Database Cheatsheet

Essential operations for database management and SQL queries

This cheatsheet provides a quick reference to fundamental database concepts, SQL syntax, and essential database operations, ideal for both beginners and experienced developers for efficient data management.

Database Creation
Create and manage databases

Data Querying
Retrieve and filter data

Data Manipulation
Insert, update, and delete records

Table Management
Create and modify table structures

Database Administration
Manage users and optimize performance

Database Creation & Management

Create Database: `CREATE DATABASE`

Create a new database for storing your data.

```
-- Create a new database
CREATE DATABASE company_db;

-- Create database with character set
CREATE DATABASE company_db
CHARACTER SET utf8mb4
COLLATE utf8mb4_general_ci;

-- Use the database
USE company_db;
```

Show Databases: `SHOW DATABASES`

List all available databases on the server.

```
-- List all databases
SHOW DATABASES;

-- Show database information
SELECT SCHEMA_NAME FROM
INFORMATION_SCHEMA.SCHEMATA;

-- Show current database
SELECT DATABASE();
```

Drop Database: `DROP DATABASE`

Delete an entire database permanently.

```
-- Drop database (be careful!)
DROP DATABASE old_company_db;

-- Drop database if it exists
DROP DATABASE IF EXISTS old_company_db;
```

Backup Database: `mysqldump`

Create backup copies of your database.

```
-- Command line backup
mysqldump -u username -p database_name > backup.sql

-- Restore from backup
mysql -u username -p database_name < backup.sql
```

Database Users: `CREATE USER`

Manage database user accounts and permissions.

```
-- Create new user
CREATE USER 'newuser'@'localhost' IDENTIFIED BY
'password';

-- Grant privileges
GRANT SELECT, INSERT ON company_db.* TO
'newuser'@'localhost';

-- Show user privileges
SHOW GRANTS FOR 'newuser'@'localhost';
```

Database Information: `INFORMATION_SCHEMA`

Query database metadata and structure information.

```
-- Show all tables
SELECT TABLE_NAME FROM
INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'company_db';

-- Show table columns
DESCRIBE employees;
```

Table Structure & Info

Understand and inspect your database table structures.

01

Create Table: `CREATE TABLE`

Define new tables with columns and data types.

```
-- Basic table creation
CREATE TABLE employees (
   id INT AUTO_INCREMENT PRIMARY
KEY,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(150) UNIQUE,
   salary DECIMAL(10,2),
   hire_date DATE,
   department VARCHAR(50)
);

-- Show table structure
DESCRIBE employees;
SHOW COLUMNS FROM employees;
```

02

Alter Table: `ALTER TABLE`

Modify existing table structure and columns.

```
-- Add new column
ALTER TABLE employees ADD
COLUMN phone VARCHAR(15);

-- Modify column type
ALTER TABLE employees MODIFY
COLUMN salary DECIMAL(12,2);

-- Drop column
ALTER TABLE employees DROP
COLUMN phone;

-- Rename table
RENAME TABLE employees TO staff;
```

03

Table Information: `SHOW`

Get detailed information about tables and their properties.

```
-- Show all tables
SHOW TABLES;

-- Show table structure
SHOW CREATE TABLE employees;

-- Show table status
SHOW TABLE STATUS LIKE
'employees';

-- Count rows in table
SELECT COUNT(*) FROM employees;
```

Data Manipulation & CRUD Operations

Insert Data: `INSERT INTO`

Add new records to your tables.

```
-- Insert single record
INSERT INTO employees (name, email, salary, hire_date,
department)
VALUES ('John Doe', 'john@company.com', 75000.00,
'2024-01-15', 'Engineering');

-- Insert multiple records
INSERT INTO employees (name, email, salary,
department) VALUES
('Jane Smith', 'jane@company.com', 80000.00,
'Marketing'),
('Bob Johnson', 'bob@company.com', 65000.00, 'Sales');

-- Insert from another table
INSERT INTO backup_employees
SELECT * FROM employees WHERE department =
'Engineering';
```

Update Data: `UPDATE`

Modify existing records in tables.

```
-- Update single record
UPDATE employees
SET salary = 85000.00, department = 'Senior Engineering'
WHERE id = 1;

-- Update multiple records
UPDATE employees
SET salary = salary * 1.05
WHERE hire_date < '2024-01-01';

-- Update with JOIN
UPDATE employees e
JOIN departments d ON e.department = d.name
SET e.salary = e.salary + d.bonus;
```

