Top 34 MySQL Interview Questions and Answers For 2025

Master MySQL with this guide to interview questions, with real-world examples and expert tips to help you excel in your next database interview!

Dec 25, 2024 · 15 min read

Contents

Have you ever noticed that MySQL is required in almost every database-related job description? There's a good reason for that — MySQL pretty much powers everything from your favorite social media platforms to the apps you use daily.

I've put together this guide to help you tackle MySQL interview questions. I'll cover all the bases, from the basics that junior developers should know to the complex stuff that senior roles require. Also, I'll share some tips to help you appear as a confident candidate in your next data-related interviews.

What is MySQL?

MySQL is an open-source RDBMS (relational database management system) built on SQL that organizes data into structured tables. It was developed by Oracle Corporation.

It ranked as the **most popular DBMS in 2024** because more and more businesses are migrating their data to MySQL. That's why if you're planning to get a database job, it's so important to have the know-how of MySQL and relational databases.

As of June 2024, MySQL is the world's most popular open-source database management system (DBMS), with a ranking score of 1061. Source: Statista.

Basic MySQL Interview Questions

In the initial interview phase, the interviewer can ask foundational questions to assess your understanding of basic database and MySQL concepts.

1. What is a database, and how is it different from a DBMS?

A database is a storage container that holds data we can access, modify, and analyze. For example, social media platforms store data about who liked our posts in databases.

A DBMS (Database Management System) is the software that lets us interact with and manage that data by creating users and managing their access. MySQL is one of the most popular DBMS options. Other examples include **PostgreSQL**, **MongoDB**, and **Microsoft SQL Server**.

2. How does MySQL differ from other relational database management systems?

MySQL is an open-source relational database management system (RDBMS) that uses SQL to manage data. It's known for its ease of use, speed, and compatibility with web-based applications.

Here's how MySQL differs from other RDBMSs:

- **Simplicity and performance:** MySQL is often praised for its simplicity and optimized performance, making it a go-to choice for web developers and startups.
- **Advanced features:** While MySQL excels in ease of use, it may lack advanced features in other RDBMSs like PostgreSQL, such as more comprehensive support for ACID transactions, advanced indexing, and a broader set of data types.
- **Storage engines:** MySQL allows you to choose different storage engines (e.g., InnoDB, MyISAM) for tables, giving them flexibility for specific use cases.

MySQL is ideal for scenarios requiring speed and scalability, but for more complex or enterprise-grade features, PostgreSQL might be a better choice.

3. What are the main data types available in MySQL?

MySQL supports a variety of data types categorized as:

- **Numeric:** INT, DECIMAL, FLOAT, DOUBLE, etc.
- **String:** CHAR, VARCHAR, TEXT, BLOB.
- **Date/time:** DATE, DATETIME, TIMESTAMP, TIME.
- **JSON:** For storing JSON objects.

4. What is the difference between INT and DECIMAL data types?

INT stores whole numbers with no decimal points. We can use it if there is no need for fractions. On the contrary, DECIMAL can store financial values and is suitable for precise calculations with decimals.

5. How is DATE different from DATETIME in MySQL?

The DATE function in MySQL stores the date in year, month, and day format:

YYYY-MM-DD

However, the DATETIME function stores the date with the time, and it looks like this:

YYYY-MM-DD HH:MM:SS

6. What is a foreign key, and how would you use it in databases?

A foreign key is a field in one table that links to the primary key of another table.

For example, in a customers table that stores customer information, each customer has a unique customer_id—in another table called transactions (which stores purchase records), we use customer_id as a foreign key.
The customer_id in the transactions table will link each purchase to a specific customer in the customers table.

Here’s how that looks in SQL:

```
CREATE TABLE customers (

  customer_id INT PRIMARY KEY,

  name VARCHAR(100),

  email VARCHAR(100)

);

CREATE TABLE transactions (

  transaction_id INT PRIMARY KEY,

  customer_id INT,

  amount DECIMAL(10,2),

  date DATE,

  FOREIGN KEY (customer_id) REFERENCES customers(customer_id)

);
```

Powered By

7. What are the differences between INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN?

Joins combine rows from two or more tables based on related columns. Here are their differences:

- **INNER JOIN:** Returns rows where there is a match in both tables.
- **LEFT JOIN:** Returns all rows from the left table and matching rows from the right table. If there’s no match, NULL is returned for the right table's columns.
- **RIGHT JOIN:** Similar to LEFT JOIN, it returns all rows from the right table and matching rows from the left.
- **FULL JOIN:** Combines the LEFT JOIN and RIGHT JOIN results, including unmatched rows from both tables.

