MySQL Cheatsheet: Beginner to Advanced

A Comprehensive Guide to MySQL Queries and Concepts

Author: Eftekher Ali Efte

LinkedIn: Eftekher Ali Efte

July 11, 2025

Contents

1 Introduction to MySQL

1.1 What is MySQL?

MySQL is an open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) to manage and manipulate data. This cheatsheet covers MySQL commands from basic to advanced, including examples, inputs, and outputs.

1.2 Key Concepts

- **Tables**: Store data in rows and columns.
- **Primary Key**: Uniquely identifies each row.
- **Foreign Key**: Links tables to enforce referential integrity.
- **Joins**: Combine data from multiple tables.
- **Aggregation Functions**: Summarize data (e.g., COUNT, SUM).

2 Database and Table Management

2.1 Creating a Database

```sql
CREATE DATABASE school;
USE school;
```

Listing 1: Create a Database

Output: Database ‘school‘ created and selected.

2.2 Creating a Table with Primary Key

```sql
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT,
    grade VARCHAR(2)
);
```

Listing 2: Create Table with Primary Key

Output: Table ‘students‘ created with ‘student$_id‘astheprimarykey.$

2.3 Altering a Table

```sql
ALTER TABLE students ADD email VARCHAR(100);
```

Listing 3: Add a Column

Output: Column ‘email‘ added to ‘students‘ table.

2.4 Renaming a Table

```
RENAME TABLE students TO pupils;
```

Listing 4: Rename Table

Output: Table ‘students‘ renamed to ‘pupils‘.

2.5 Dropping a Table or Database

```
DROP TABLE pupils;
DROP DATABASE school;
```

Listing 5: Drop Table and Database

Output: Table ‘pupils‘ and database ‘school‘ deleted.

3 Data Manipulation

3.1 Inserting Data

```
INSERT INTO students (student_id, name, age, grade, email)
VALUES (1, ’John Doe’, 15, ’A’, ’john.doe@email.com’);
```

Listing 6: Insert Single Row

Output: One row inserted into ‘students‘.

```
INSERT INTO students (student_id, name, age, grade, email)
VALUES
    (2, ’Jane Smith’, 16, ’B’, ’jane.smith@email.com’),
    (3, ’Alice Brown’, 15, ’A’, ’alice.brown@email.com’);
```

Listing 7: Insert Multiple Rows

Output: Two rows inserted.

3.2 Selecting Data

```
SELECT * FROM students;
```

Listing 8: Select All Columns

Output:

student$_{i}d$	name	age	grade	email
1	John Doe	15	A	john.doe@email.com
2	Jane Smith	16	B	jane.smith@email.com
3	Alice Brown	15	A	alice.brown@email.com

```
SELECT name, grade FROM students WHERE age = 15;
```

Listing 9: Select Specific Columns with WHERE

Output:

name	grade
John Doe	A
Alice Brown	A

3.3 Updating Data

```
UPDATE students SET grade = 'A+' WHERE age = 15;
```

Listing 10: Update Rows

Output: Rows with ‘age = 15‘ updated to ‘grade = ’A+’‘.

3.4 Deleting Data

```
DELETE FROM students WHERE student_id = 2;
```

Listing 11: Delete Rows

Output: Row with ‘student$_{i}d = 2‘deleted.$

4 Joins

4.1 Creating a Related Table with Foreign Key

```
CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(50),
    student_id INT,
    FOREIGN KEY (student_id) REFERENCES students(student_id)
);
INSERT INTO courses VALUES
    (101, 'Math', 1),
    (102, 'Science', 1),
    (103, 'History', 3);
```

Listing 12: Create Table with Foreign Key

Output: Table ‘courses‘ created, three rows inserted.

4.2 Inner Join

```
SELECT students.name, courses.course_name
FROM students
INNER JOIN courses ON students.student_id = courses.student_id;
```

Listing 13: Inner Join

Output:

name	course$_{n}ame$
John Doe	Math
John Doe	Science
Alice Brown	History

4.3 Left Join

```sql
SELECT students.name, courses.course_name
FROM students
LEFT JOIN courses ON students.student_id = courses.student_id;
```

Listing 14: Left Join

Output:

name	course$_name$
John Doe	Math
John Doe	Science
Jane Smith	NULL
Alice Brown	History

4.4 Right Join

```sql
SELECT students.name, courses.course_name
FROM students
RIGHT JOIN courses ON students.student_id = courses.student_id;
```

Listing 15: Right Join

Output:

name	course$_name$
John Doe	Math
John Doe	Science
Alice Brown	History

4.5 Union

```sql
SELECT name FROM students WHERE age = 15
UNION
SELECT name FROM students WHERE grade = 'B';
```

Listing 16: Union

Output:

name
John Doe
Alice Brown
Jane Smith

5 Aggregation Functions

5.1 Count

```sql
SELECT COUNT(*) AS total_students FROM students;
```

Listing 17: Count Rows

Output:

total$_students$
3

5.2 Sum, Average, Min, Max

```
SELECT
    SUM(age) AS total_age,
    AVG(age) AS avg_age,
    MIN(age) AS min_age,
    MAX(age) AS max_age
FROM students;
```

