete with JOIN
DELETE u FROM users u
JOIN inactive_accounts i ON u.id = i.user_id;
```

Replace Data: `REPLACE` / `INSERT ... ON DUPLICATE KEY`

Handle duplicate key situations during inserts.

```
# Replace existing or insert new
REPLACE INTO users (id, username, email)
VALUES (1, 'updated_user', 'new@email.com');
# Insert or update on duplicate key
INSERT INTO users (username, email)
VALUES ('john', 'john@email.com')
ON DUPLICATE KEY UPDATE email = VALUES(email);
```

Data Querying & Selection

Basic SELECT: `SELECT * FROM`

Retrieve data from tables with various conditions.

```
# Select all columns
SELECT * FROM users;
# Select specific columns
SELECT username, email FROM users;
# Select with WHERE condition
SELECT * FROM users WHERE age > 25;
# Select with multiple conditions
SELECT * FROM users WHERE age > 20 AND email LIKE '%gmail.com';
```

Sorting & Limiting: `ORDER BY` / `LIMIT`

Control the order and number of returned results.

```
# Sort results
SELECT * FROM users ORDER BY age DESC;
# Sort by multiple columns
SELECT * FROM users ORDER BY age DESC, username ASC;
# Limit results
SELECT * FROM users LIMIT 10;
# Pagination (skip first 10, take next 10)
SELECT * FROM users LIMIT 10 OFFSET 10;
```

Filtering: `WHERE` / `LIKE` / `IN`

Filter data using various comparison operators.

```
# Pattern matching
SELECT * FROM users WHERE username LIKE 'john%';
# Multiple values
SELECT * FROM users WHERE age IN (25, 30, 35);
# Range filtering
SELECT * FROM users WHERE age BETWEEN 20 AND 30;
# NULL checks
SELECT * FROM users WHERE email IS NOT NULL;
```

Grouping: `GROUP BY` / `HAVING`

Group data and apply aggregate functions.

```
# Group by column
SELECT age, COUNT(*) FROM users GROUP BY age;
# Group with condition on groups
SELECT age, COUNT(*) as count FROM users
GROUP BY age HAVING count > 1;
# Multiple grouping columns
SELECT age, gender, COUNT(*) FROM users
GROUP BY age, gender;
```

Advanced Querying

Complex query operations and data analysis techniques.

JOIN Operations: `INNER` / `LEFT` / `RIGHT`

Combine data from multiple tables.

```
# Inner join (matching records only)
SELECT u.username, o.order_date
FROM users u
INNER JOIN orders o ON u.id = o.user_id;
# Left join (all users, matched orders)
SELECT u.username, o.order_date
FROM users u
LEFT JOIN orders o ON u.id = o.user_id;
# Multiple joins
SELECT u.username, o.order_date, p.product_name
FROM users u
JOIN orders o ON u.id = o.user_id
JOIN products p ON o.product_id = p.id;
```

Subqueries: `SELECT` within `SELECT`

Use nested queries for complex data retrieval.

```
# Subquery in WHERE clause
SELECT * FROM users
WHERE id IN (SELECT user_id FROM orders WHERE total > 100);
# Correlated subquery
SELECT username FROM users u1
WHERE age > (SELECT AVG(age) FROM users u2);
# Subquery in SELECT
SELECT username,
(SELECT COUNT(*) FROM orders WHERE user_id = users.id) as order_count
FROM users;
```

Aggregate Functions: `COUNT` / `SUM` / `AVG`

Calculate statistics and summaries from data.

```
# Basic aggregates
SELECT COUNT(*) FROM users;
SELECT AVG(age), MIN(age), MAX(age) FROM users;
SELECT SUM(total) FROM orders;
# Aggregate with grouping
SELECT department, AVG(salary)
FROM employees GROUP BY department;
# Multiple aggregates
SELECT
   COUNT(*) as total_users,
   AVG(age) as avg_age,
   MAX(created_at) as latest_signup
FROM users;
```

Window Functions: `OVER` / `PARTITION BY`

Perform calculations across sets of table rows.