Delete Data: `DELETE FROM`

Remove records from tables.

```
-- Delete specific records
DELETE FROM employees
WHERE department = 'Temporary';

-- Delete with conditions
DELETE FROM employees
WHERE hire_date < '2020-01-01' AND salary < 50000;

-- Truncate table (faster for all records)
TRUNCATE TABLE temp_employees;
```

Replace Data: `REPLACE INTO`

Insert or update records based on primary key.

```
-- Replace record (insert or update)
REPLACE INTO employees (id, name, email, salary)
VALUES (1, 'John Doe Updated',
'john.new@company.com', 90000);

-- On duplicate key update
INSERT INTO employees (id, name, salary)
VALUES (1, 'John Doe', 85000)
ON DUPLICATE KEY UPDATE salary = VALUES(salary);
```

Data Querying & Selection

Basic SELECT: `SELECT`

Retrieve data from database tables.

```
-- Select all columns
SELECT * FROM employees;

-- Select specific columns
SELECT name, email, salary FROM employees;

-- Select with alias
SELECT name AS employee_name, salary AS
annual_salary
FROM employees;

-- Select distinct values
SELECT DISTINCT department FROM employees;
```

Filtering Data: `WHERE`

Apply conditions to filter query results.

```
-- Basic conditions
SELECT * FROM employees WHERE salary > 70000;

-- Multiple conditions
```

Sorting Data: `ORDER BY`

Sort query results in ascending or descending order.

```
-- Sort by single column
SELECT * FROM employees ORDER BY salary DESC;

-- Sort by multiple columns
SELECT * FROM employees
ORDER BY department ASC, salary DESC;

-- Sort with LIMIT
SELECT * FROM employees
ORDER BY hire_date DESC LIMIT 10;
```

Limiting Results: `LIMIT`

Control the number of records returned.

```
-- Limit number of results
SELECT * FROM employees LIMIT 5;

-- Pagination with OFFSET
SELECT * FROM employees
ORDER BY id LIMIT 10 OFFSET 20;
```

```
SELECT * FROM employees
WHERE department = 'Engineering' AND salary > 75000;

-- Pattern matching
SELECT * FROM employees WHERE name LIKE 'John%';

-- Range queries
SELECT * FROM employees
WHERE hire_date BETWEEN '2023-01-01' AND '2023-12-
31';
```

```
-- Top N results
SELECT * FROM employees
ORDER BY salary DESC LIMIT 5;
```

Advanced Querying

Aggregate Functions: `COUNT`, `SUM`, `AVG`

Perform calculations on groups of data.

```
-- Count records
SELECT COUNT(*) FROM employees;

-- Sum and average
SELECT SUM(salary) as total_payroll, AVG(salary) as
avg_salary
FROM employees;

-- Group statistics
SELECT department, COUNT(*) as employee_count,
AVG(salary) as avg_salary
FROM employees GROUP BY department;

-- Having clause for group filtering
SELECT department, COUNT(*) as count
FROM employees
GROUP BY department
HAVING COUNT(*) > 5;
```

Subqueries: Nested Queries

Use queries within other queries for complex operations.

```
-- Subquery in WHERE clause
SELECT * FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);

-- Subquery with IN
SELECT * FROM employees
WHERE department IN (
    SELECT name FROM departments WHERE budget >
100000
);

-- Correlated subquery
SELECT * FROM employees e1
WHERE salary > (
    SELECT AVG(salary) FROM employees e2
    WHERE e2.department = e1.department
);
```

Table Joins: `JOIN`

Combine data from multiple tables.

```
-- Inner join
SELECT e.name, e.salary, d.department_name
FROM employees e
INNER JOIN departments d ON e.department = d.id;

-- Left join
SELECT e.name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department = d.id;

-- Multiple joins
SELECT e.name, d.department_name, p.project_name
FROM employees e
LEFT JOIN departments d ON e.department = d.id
LEFT JOIN projects p ON e.id = p.employee_id;
```

Window Functions: Advanced Analytics

Perform calculations across related rows.

```
-- Row numbering
SELECT name, salary,
    ROW_NUMBER() OVER (ORDER BY salary DESC) as rank
FROM employees;

-- Running totals
SELECT name, salary,
    SUM(salary) OVER (ORDER BY hire_date) as
running_total
FROM employees;

-- Partition by groups
SELECT name, department, salary,
    AVG(salary) OVER (PARTITION BY department) as
dept_avg
FROM employees;
```

Database Constraints & Integrity

Maintain data quality and relationships between tables.

