



# Hands-on Lab: Monitoring and Optimizing Your Databases in MySQL

**Estimated time needed:** 45 minutes

In this lab, you'll learn how to monitor and optimize your database in MySQL with the help of phpMyAdmin.

## Objectives

After completing this lab, you will be able to:

1. Maintain the health and performance of your database with phpMyAdmin
2. Identify the difference between an unoptimized and optimized database
3. Optimize your database with phpMyAdmin by applying best practices

## Software Used in this Lab

In this lab, you will use [MySQL](#). MySQL is a Relational Database Management System (RDBMS) designed to efficiently store, manipulate, and retrieve data.



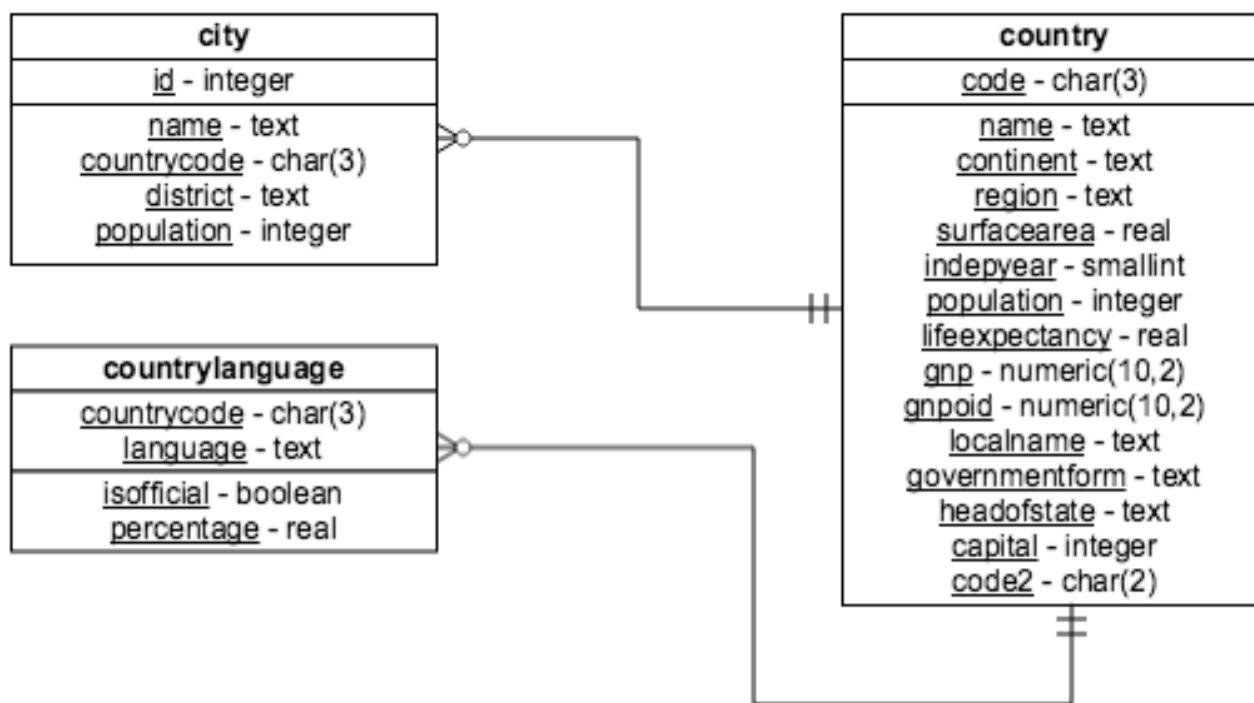
To complete this lab, you will utilize the MySQL relational database service available as part of the IBM Skills Network Labs (SN Labs) Cloud IDE. SN Labs is a virtual lab environment used in this course.

## Database Used in this Lab

The World database used in this lab comes from the following source: <https://dev.mysql.com/doc/world-setup/en/> under [CC BY 4.0 License](#) with [Copyright 2021 - Statistics Finland](#).

You will use a modified version of the database for the lab, so to follow the lab instructions successfully please use the database provided with the lab, rather than the database from the original source.