```
# Ranking functions
SELECT username, age,
RANK() OVER (ORDER BY age DESC) as age_rank
FROM users;
# Partition by group
SELECT username, department, salary,
AVG(salary) OVER (PARTITION BY department) as dept_avg
FROM employees;
# Running totals
SELECT order_date, total,
SUM(total) OVER (ORDER BY order_date) as running_total
FROM orders;
```

Indexes & Performance

Optimize database performance with proper indexing strategies.

Create Indexes: `CREATE INDEX`

Improve query performance with database indexes.

```
# Create regular index
CREATE INDEX idx_username ON users(username);
# Create composite index
CREATE INDEX idx_user_age ON users(username, age);
# Create unique index
CREATE UNIQUE INDEX idx_email ON users(email);
# Show indexes on table
SHOW INDEXES FROM users;
```

Query Analysis: `EXPLAIN`

Analyze query execution plans and performance.

```
# Show query execution plan
EXPLAIN SELECT * FROM users WHERE age > 25;
# Detailed analysis
EXPLAIN FORMAT=JSON SELECT u.*, o.total
FROM users u JOIN orders o ON u.id = o.user_id;
# Show query performance
SHOW PROFILES;
SET profiling = 1;
```

Optimize Queries: Best Practices

Techniques for writing efficient SQL queries.

```
# Use specific columns instead of *
SELECT username, email FROM users WHERE id = 1;
# Use LIMIT for large datasets
SELECT * FROM logs ORDER BY created_at DESC LIMIT 1000;
# Use proper WHERE conditions
SELECT * FROM orders WHERE user_id = 123 AND status = 'pending';
-- Use covering indexes when possible
```

Table Maintenance: `OPTIMIZE` / `ANALYZE`

Maintain table performance and statistics.

```
# Optimize table storage
OPTIMIZE TABLE users;
# Update table statistics
ANALYZE TABLE users;
# Check table integrity
CHECK TABLE users;
# Repair table if needed
REPAIR TABLE users;
```

Data Import/Export

Transfer data between MySQL and external formats.

Load Data: `LOAD DATA INFILE`

Import data from CSV and text files.

```
# Load CSV file
LOAD DATA INFILE '/path/to/data.csv'
INTO TABLE users
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
# Load with specific columns
LOAD DATA INFILE '/path/to/data.csv'
INTO TABLE users (username, email, age);
```

Export Data: `SELECT INTO OUTFILE`

Export query results to files.

```
# Export to CSV file
SELECT username, email, age
FROM users
INTO OUTFILE '/path/to/export.csv'
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Backup & Restore: `mysqldump` / `mysql`

Create and restore database backups.

```
# Full database backup
mysqldump -u username -p --routines --triggers database_name > backup.sql
# Backup specific tables
mysqldump -u username -p database_name table1 table2 > tables_backup.sql
# Restore from backup
mysql -u username -p database_name < backup.sql
```

Remote Data Transfer

Transfer data between different MySQL servers.

```
# Export from remote server
mysqldump -h remote_host -u username -p database_name > remote_backup.sql
# Import to local database
mysql -u local_user -p local_database < remote_backup.sql
# Direct data copying between servers
mysqldump -h source_host -u user -p db_name | mysql -h dest_host -u user -p db_name
```

Data Types & Functions

Understanding MySQL data types and built-in functions.

Common Data Types: Numbers, Text, Dates

Choose appropriate data types for your columns.

```
# Numeric types
INT, BIGINT, DECIMAL(10,2), FLOAT, DOUBLE
# String types
VARCHAR(255), TEXT, CHAR(10), MEDIUMTEXT, LONGTEXT
# Date and time types
DATE, TIME, DATETIME, TIMESTAMP, YEAR
# Boolean and binary
BOOLEAN, BLOB, VARBINARY
# Example table creation
CREATE TABLE products (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   price DECIMAL(10,2),
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

String Functions: `CONCAT` / `SUBSTRING` / `LENGTH`

Manipulate text data with built-in string functions.