Listing 18: Aggregation Functions

Output:

total$_age$	avg$_age$	min$_age$	max$_age$
46	15.33	15	16

5.3 Group By

```
SELECT grade, COUNT(*) AS count
FROM students
GROUP BY grade;
```

Listing 19: Group By with Aggregation

Output:

grade	count
A+	2
B	1

5.4 Having

```
SELECT grade, COUNT(*) AS count
FROM students
GROUP BY grade
HAVING count > 1;
```

Listing 20: Having Clause

Output:

grade	count
A+	2

6 Advanced Topics

6.1 Subqueries

```
SELECT name
FROM students
WHERE student_id IN (
    SELECT student_id
    FROM courses
    WHERE course_name = 'Math'
);
```

Listing 21: Subquery

Output:

name
John Doe

6.2 Case Statements

```
SELECT name,
    CASE
        WHEN age >= 16 THEN 'Senior'
        ELSE 'Junior'
    END AS student_level
FROM students;
```

Listing 22: Case Statement

Output:

name	student$_levell$
John Doe	Junior
Jane Smith	Senior
Alice Brown	Junior

6.3 Indexes

```
CREATE INDEX idx_name ON students(name);
```

Listing 23: Create Index

Output: Index ‘idx$_name‘createdon‘name‘column.$

6.4 Views

```
CREATE VIEW student_courses AS
SELECT students.name, courses.course_name
FROM students
INNER JOIN courses ON students.student_id = courses.student_id;
```

Listing 24: Create View

Output: View ‘student$_courses‘created.$

6.5 Stored Procedures

```
DELIMITER //
CREATE PROCEDURE GetStudentCount()
BEGIN
    SELECT COUNT(*) AS total_students FROM students;
END //
DELIMITER ;
CALL GetStudentCount();
```

Listing 25: Stored Procedure

Output:

total$_students$
3

6.6 Triggers

```
DELIMITER //
CREATE TRIGGER before_student_insert
BEFORE INSERT ON students
FOR EACH ROW
BEGIN
    IF NEW.age < 0 THEN
        SIGNAL SQLSTATE ’45000’
        SET MESSAGE_TEXT = ’Age cannot be negative’;
    END IF;
END //
DELIMITER ;
```

Listing 26: Create Trigger

Output: Trigger ‘before$_{s}tudent_{i}nsert‘createdtopreventnegativeages.$

6.7 Transactions

```
START TRANSACTION;
INSERT INTO students (student_id, name, age, grade)
VALUES (4, ’Bob Wilson’, 14, ’C’);
UPDATE students SET grade = ’B’ WHERE student_id = 4;
COMMIT;
```

Listing 27: Transaction

Output: Row inserted and updated within a transaction.

7 Constraints

7.1 Primary Key

Ensures each row is uniquely identifiable.

```
CREATE TABLE employees (
    emp_id INT PRIMARY KEY,
    name VARCHAR(50)
);
```

Listing 28: Primary Key Constraint

Output: Table ‘employees‘ created with ‘emp$_{i}d‘asprimarykey.$

7.2 Foreign Key

Enforces referential integrity between tables.

```
CREATE TABLE enrollments (
    enrollment_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
);
```

Listing 29: Foreign Key Constraint

Output: Table ‘enrollments‘ created with foreign keys.

7.3 Unique Constraint

Ensures all values in a column are unique.

```
ALTER TABLE students ADD CONSTRAINT unique_email UNIQUE (email);
```

Listing 30: Unique Constraint

Output: Unique constraint added to ‘email‘ column.

7.4 Check Constraint

Enforces a condition on column values.

```
ALTER TABLE students ADD CONSTRAINT check_age CHECK (age >= 0);
```

Listing 31: Check Constraint

Output: Check constraint added to ensure ‘age‘ is non-negative.

8 Query Optimization

8.1 Explain Plan

```
EXPLAIN SELECT * FROM students WHERE age = 15;
```

Listing 32: Explain Plan

Output: Displays query execution plan for optimization analysis.

8.2 Using Indexes in Queries

```
SELECT name FROM students WHERE name = 'John Doe';
```

Listing 33: Using Index

Output: Query uses ‘idx$_{n}ame‘indexforfasterexecution.$

8.3 Limiting Results

```
SELECT * FROM students ORDER BY age LIMIT 2;
```

Listing 34: Limit Clause

Output:

student$_id$	name	age	grade	email
1	John Doe	15	A+	john.doe@email.com
3	Alice Brown	15	A+	alice.brown@email.com

9 Window Functions

9.1 Row Number

```
SELECT name, age,
    ROW_NUMBER() OVER (ORDER BY age) AS row_num
FROM students;
```

Listing 35: Row Number

Output:

name	age	row$_num$
John Doe	15	1
Alice Brown	15	2
Jane Smith	16	3

9.2 Rank and Dense Rank

```
SELECT name, age,
    RANK() OVER (ORDER BY age) AS rank,
    DENSE_RANK() OVER (ORDER BY age) AS dense_rank
FROM students;
```

Listing 36: Rank and Dense Rank

Output:

name	age	rank	dense$_rank$
John Doe	15	1	1
Alice Brown	15	1	1
Jane Smith	16	3	2

9.3 Partition By

```
SELECT name, grade, age,
    AVG(age) OVER (PARTITION BY grade) AS avg_age_by_grade
FROM students;
```

Listing 37: Partition By

Output:

name	grade	age	avg$_age_by_grade$
John Doe	A+	15	15
Alice Brown	A+	15	15
Jane Smith	B	16	16