8. What is the difference between DELETE, TRUNCATE, and DROP in MySQL?

Commands like DELETE, TRUNCATE, and DROP may sound similar, but they actually have different behavior:

- **DELETE:** Removes rows from a table based on a condition. It can be rolled back if inside a transaction. Example:

```
DELETE FROM employees WHERE department_id = 5;
```

Powered By

- **TRUNCATE:** Deletes all rows from a table, but the table structure remains intact. It is faster than DELETE and cannot be rolled back. Example:

```
TRUNCATE TABLE employees;
```

Powered By

- **DROP:** Completely removes the table structure and data, along with any dependencies like indexes. Example:

```
DROP TABLE employees;
```

Powered By

9. How do you create and modify a table in MySQL? Provide examples.

For creating tables, you can use the CREATE TABLE statement, and for modifying, usually the ALTER TABLE. Here are some examples:

- Create table:

```
CREATE TABLE employees (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(50), department VARCHAR(50));
```

Powered By

- Modify to add a column:

```
ALTER TABLE employees ADD COLUMN salary DECIMAL(10, 2);
```

Powered By

10. What is a temporary table in SQL?

A temporary table only exists during the current database session. Once we close the session, the table is deleted. This type of table can temporarily store intermediate results. We can use it for testing, filtering, or preparing data before inserting it into a permanent table.

Here's an example:

```
CREATE TEMPORARY TABLE temp_employees (
   id INT,
   name VARCHAR(50)
);
INSERT INTO temp_employees VALUES (1, 'John Doe');
SELECT * FROM temp_employees;
```

Powered By

11. What is a subquery in MySQL? Explain with an example.

A subquery (also known as a nested query) is nested inside another query. It breaks down complex database operations into more manageable steps. For example, you can create a subquery to find employees earning above the average pay:

```
SELECT first_name, last_name, salary
FROM employees
WHERE salary > (
   SELECT AVG(salary)
   FROM employees
);
```

Powered By

Let's break this down:

1. The inner query SELECT AVG(salary) FROM employees calculates the average salary first.
2. The outer query then uses this average to find employees who earn above it.

12. How would you use an INSERT statement in MySQL to add data to a table? Also, do you have some best practices for this?

We can use the INSERT() statement to add data to a table. The basic syntax is:

```
INSERT INTO table_name (column1, column2, ...)

VALUES (value1, value2, ...);
```

Powered By

Here are a few best practices you can follow when I use the INSERT() statement:

1. Explicitly list your columns. This makes the code clearer and prevents errors if the table structure changes later.
2. For AUTO_INCREMENT columns like IDs, skip them in the INSERT() statement. MySQL handles these automatically to prevent duplicate IDs.
3. Be consistent with string quotes. I personally prefer single quotes, but either works.
4. If you're inserting multiple rows, you can do it in a single statement for better performance.

13. What is the significance of the AUTO_INCREMENT attribute in MySQL?

The AUTO_INCREMENT attribute in MySQL generates unique, sequential numbers for a column, typically the primary key of a table.

Here's an example of how to create a table with an AUTO_INCREMENT column:

```
CREATE TABLE employees (

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(50),

   department VARCHAR(50)

);
```

Powered By

And to insert rows into it:

```
INSERT INTO employees (name, department) VALUES ('John Doe', 'Sales');

INSERT INTO employees (name, department) VALUES ('Jane Smith', 'Marketing');
```

Powered By

14. What is a view in MySQL?

A view is a saved query that works like a virtual table. With this, we can take a complex query, give it a name, and use it like a table for future queries. This way, you don't have to retype the entire query every time.

For example, to simplify querying employee details along with their department names, you can create a view:

```
CREATE VIEW employee_details AS

SELECT

   e.id,
   e.name,
   d.department_name,
   e.salary
FROM
   employees e
JOIN
   departments d ON e.department_id = d.department_id;
```

Powered By

You can now query the employee_details view just like a table:

```
SELECT * FROM employee_details;
```

Powered By

However, we cannot use views to insert and update the data. Most support the read-only option and prevent users from accessing the database, enhancing data security. Views can sometimes slow down queries, as they run the underlying query every time they are accessed.