```
# String concatenation
SELECT CONCAT(first_name, ' ', last_name) as full_name FROM users;
# String operations
SELECT SUBSTRING(email, 1, LOCATE('@', email)-1) as username FROM users;
SELECT LENGTH(username), UPPER(username) FROM users;
# Pattern matching and replacement
SELECT REPLACE(phone, '-', '.') FROM users WHERE phone LIKE '___-___-____';
```

Date Functions: `NOW()` / `DATE_ADD` / `DATEDIFF`

Work with dates and times effectively.

```
# Current date and time
SELECT NOW(), CURDATE(), CURTIME();
# Date arithmetic
SELECT DATE_ADD(created_at, INTERVAL 30 DAY) as expiry_date FROM users;
SELECT DATEDIFF(NOW(), created_at) as days_since_signup FROM users;
# Date formatting
SELECT DATE_FORMAT(created_at, '%Y-%m-%d %H:%i') as formatted_date FROM orders;
```

Numeric Functions: `ROUND` / `ABS` / `RAND`

Perform mathematical operations on numeric data.

```
# Mathematical functions
SELECT ROUND(price, 2), ABS(profit_loss), SQRT(area) FROM products;
# Random and statistical
SELECT RAND(), FLOOR(price), CEIL(rating) FROM products;
# Aggregate math
SELECT AVG(price), STDDEV(price), VARIANCE(price) FROM products;
```

Transaction Management

Ensure data integrity with transaction control.

Transaction Control: `BEGIN` / `COMMIT` / `ROLLBACK`

Manage database transactions for data consistency.

```
# Start transaction
BEGIN;
# or
START TRANSACTION;

# Perform operations
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;

# Commit changes
COMMIT;

# Or rollback if error
ROLLBACK;
```

Transaction Isolation: `SET TRANSACTION ISOLATION`

Control how transactions interact with each other.

```
# Set isolation level
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
# Show current isolation level
SELECT @@transaction_isolation;
```

Locking: `LOCK TABLES` / `SELECT FOR UPDATE`

Control concurrent access to data.

```
# Lock tables for exclusive access
LOCK TABLES users WRITE, orders READ;
# Perform operations
# ...
UNLOCK TABLES;

# Row-level locking in transactions
BEGIN;
SELECT * FROM accounts WHERE id = 1 FOR UPDATE;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
COMMIT;
```

Savepoints: `SAVEPOINT` / `ROLLBACK TO`

Create rollback points within transactions.

```
BEGIN;
INSERT INTO users (username) VALUES ('user1');
SAVEPOINT sp1;
INSERT INTO users (username) VALUES ('user2');
SAVEPOINT sp2;
INSERT INTO users (username) VALUES ('user3');
# Rollback to savepoint
ROLLBACK TO sp1;
COMMIT;
```

MySQL Installation & Setup

Install and configure MySQL for your development environment.

Installation: Package Managers

Install MySQL using system package managers.

```
# Ubuntu/Debian
sudo apt update
sudo apt install mysql-server
# CentOS/RHEL
sudo yum install mysql-server
# macOS with Homebrew
brew install mysql
# Start MySQL service
sudo systemctl start mysql
```

Docker: `docker run mysql`

Run MySQL in Docker containers for development.

```
# Run MySQL container
docker run --name mysql-dev -e MYSQL_ROOT_PASSWORD=password -p 3306:3306 -d mysql:8.0
# Connect to containerized MySQL
docker exec -it mysql-dev mysql -u root -p
# Create database in container
docker exec -it mysql-dev mysql -u root -p -e "CREATE DATABASE testdb;"
```

Initial Setup & Security

Secure your MySQL installation and verify setup.