Primary Keys: `PRIMARY KEY`

Ensure unique identification for each record.

```
-- Single column primary key
CREATE TABLE employees (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100)
);

-- Composite primary key
CREATE TABLE order_items (
    order_id INT,
    product_id INT,
    quantity INT,
    PRIMARY KEY (order_id, product_id)
);
```

Foreign Keys: `FOREIGN KEY`

Maintain referential integrity between tables.

```
-- Add foreign key constraint
CREATE TABLE orders (
    id INT PRIMARY KEY,
    customer_id INT,
    FOREIGN KEY (customer_id) REFERENCES customers(id)
);

-- Add foreign key to existing table
ALTER TABLE employees
ADD CONSTRAINT fk_department
FOREIGN KEY (department_id) REFERENCES
departments(id);
```

Unique Constraints: `UNIQUE`

Prevent duplicate values in columns.

```
-- Unique constraint on single column
ALTER TABLE employees
ADD CONSTRAINT unique_email UNIQUE (email);

-- Composite unique constraint
ALTER TABLE employees
ADD CONSTRAINT unique_name_dept UNIQUE (name,
department);
```

Check Constraints: `CHECK`

Enforce business rules and data validation.

```
-- Simple check constraint
ALTER TABLE employees
ADD CONSTRAINT check_salary CHECK (salary > 0);

-- Complex check constraint
ALTER TABLE employees
ADD CONSTRAINT check_age
CHECK (YEAR(CURDATE()) - YEAR(birth_date) >= 18);
```

Database Performance & Optimization

Optimize database queries and improve system performance.

Indexes: `CREATE INDEX`

Speed up data retrieval with database indexes.

```
-- Create index on single column
CREATE INDEX idx_employee_name ON
employees(name);

-- Composite index
CREATE INDEX idx_dept_salary ON
employees(department, salary);

-- Unique index
CREATE UNIQUE INDEX idx_employee_email ON
employees(email);

-- Show table indexes
SHOW INDEX FROM employees;
```

Query Optimization: `EXPLAIN`

Analyze and optimize query performance.

```
-- Analyze query execution plan
EXPLAIN SELECT * FROM employees WHERE salary >
75000;

-- Detailed analysis
EXPLAIN ANALYZE SELECT e.name, d.department_name
FROM employees e
JOIN departments d ON e.department = d.id;
```

Performance Monitoring

Monitor database performance and identify bottlenecks.

```
-- Show running processes
SHOW PROCESSLIST;

-- Show database status
SHOW STATUS LIKE 'Slow_queries';

-- Query cache information
SHOW STATUS LIKE 'Qcache%';

-- Table optimization
OPTIMIZE TABLE employees;
```

Database Maintenance

Regular maintenance tasks for optimal performance.

```
-- Analyze table statistics
ANALYZE TABLE employees;

-- Check table integrity
CHECK TABLE employees;

-- Repair table if needed
REPAIR TABLE employees;
```

Data Import/Export

Transfer data between different systems and formats.

Import Data: `LOAD DATA`

Import data from external files into database tables.

```
-- Import from CSV file
LOAD DATA INFILE 'employees.csv'
INTO TABLE employees
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

-- Import with column mapping
LOAD DATA INFILE 'data.csv'
INTO TABLE employees (name, email, salary);
```

Export Data: `SELECT INTO`

Export query results to external files.

```
-- Export to CSV file
SELECT name, email, salary
```

Data Migration: Between Databases

Move data between different database systems.

```
-- Create table from existing structure
CREATE TABLE employees_backup LIKE employees;

-- Copy data between tables
INSERT INTO employees_backup SELECT * FROM
employees;

-- Migrate with conditions
INSERT INTO new_employees
SELECT * FROM old_employees WHERE active = 1;
```

Bulk Operations

Handle large-scale data operations efficiently.

```
-- Bulk insert with INSERT IGNORE
INSERT IGNORE INTO employees (name, email) VALUES
```

```
INTO OUTFILE 'employee_export.csv'
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
FROM employees;

-- Export with mysqldump
mysqldump -u username -p --tab=/path/to/export database_name table_name
```

```
('John Doe', 'john@email.com'),
('Jane Smith', 'jane@email.com');

-- Batch updates
UPDATE employees SET salary = salary * 1.1 WHERE department = 'Sales';
```

Database Security & Access Control

Secure your database and manage user permissions.

