Performance Best Practices

Mosh Hamedani

Code with Mosh (codewithmosh.com)

1st Edition

About the Author

Hi! My name is Mosh Hamedani. I'm a software engineer with two decades of experience. I've taught over three million students how to code and become professional software engineers. It's my mission to make software engineering simple and accessible for everyone.

https://codewithmosh.com

https://youtube.com/user/programmingwithmosh

https://twitter.com/moshhamedani

https://facebook.com/programmingwithmosh/

Here is a summary of all the performance best practices we discussed in this course. This list is not complete by any means, but it contains guidelines that can help solve a lot of performance problems. If you're interested in learning more about this topic, take a look at the additional resources later in this document.

1. Smaller tables perform better. Don't store the data you don't need. Solve today's problems, not tomorrow's future problems that may never happen.

2. Use the smallest data types possible. If you need to store people's age, a TINYINT is sufficient. No need to use an INT. Saving a few bytes is not a big deal in a small table, but has a significant impact in a table with millions of records.

3. Every table must have a primary key.

4. Primary keys should be short. Prefer TINYINT to INT if you only need to store a hundred records.

5. Prefer numeric types to strings for primary keys. This makes looking up records by the primary key faster.

6. Avoid BLOBs. They increase the size of your database and have a negative impact on the performance. Store your files on disk if you can.

7. If a table has too many columns, consider splitting it into two related tables using a one-to-one relationship. This is called *vertical partitioning*. For example, you may have a customers table with columns for storing their address. If these columns don't get read often, split the table into two tables (users and user_addresses).

8. In contrast, if you have several joins in your queries due to data fragmentation, you may want to consider denormalizing data. Denormalizing is the opposite of normalization. It involves duplicating a column from one table in another table (to reduce the number of joins) required.

9. Consider creating summary/cache tables for expensive queries. For example, if the query to fetch the list of forums and the number of posts in each forum is expensive, create a table called forums_summary that contains the list of forums and the number of posts in them. You can use events to regularly refresh the data in this table. You may also use triggers to update the counts every time there is a new post.

10. Full table scans are a major cause of slow queries. Use the EXPLAIN statement and look for queries with type = ALL. These are full table scans. Use indexes to optimize these queries.

11. When designing indexes, look at the columns in your WHERE clauses first. Those are the first candidates because they help narrow down the searches. Next, look at the columns used in the ORDER BY clauses. If they exist in the index, MySQL can scan your index to return ordered data without having to perform a sort operation (filesort). Finally, consider adding the columns in the SELECT clause to your indexes. This gives you a *covering* index that covers everything your query needs. MySQL doesn't need to retrieve anything from your tables.

12. Prefer composite indexes to several single-column index.

13. The order of columns in indexes matter. Put the most frequently used columns and the columns with a higher cardinality first, but always take your queries into account.

14. Remove duplicate, redundant and unused indexes. Duplicate indexes are the indexes on the same set of columns with the same order. Redundant indexes are unnecessary indexes that can be replaced with the existing indexes. For example, if you have an index on columns (A, B) and create another index on column (A), the latter is redundant because the former index can help.

15. Don't create a new index before analyzing the existing ones.

16. Isolate your columns in your queries so MySQL can use your indexes.

17. Avoid SELECT *. Most of the time, selecting all columns ignores your indexes and returns unnecessary columns you may not need. This puts an extra load on your database server.

18. Return only the rows you need. Use the LIMIT clause to limit the number of rows returned.

19. Avoid LIKE expressions with a leading wildcard (eg LIKE '%name').

20. If you have a slow query that uses the OR operator, consider chopping up the query into two queries that utilize separate indexes and combine them using the UNION operator.

Additional Reading

High Performance MySQL: Optimization, Backups, and Replication by Baron Schwartz

Relational Database Index Design and the Optimizers by Tapio Lahdenmaki

This document shows a sample ticket generated by an imaginary flight booking system.
Design a database for this system to record the necessary data.

Los Angeles, CA -> San Francisco, CA
San Francisco, CA -> Los Angeles, CA
Fri Apr 05 2019 -> Sun Apr 07 2019
2 Tickets

Airline Confirmation Numbers
Alaska Airline: TAEGKX

Passengers and Ticket Numbers:

John Smith
Ticket Number: 0177200658

Jennifer Smith
Ticket Number: 0178410326

Fri Apr 05
Los Angeles -> San Francisco
LAX -> SFO
08:20 AM - 09:35 AM
Alaska Airlines Flight 1490
1h 15m, 236 miles

Depart: Los Angeles Intl Airport (LAX)
Arrive: San Francisco Intl Airport (SFO)

Economy Class

Sun Apr 07
San Francisco -> Los Angeles
SFO -> LAX
02:00 PM - 03:15 PM
Alaska Airlines Flight 1473
1h 15m, 236 miles

Depart: San Francisco Intl Airport (SFO)
Arrive: Los Angeles Intl Airport (LAX)

Economy Class

Price Summary

Traveler 1: $357.60
Traveler 2: $357.60
Total: $756.20

Requirements

We're going to build a desktop application called Vidly. This application will be used at a video rental store. We need different levels of permissions for different users.