```
# Run security script
sudo mysql_secure_installation
# Connect to MySQL
mysql -u root -p
# Show MySQL version
SELECT VERSION();
# Check connection status
STATUS;
# Set root password
ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
```

Configuration & Settings

Configure MySQL server settings and optimize performance.

Configuration Files: `my.cnf`

Modify MySQL server configuration settings.

```
# Common configuration locations
# Linux: /etc/mysql/my.cnf
# Windows: C:\ProgramData\MySQL\MySQL Server\my.ini
# macOS: /usr/local/etc/my.cnf

[mysqld]
max_connections = 200
innodb_buffer_pool_size = 1G
query_cache_size = 64M
slow_query_log = 1
long_query_time = 2
```

Runtime Configuration: `SET GLOBAL`

Change settings while MySQL is running.

```
# Set global variables
SET GLOBAL max_connections = 500;
SET GLOBAL slow_query_log = ON;
# Show current settings
SHOW VARIABLES LIKE 'max_connections';
SHOW GLOBAL STATUS LIKE 'Slow_queries';
```

Performance Tuning: Memory & Cache

Optimize MySQL performance settings.

```
# Show memory usage
SHOW VARIABLES LIKE '%buffer_pool_size%';
SHOW VARIABLES LIKE '%query_cache%';
# Monitor performance
SHOW STATUS LIKE 'Qcache%';
SHOW STATUS LIKE 'Created_tmp%';
# InnoDB settings
SET GLOBAL innodb_buffer_pool_size = 2147483648; -- 2GB
```

Logging Configuration: Error & Query Logs

Configure MySQL logging for monitoring and debugging.

```
# Enable query logging
SET GLOBAL general_log = 'ON';
SET GLOBAL general_log_file = '/var/log/mysql/query.log';
# Slow query log
SET GLOBAL slow_query_log = 'ON';
SET GLOBAL long_query_time = 1;
# Show log settings
SHOW VARIABLES LIKE '%log%';
```

Advanced SQL Techniques

Complex SQL operations and advanced database programming.

Common Table Expressions (CTEs): `WITH`

Create temporary result sets for complex queries.

```
# Simple CTE
WITH user_orders AS (
   SELECT user_id, COUNT(*) as order_count,
          SUM(total) as total_spent
   FROM orders
   GROUP BY user_id
)
SELECT u.username, uo.order_count, uo.total_spent
FROM users u
JOIN user_orders uo ON u.id = uo.user_id
WHERE uo.total_spent > 1000;
```

Stored Procedures: `CREATE PROCEDURE`

Create reusable database procedures.

```
# Create stored procedure
DELIMITER //
CREATE PROCEDURE GetUserOrders(IN user_id INT)
BEGIN
   SELECT o.*, p.product_name
   FROM orders o
   JOIN products p ON o.product_id = p.id
   WHERE o.user_id = user_id;
END //
DELIMITER ;

# Call procedure
CALL GetUserOrders(123);
```

Triggers: `CREATE TRIGGER`

Automatically execute code in response to database events.

```
# Create trigger for audit logging
CREATE TRIGGER user_update_audit
AFTER UPDATE ON users
FOR EACH ROW
BEGIN
   INSERT INTO user_audit (user_id, old_email, new_email, changed_at)
   VALUES (NEW.id, OLD.email, NEW.email, NOW());
END;

# Show triggers
SHOW TRIGGERS;
```

Views: `CREATE VIEW`

Create virtual tables based on query results.

```
# Create view
CREATE VIEW active_users AS
SELECT id, username, email, created_at
FROM users
WHERE status = 'active' AND last_login > DATE_SUB(NOW(), INTERVAL 30 DAY);

# Use view like a table
SELECT * FROM active_users WHERE username LIKE 'john%';
# Drop view
DROP VIEW active_users;
```

Reference: This MySQL cheat sheet provides a concise and handy reference to the most commonly used MySQL commands and functionalities. It spans a range of topics, from connecting to a MySQL server and managing database contents, to the basic syntax for table creation and modification.