SQL Upskilling for Beginners

Gain the SQL skills to interact with and query your data.

Intermediate MySQL Interview Questions

In this section, we will cover intermediate-level topics. These questions are primarily asked to test your knowledge of MySQL data types and structure.

15. What are system-versioned tables, and how do they work?

System-versioned tables maintain a full history of changes made in a table. Since they keep previous versions of each row, we can use them to audit and recover data.

They work by adding two extra columns — StartTime and EndTime — to record when each row is valid. When we insert, update, or delete data, these timestamps will be updated:

- **Insert:** A new row is added with a StartTime set to the current timestamp and an EndTime of 9999-12-31 23:59:59 (or a similar value representing "forever"). This indicates that the row is currently valid.
- **Update:** The original row's EndTime is updated to the current timestamp to mark it as no longer valid. Then, a new row with the updated data is created, with its StartTime set to the current timestamp and EndTime as "forever."
- **Delete:** The EndTime of the existing row is updated to the current timestamp, indicating the row is no longer valid.

Using SQL's FOR SYSTEM_TIME clause, you can query the table to view its state at a specific point in time or over a range of time. For example:

- FOR SYSTEM_TIME AS OF '2024-01-01': Retrieves the table's state as it was on January 1, 2024.
- FOR SYSTEM_TIME BETWEEN '2024-01-01' AND '2024-12-31': Shows all rows valid within this date range.

16. What are MySQL transactions, and how do you use them?

Transactions are a set of operations executed as a single unit. They ensure data integrity by allowing all operations to succeed or fail together.

Here's an example of their usage:

```
START TRANSACTION;

UPDATE accounts SET balance = balance - 500 WHERE account_id = 1;

UPDATE accounts SET balance = balance + 500 WHERE account_id = 2;

COMMIT; -- Saves changes permanently

-- or

ROLLBACK; -- Reverts changes
```

Powered By

17. What is a default constraint in MySQL? How do you set a default value for a column?

A default constraint in MySQL assigns a default value to a column when no explicit value is provided during an INSERT operation. This ensures the column is valid even if the user omits it during data entry.

Here's how to create a table with a default value:

```
CREATE TABLE employees (

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(50),

   status VARCHAR(10) DEFAULT 'active'

);
```

Powered By

Then, you can insert a row without specifying the status:

```
INSERT INTO employees (name) VALUES ('John Doe');
```

Powered By

This approach reduces the likelihood of NULL or invalid data in critical columns and simplifies queries by removing the need to handle default cases explicitly.

Result:

Field	Type	Null	Key	Default	Extra
id	INT	NO	PRI	NULL	AUTO_INCREMENT
name	VARCHAR(50)	YES		NULL	
status	VARCHAR(10)	YES		active	

This command is useful because:

- It helps developers understand the table schema before writing queries.

- It can be used for debugging, especially when working with unfamiliar databases.
- It quickly identifies constraints, such as primary keys or defaults.

19. How would you use string functions in SQL to manage text?

Different string functions in SQL work with names and other text data. For example:

- The LENGTH() function displays the number of characters in a name.
- UPPER() and LOWER() functions convert text to all uppercase or lowercase.
- CONCAT() function joins the first and last name in a column.
- SUBSTRING() extracts specific parts of the text. For example, we can use it to separate the month from the birthdate.

Here's an example query:

```
SELECT
  UPPER(first_name) AS upper_name,
  CONCAT(first_name, ' ', last_name) AS full_name,
  SUBSTRING(birthdate, 6, 2) AS birth_month,
  TRIM(last_name) AS trimmed_last_name,
  REPLACE(first_name, 'a', '@') AS replaced_name
FROM employees;
```

Powered By

This query:

- Converts names to uppercase.
- Combines first and last names into a full name.
- Extracts the birth month from a birthdate column.
- Trims spaces from last names.
- Replaces all occurrences of "a" with "@" in first names.

20. How would you update a specific row in a database with SQL?

You can use the UPDATE statement and the WHERE clause to identify the record you want to change.

For example, if you want to update the genre of the movie "Inception" from 2010 to "Sci-fi," you can use the following query:

```
UPDATE movies
SET genre = 'Sci-Fi'
WHERE movie_title = 'Inception' AND year = 2010;
```

Powered By

Here, UPDATE movies specifies the table we want to update, and the WHERE clause targets the row where the title is "Inception" and the year is "2010".