The store manager should be able to add/update/delete the list of movies. They will be in charge of setting the stock for each movie as well as the daily rental rate.
Cashiers should have a read-only view of the list of movies. They should be able to manage the list of customers and the movies they rent.

At check out, a customer brings one or more movies. The cashier looks up a customer by their phone number. If the customer is a first-time customer, the cashier asks their full name, email and phone number, and then registers them in the system. The cashier then scans the movies the customer has brought to check out and records them in the system. Each movie has a 10 digit barcode printed on the cover.

When the customer returns to the store, they'll bring the movies they rented. If a movie is lost, the customer should be charged 5 times the daily rental rate of the movie. The cashier should mark the movie as lost and this will reduce the stock. There is no need to keep track of the lost movies. All we need to know is the the number of movies in stock and how much the customer was charged.

For other movies, the customer should be charged based on the number of days and the daily rental rate.

We issue discount coupons from time to time. The customer can bring a coupon when returning the movies.

It is possible that a customer returns the movies they've rented in multiple visits.

We need to be able to track the
- top movies
- top customers
- revenue per day, month and year

SQL Cheat Sheet

Mosh Hamedani

Code with Mosh (codewithmosh.com)

1st Edition

About this Cheat Sheet

This cheat sheet includes the materials I've covered in my SQL tutorial for Beginners on YouTube.

https://youtu.be/7S_tz1z_5bA

Both the YouTube tutorial and this cheat cover the core language constructs and they are not complete by any means.

If you want to learn everything SQL has to offer and become a SQL expert, check out my Complete SQL Mastery Course.

Use the **coupon code CHEATSHEET** upon checkout to get this course with a 90% discount:

https://codewithmosh.com/p/complete-sql-mastery/

About the Author

Hi! My name is Mosh Hamedani. I'm a software engineer with two decades of experience and I've taught over three million how to code or how to become a professional software engineer. It's my mission to make software engineering simple and accessible to everyone.

https://codewithmosh.com

https://youtube.com/user/programmingwithmosh

https://twitter.com/moshhamedani

https://facebook.com/programmingwithmosh/

Basics

```
USE sql_store;

SELECT *
FROM customers
WHERE state = 'CA'
ORDER BY first_name
LIMIT 3;
```

- SQL is **not** a case-sensitive language.
- In MySQL, every statement must be terminated with a semicolon.

Comments

We use comments to add notes to our code.

```
-- This is a comment and it won't get executed.
```

SELECT Clause

```
-- Using expressions

SELECT (points * 10 + 20) AS discount_factor
FROM customers
```

Order of operations:

- Parenthesis
- Multiplication / division
- Addition / subtraction

```
-- Removing duplicates

SELECT DISTINCT state
FROM customers
```

WHERE Clause

We use the WHERE clause to filter data.

Comparison operators:

- Greater than: >
- Greater than or equal to: >=
- Less than: <
- Less than or equal to: <=
- Equal: =
- Not equal: <>
- Not equal: !=

Logical Operators

```
-- AND (both conditions must be True)
SELECT *
FROM customers
WHERE birthdate > '1990-01-01' AND points > 1000


-- OR (at least one condition must be True)
SELECT *
FROM customers
WHERE birthdate > '1990-01-01' OR points > 1000

-- NOT (to negate a condition)
SELECT *
FROM customers
WHERE NOT (birthdate > '1990-01-01')
```

IN Operator

```
-- Returns customers in any of these states: VA, NY, CA
SELECT *
FROM customers
WHERE state IN ('VA', 'NY', 'CA')
```

BETWEEN Operator

```
SELECT *
FROM customers
WHERE points BETWEEN 100 AND 200
```

LIKE Operator

```
-- Returns customers whose first name starts with b
SELECT *
FROM customers
WHERE first_name LIKE 'b%'
```

- %: any number of characters
- _: exactly one character

REGEXP Operator

```
-- Returns customers whose first name starts with a
SELECT *
FROM customers
WHERE first_name REGEXP '^a'
```

- ^: beginning of a string
- $: end of a string
- |: logical OR
- [abc]: match any single characters
- [a-d]: any characters from a to d

More Examples

```
-- Returns customers whose first name ends with EY or ON
WHERE first_name REGEXP 'ey$|on$'

-- Returns customers whose first name starts with MY
-- or contains SE
WHERE first_name REGEXP '^my|se'

-- Returns customers whose first name contains B followed by
-- R or U
WHERE first_name REGEXP 'b[ru]'
```

IS NULL Operator

```
-- Returns customers who don't have a phone number
SELECT *
FROM customers
WHERE phone IS NULL
```

ORDER BY Clause

```
-- Sort customers by state (in ascending order), and then
-- by their first name (in descending order)
SELECT *
FROM customers
ORDER BY state, first_name DESC
```

LIMIT Clause

```
-- Return only 3 customers
SELECT *
FROM customers
LIMIT 3
```

```sql
-- Skip 6 customers and return 3
SELECT *
FROM customers
LIMIT 6, 3
```