The following ERD diagram shows the schema of the World database:



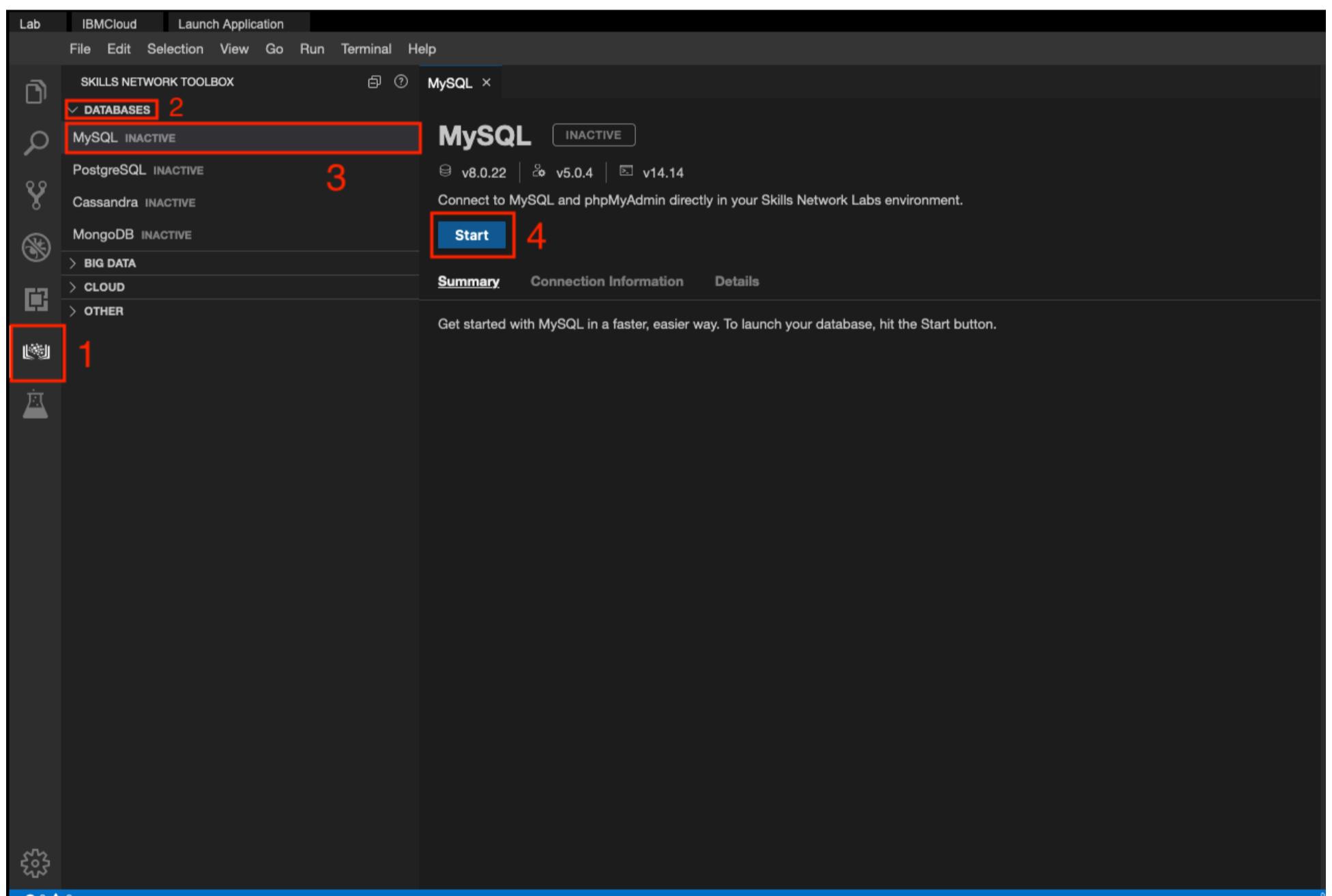
The first row is the table name, the second is the primary key, and the remaining items are any additional attributes.

## Exercise 1: Create Your Database

To begin, we'll create the **World** database and load the necessary data.

1. First, we'll launch MySQL. We can do that by navigating to the **Skills Network Toolbox**, selecting **Databases** and then **MySQL**.

Press **Start**. This will start a session of MySQL in SN Labs.



The **Inactive** label will change to **Starting**. This may take a few moments.

When it changes to **Active**, it means your session has started.