Advanced MySQL Interview Questions

Advanced interview questions test your ability to manage complex MySQL scenarios and give the interviewer an idea of your decision-making abilities.

21. What is a trigger in MySQL? How do you implement it?

In MySQL, a trigger is a set of actions that run when a database event occurs. Triggers can be configured to execute before or after events like INSERT, UPDATE, or DELETE.

For example, suppose there's a table orders where new orders are added. We can create a trigger that logs every new order into an order_history table:

```
CREATE TRIGGER after_order_insert

AFTER INSERT ON orders

FOR EACH ROW

BEGIN

   INSERT INTO order_history (order_id, action, timestamp)

   VALUES (NEW.order_id, 'inserted', NOW());

END;
```

Powered By

After the trigger runs, the order_history table is automatically updated:

history_id	order_id	action	timestamp
1	1	inserted	2024-12-24 10:00:00
2	2	inserted	2024-12-24 11:00:00

22. Why does adding an index make SQL queries faster?

If there's no index, the database will have to scan each row to find a specific entry. An index acts like a table of contents, allowing the database to access the relevant rows. So, adding an index cuts down the search time and makes queries run faster.

Indexes are typically implemented using data structures like B-trees or hash tables, which allow the database to perform searches, lookups, and range scans efficiently.

Here’s an example of how to create an index:

```
-- Without an index:

SELECT * FROM employees WHERE last_name = 'Smith';

-- Adding an index on the last_name column:

CREATE INDEX idx_last_name ON employees(last_name);

-- With the index, the database can quickly locate rows with 'Smith' in the last_name column.
```

Powered By

Indexes have some drawbacks, for example:

- **Slower write operations:** INSERT, UPDATE, and DELETE operations are slower because the index must be updated each time data changes.
- **Storage cost:** Indexes require additional storage space.

23. What data type do we use for the product's weight and price in an SQL table, and why?

We use the FLOAT or REAL data type to store weight because weight often includes decimal values and accepts a small margin of error. Since these data types can store approximate values, they're more suitable for weight columns.

The DECIMAL data type is commonly used for storing prices because financial values like prices don't accept even a small rounding error. This accuracy comes with the decimal data type only (e.g., DECIMAL(10, 2) for 10 digits, with 2 after the decimal point).

24. How do you find duplicate rows in SQL with a window function?

Here's how you can find duplicates using the ROW_NUMBER() window function:

```
WITH DuplicateCheck AS (
   SELECT product_name,
         category,
         ROW_NUMBER() OVER(
            PARTITION BY product_name, category
            ORDER BY id
         ) AS row_num
   FROM sales
)
SELECT *
FROM DuplicateCheck
WHERE row_num > 1;
```

Powered By

Let's break down how this works:

1. ROW_NUMBER() assigns a number to each row in our result.
2. PARTITION BY groups rows by product_name and category.
3. Within each group, rows are numbered starting from 1.
4. Any row_num greater than 1 indicates a duplicate.

For example, if we have these records:

- Product A, Category X, row_num = 1
- Product A, Category X, row_num = 2 (duplicate)
- Product B, Category Y, row_num = 1

The query will show us the second row since its row_num is greater than 1.

25. How do you create and use a stored procedure with parameters in MySQL? Explain with an example.

We can save and reuse complex queries with stored procedures to make database operations more efficient and maintainable. Let's see how to create and use them with parameters through a practical example.

Suppose we have a student database and want to create a procedure to filter students by age. Here's how we can do it:

First, let's create a simple stored procedure that takes an age parameter:

```
CREATE PROCEDURE get_student_info(IN age INT)

BEGIN

   SELECT * FROM student WHERE student.age = age;

END;
```

[Powered By]

To use this procedure, we simply CALL it with our desired age:

```
CALL get_student_info(21);
```

[Powered By]

We can make our procedures more sophisticated by using output parameters. For example, let's create a procedure that counts students of a specific age:

```
CREATE PROCEDURE count_students_by_age(IN age INT, OUT student_count INT)

BEGIN

   SELECT COUNT(*) INTO student_count FROM students WHERE students.age = age;

END;
```

[Powered By]

To get the result from this procedure:

```
SET @count = 0;

CALL count_students_by_age(21, @count);

SELECT @count AS total_students;
```

[Powered By]

26. Why is referential integrity important in a database?

Referential integrity keeps relationships between tables accurate. When we create a foreign key, it ensures that the values in a table match the unique value in the referenced table.