Inner Joins

```sql
SELECT *
FROM customers c
JOIN orders o
   ON c.customer_id = o.customer_id
```

Outer Joins

```sql
-- Return all customers whether they have any orders or not
SELECT *
FROM customers c
LEFT JOIN orders o
   ON c.customer_id = o.customer_id
```

USING Clause

If column names are exactly the same, you can simplify the join with the USING clause.

```sql
SELECT *
FROM customers c
JOIN orders o
   USING (customer_id)
```

Cross Joins

```sql
-- Combine every color with every size
SELECT *
FROM colors
CROSS JOIN sizes
```

Unions

```
-- Combine records from multiple result sets
SELECT name, address
FROM customers
UNION
SELECT name, address
FROM clients
```

Inserting Data

```
-- Insert a single record
INSERT INTO customers(first_name, phone, points)
VALUES ('Mosh', NULL, DEFAULT)
```

```
-- Insert multiple single records
INSERT INTO customers(first_name, phone, points)
VALUES
    ('Mosh', NULL, DEFAULT),
    ('Bob', '1234', 10)
```

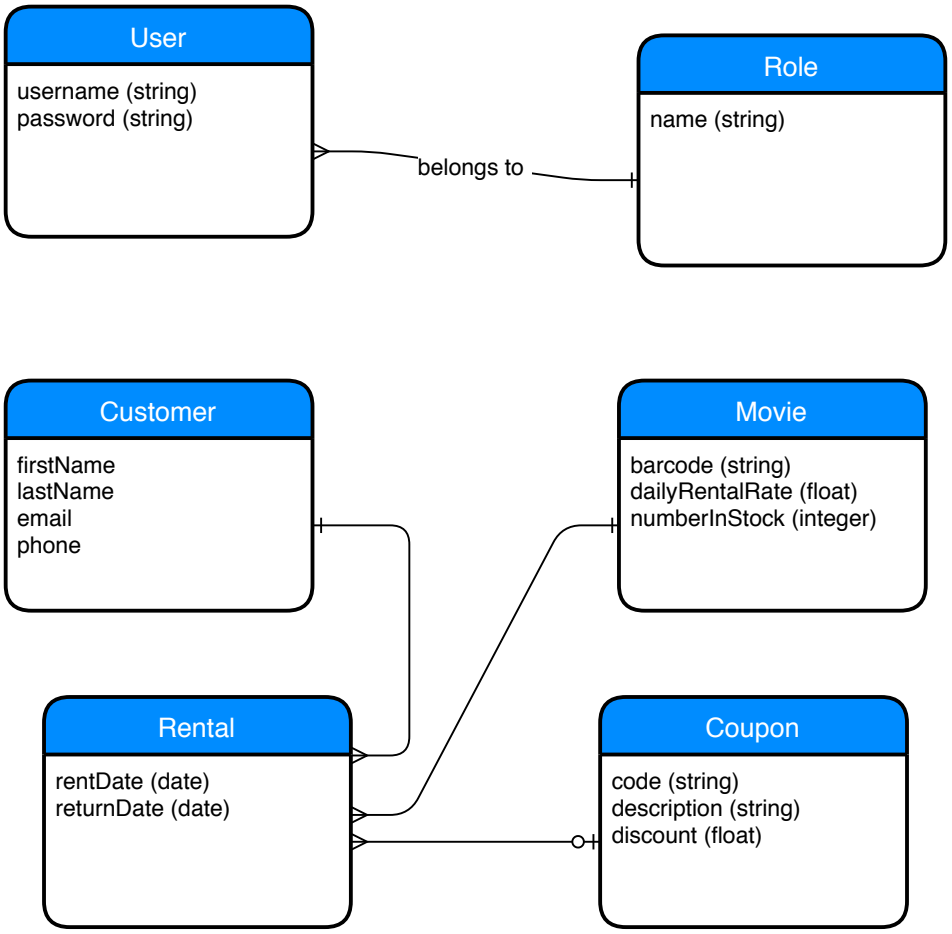
Want to Become a SQL Expert?

If you're serious about learning SQL and getting a job as a software developer or data scientist, I highly encourage you to enroll in my Complete SQL Mastery Course. Don't waste your time following disconnected, outdated tutorials. My Complete SQL Mastery Course has everything you need in one place:

- 10 hours of HD video
- Unlimited access - watch it as many times as you want
- Self-paced learning - take your time if you prefer
- Watch it online or download and watch offline
- Certificate of completion - add it to your resume to stand out
- 30-day money-back guarantee - no questions asked

The price for this course is $149 but the first 200 people who have downloaded this cheat sheet can get it for $12.99 using the coupon code **CHEATSHEET**:

https://codewithmosh.com/p/complete-sql-mastery/


User
username (string)
password (string)
belongs to
Role
name (string)
Customer
firstName
lastName
email
phone
Movie
barcode (string)
dailyRentalRate (float)
numberInStock (integer)
Rental
rentDate (date)
returnDate (date)
Coupon
code (string)
description (string)
discount (float)