User Management: `CREATE USER`

Create and manage database user accounts.

```
-- Create user with password
CREATE USER 'app_user'@'localhost' IDENTIFIED BY 'secure_password';

-- Create user for specific host
CREATE USER 'remote_user'@'192.168.1.%' IDENTIFIED BY 'password';

-- Drop user
DROP USER 'old_user'@'localhost';
```

Database Roles

Organize permissions using database roles.

```
-- Create role (MySQL 8.0+)
CREATE ROLE 'app_read_role', 'app_write_role';

-- Grant privileges to role
GRANT SELECT ON company_db.* TO 'app_read_role';
GRANT INSERT, UPDATE, DELETE ON company_db.* TO 'app_write_role';

-- Assign role to user
GRANT 'app_read_role' TO 'readonly_user'@'localhost';
```

Permissions: `GRANT` & `REVOKE`

Control access to database objects and operations.

```
-- Grant specific privileges
GRANT SELECT, INSERT ON company_db.employees TO 'app_user'@'localhost';

-- Grant all privileges
GRANT ALL PRIVILEGES ON company_db.* TO 'admin_user'@'localhost';

-- Revoke privileges
REVOKE INSERT ON company_db.employees FROM 'app_user'@'localhost';

-- Show user grants
SHOW GRANTS FOR 'app_user'@'localhost';
```

SQL Injection Prevention

Protect against common security vulnerabilities.

```
-- Use prepared statements (application level)
-- Bad: SELECT * FROM users WHERE id = ' + userInput
-- Good: Use parameterized queries

-- Validate input data types
-- Use stored procedures when possible
-- Apply principle of least privilege
```

Database Installation & Setup

Install and configure database management systems.

MySQL Installation

Popular open-source relational database.

```
-- Ubuntu/Debian
sudo apt update
sudo apt install mysql-server

-- Start MySQL service
sudo systemctl start mysql
sudo systemctl enable mysql

-- Secure installation
sudo mysql_secure_installation
```

PostgreSQL Installation

Advanced open-source relational database.

```
-- Ubuntu/Debian
sudo apt update
sudo apt install postgresql postgresql-contrib

-- Switch to postgres user
sudo -u postgres psql

-- Create database and user
CREATE DATABASE myapp;
CREATE USER myuser WITH PASSWORD 'mypassword';
```

SQLite Setup

Lightweight file-based database.

```
-- Install SQLite
sudo apt install sqlite3

-- Create database file
sqlite3 mydatabase.db

-- Basic SQLite commands
.help
.tables
.schema tablename
.quit
```

Database Configuration & Tuning

Configure database settings for optimal performance.

MySQL Configuration

Key MySQL configuration parameters.

```
-- my.cnf configuration file
[mysqld]
innodb_buffer_pool_size = 1G
max_connections = 200
query_cache_size = 64M
tmp_table_size = 64M
max_heap_table_size = 64M

-- Show current settings
SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
SHOW STATUS LIKE 'Connections';
```

Backup Configuration

Set up automated database backups.

```
-- Automated backup script
#!/bin/bash
DATE=$(date +%Y%m%d_%H%M%S)
mysqldump -u backup_user -p mydatabase > backup_$DATE.sql

-- Schedule with cron
0 2 * * * /path/to/backup_script.sh

-- Point-in-time recovery setup
SET GLOBAL log_bin = ON;
```

Connection Management

Manage database connections and pooling.

```
-- Show current connections
SHOW PROCESSLIST;

-- Kill specific connection
KILL CONNECTION 123;

-- Connection timeout settings
SET SESSION wait_timeout = 600;
SET SESSION interactive_timeout = 600;
```

Monitoring & Logging

Monitor database activity and performance.

```
-- Enable slow query log
SET GLOBAL slow_query_log = 'ON';
SET GLOBAL long_query_time = 2;

-- Show database size
SELECT
    table_schema AS 'Database',
    SUM(data_length + index_length) / 1024 / 1024 AS 'Size (MB)'
FROM information_schema.tables
GROUP BY table_schema;
```

SQL Best Practices

Follow these guidelines for writing efficient and maintainable SQL code.

Query Writing Best Practices

Write clean, efficient, and readable SQL queries.

```
-- Use meaningful table aliases
SELECT e.name, d.department_name
FROM employees e
JOIN departments d ON e.dept_id = d.id;

-- Specify column names instead of SELECT *
SELECT name, email, salary FROM employees;

-- Use appropriate data types
CREATE TABLE products (
    id INT PRIMARY KEY,
    price DECIMAL(10,2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Performance Optimization Tips

Optimize queries for better database performance.

```
-- Use indexes on frequently queried columns
CREATE INDEX idx_employee_dept ON employees(department);

-- Limit result sets when possible
SELECT name FROM employees WHERE active = 1 LIMIT 100;

-- Use EXISTS instead of IN for subqueries
SELECT * FROM customers c
WHERE EXISTS (SELECT 1 FROM orders o WHERE o.customer_id = c.id);
```

Reference: This cheatsheet covers essential database commands and best practices for efficient database management and SQL query development in modern applications.