Here's a practical example: Suppose you're managing an e-commerce database. You have a Customers table and an Orders table. Each order must belong to a real customer. Referential integrity, implemented through foreign keys, enforces this relationship by ensuring:

- You can't create an order for a non-existent customer.
- You can't delete a customer with existing orders (unless you specifically configure what should happen to those orders).
- You can't update a customer's ID if that customer has existing orders.

So when you create a foreign key constraint like this:

```
ALTER TABLE Orders
```

ADD FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID);

Powered By

The database automatically enforces these rules:

- Every CustomerID in the Orders table must exist in the Customers table.
- Attempts to violate these rules (like inserting an invalid CustomerID) will be rejected.

This prevents data inconsistencies that could cause serious problems, such as orders that can't be traced back to actual customers or reports generated with missing customer information.

MySQL Interview Questions for Database Administrators

If you're specifically applying for a database administrator role, here are some questions that recruiters can ask.

27. Why would a large application use database sharding? Also, tell me about the challenges with it.

A large application uses database sharding to divide big data across multiple servers. Each part or shard contains a small part of the data. Since it spreads the data load, there's no need for high-end hardware. Although the speed and scalability also improve, it has some challenges too:

- Some queries like joins may not work, which can complicate data management.
- When data grows, the shards may become overwhelmed, creating hotspots that slow down performance.

28. Explain the role of redo logs in MySQL crash recovery.

Every time data is modified in MySQL, it needs to be written to disk. However, writing directly to data files is slow and risky. That's why before MySQL modifies any data files, it first writes down what it will do in the redo log. This is safer than randomly updating data files.

So, let's say you update a customer's address:

1. MySQL first writes this change to the redo log.
2. It then acknowledges your transaction as committed.
3. Eventually, it applies the change to the actual data files.

Crash recovery is important if MySQL crashes after step 1 or 2 but before step 3. When MySQL restarts, it looks at its redo logs and then completes any unfinished work by replaying the changes recorded in the redo logs. This guarantees that committed transactions aren't lost, even if MySQL crashes at an inconvenient moment.

29. What are the different storage engines available in MySQL, and how do they differ?

MySQL supports multiple storage engines, each optimized for different use cases. Here's a comparison of the most common ones:

Storage engine	Features	Use cases
InnoDB	- Default storage engine. - ACID-compliant. - Row-level locking.	Applications requiring high data integrity, such as e-commerce platforms or financial systems.

	- Supports transactions and foreign keys.	
MyISAM	- Fast for read-heavy operations. - Table-level locking. - No support for transactions or foreign keys.	Read-intensive applications where speed is more critical than data integrity.
Memory	- Stores data in RAM. - Extremely fast. - Data is lost on restart. - Limited size.	Temporary data storage, caching, or session management.
CSV	- Stores data in plain CSV files. - Easy integration with external tools. - No indexing support.	Data exchange between applications or for simple data storage without complex querying needs.
Archive	- Optimized for high compression. - Supports only INSERT and SELECT operations. - No indexes.	Historical or log data storage where data retrieval is infrequent.
NDB (Clustered)	- Distributed storage for MySQL Cluster. - High availability and fault tolerance. - Supports transactions.	Large-scale distributed applications requiring real-time performance.

30. How would you set a default storage engine in MySQL?

First, you can check the current default storage engine:

SHOW ENGINES;

InnoDB is recommended as the default engine since it supports important features like:

- ACID-compliant transactions
- Foreign key constraints
- Crash recovery
- Row-level locking

To temporarily change the default engine for my current session, you could use:

```
SET default_storage_engine = 'InnoDB';
```

Powered By

For a permanent change, you can modify the MySQL configuration file by adding this line under the [mysqld] section:

```
default-storage-engine = InnoDB
```

Powered By

31. How would you repair corrupted tables in MySQL?

First, you can check all the databases with this command:

```
mysqlcheck --check --all-databases -u root -p
```

Powered By

It will scan all the tables and report if there's any corruption. Then, you may run the following query to repair the table:

```
mysqlcheck --repair database_name table_name -u root -p
```

Powered By

Repairs can lead to data loss in cases of severe corruption, so make sure to back up the data.

Scenario-Based and Problem-Solving MySQL Interview Questions

These questions gauge your experience with real-world complex scenarios and problem-solving abilities.

32. Explain a scenario where you used subqueries in MySQL.

Here’s how you can respond to a question like this:

I managed an e-commerce store database at my recent job, where I had to prepare a product report. The goal was to find products that generated above-average sales, which required using subqueries to perform this multi-step analysis.

Here's the SQL query I developed to solve this:

```
SELECT
   p.product_id,
   p.product_name,
   s.sales_amount
FROM products p
JOIN sales s ON p.product_id = s.product_id
WHERE s.sales_amount > (
   SELECT AVG(sales_amount)
```

```
    FROM sales
)
ORDER BY s.sales_amount DESC;
```

Powered By

First, I established our baseline by calculating the average sales amount across all products. To do so, I used a subquery in the WHERE clause that calculated AVG(sales_amount) from the sales table. This subquery acted as a dynamic threshold against which each product's performance could be measured.

The main query then joined the products and sales tables to pull relevant product details, while the WHERE clause filtered out any products with sales below our calculated average.

By structuring the query this way, I could identify high-performing products in a single database operation rather than running multiple separate queries.

33. Can you explain a situation where you used SQL joins to combine data from multiple tables?

Here's an example answer to the above question:

Recently, I was working on a project where we had two main tables — one with product sales data and another with product details. My task was to create a report showing sales, product name, category, and price.

To combine the relevant data, I used an INNER JOIN on the common column, product_id, to link the sales transactions with product details:

```
SELECT
    s.sales_date,
    p.product_name,
    p.category,
    s.quantity_sold,
    p.price
FROM
    sales s
INNER JOIN
    products p
ON
    s.product_id = p.product_id;
```

Powered By

The report provided a clear picture of sales trends, helping stakeholders identify which product categories were performing well and which needed attention.

34. Do you have any experience with triggers? Explain how you've used them.

Here's an example response to the above question:

Yes, I have extensive experience with database triggers. In my recent role, I implemented an AFTER UPDATE trigger for price change auditing.

Let me walk you through the specific implementation: I created a trigger that automatically captures price history whenever a product price changes. Here's the SQL script I developed:

```
CREATE TRIGGER tr_AuditPriceChanges
ON Products
AFTER UPDATE AS
BEGIN
  -- Only execute if the price was actually changed
  IF UPDATE(UnitPrice)
  BEGIN
    INSERT INTO PriceAudit (
      ProductID,
      OldPrice,
      NewPrice,
      ChangedBy,
      ChangeDate,
      PercentageChange
    )
    SELECT
      i.ProductID,
      d.UnitPrice AS OldPrice,
      i.UnitPrice AS NewPrice,
      SYSTEM_USER,
      GETDATE(),
      ROUND(((i.UnitPrice - d.UnitPrice) / d.UnitPrice * 100), 2)
    FROM INSERTED i
    JOIN DELETED d ON i.ProductID = d.ProductID
    WHERE i.UnitPrice <> d.UnitPrice;
  END
END;
```

What made this solution particularly effective was:

1. It only fires when price changes actually occur.
2. It captures the user making the change using SYSTEM_USER.
3. It calculates the percentage change for reporting purposes.

4. It includes a WHERE clause to filter out non-changes that might occur from other column updates.

I also added error handling and logging when we noticed some edge cases with NULL prices.

Tips for Preparing for a MySQL Interview

If you're just starting your career, here are a few tips that will help you ace your upcoming interview:

Master MySQL core concepts: Learn the **database fundamentals** like indexing, transactions, and query optimizer. Understand how MySQL processes queries and manages data storage. This will help you write efficient queries and explain your solutions during the interview.

Get hands-on experience: **Install MySQL on your computer** and practice regularly. Create test databases, write different types of queries, and try to optimize them. Real practice is the best way to learn how things work and build confidence for the interview.

To further brush up your knowledge, check out DataCamp's resources:

- For SQL introduction: **Introduction to SQL Course**
- For practicing SQL: **Applying SQL to real-world problems**

Learn about MySQL tools and integrations: Get familiar with MySQL Workbench or other tools for database management and basic monitoring. You can also explore **how MySQL works with Python** and relevant frameworks, showing you can work in a real development environment.