The screenshot shows the Skills Network Toolbox interface with the MySQL tab active. The MySQL section is labeled "ACTIVE" and shows version information: v8.0.22, v5.0.4, and v14.14. There is a "Stop" button. Below it, there are tabs for "Summary", "Connection Information", and "Details". A note says: "Your database and phpMyAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate MySQL, please check out the Details section." It includes fields for "Username" and "Password" with copy icons, and a link to "phpMyAdmin". Another section says: "You can manage MySQL via:" followed by "MySQL CLI" and "New Terminal" buttons.

2. In the menu bar, select **Terminal** > **New Terminal**. This will open the Terminal.

To download the zip file containing the **.sql** file for the **World** database, copy and paste the following into the Terminal:

```
wget https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DB0231EN-SkillsNetwork/datasets/World/world_mysql_script.sql
```

3. Next, initiate a MySQL command-line session by clicking **MySQL CLI** from the MySQL tab. Doing this will allow us to create a table in MySQL via the command-line.

MySQL X

# MySQL

ACTIVE

v8.0.22 | v5.0.4 | v14.14

Connect to MySQL and phpMyAdmin directly in your Skills Network Labs environment.

**Stop**

**Summary** **Connection Information** **Details**

Your database and phpMyAdmin server are now ready to use and available with the following login credentials. For more details on how to navigate MySQL, please check out the Details section.

**Username:** [REDACTED] 

**Password:** [REDACTED] 

You can manage MySQL via:

**phpMyAdmin** 

Or to interact with the database in the terminal, select one of these options:

**MySQL CLI** **New Terminal**

4. In the Terminal, create a new database called **world**.

- ▶ Hint (Click Here)
- ▶ Solution (Click Here)

5. Now, we'll want to use the newly created database, **world**. How can we go about doing that?

- ▶ Hint (Click Here)
- ▶ Solution (Click Here)

6. In order to complete the database creation process, we'll want to execute the MySQL script containing the data that we downloaded. This file will create tables and insert data into those tables.

- ▶ Hint (Click Here)
- ▶ Solution (Click Here)

This may take about thirty seconds or so.

7. With our database created, we'll want to take a look at the tables inside it to get a general idea of the data we have. What tables do we have?

- ▶ Hint (Click Here)
- ▶ Solution (Click Here)

## Exercise 2: Monitor Your Database

Database monitoring refers to the tasks related to reviewing the operational status of your database. Monitoring is critical in maintaining the health and performance of your database because it'll help you identify issues in a timely manner. In doing so, you'll be able to avoid problems such as database outages that affect your users.

With phpMyAdmin, we can take a look at how our database is doing and how we can improve it.

1. In the MySQL tab, select button next to **phpMyAdmin** to access the tool in a new tab.

The screenshot shows the MySQL tab interface. At the top, it displays the MySQL version (v8.0.22), PHP version (v5.0.4), and MySQL client version (v14.14). Below this, a message says "Connect to MySQL and phpMyAdmin directly in your Skills Network Labs environment." A large blue "Stop" button is visible. Below the button, there are three tabs: "Summary" (selected), "Connection Information", and "Details". The "Summary" tab contains a message about ready-to-use login credentials and a "Details" section. Under "Details", there are fields for "Username" and "Password", both of which are redacted. Below these fields, a message says "You can manage MySQL via:" followed by a "phpMyAdmin" button, which is highlighted with a red box. Further down, a message says "Or to interact with the database in the terminal, select one of these options:" followed by two buttons: "MySQL CLI" and "New Terminal".

2. On the phpMyAdmin homepage, you'll notice a left sidebar and right panel.

On the left sidebar, you'll see all your databases, including the one we just created, **world**.

For now, we'll want to focus on the right panel to see more information about our servers and databases.

Select **Status**.

The screenshot shows the phpMyAdmin Status page. The top navigation bar includes "Server: mysql:3306", "Databases", "SQL", "Status" (which is selected and highlighted with a red box), "User accounts", "Export", "Import", "Settings", "Binary log", "Replication", "Variables", "Charsets", "Engines", and "Plugins". The left sidebar lists databases: "information\_schema", "mysql", "performance\_schema", "sys", and "world". The main content area is divided into several sections: "General settings" (Server connection collation: utf8mb4\_unicode\_ci, More settings), "Appearance settings" (Language: English, Theme: pmahomme), "Database server" (Server: mysql via TCP/IP, Server type: MySQL, Server connection: SSL is not being used, Server version: 8.0.22 - MySQL Community Server - GPL, Protocol version: 10, User: root@172.25.0.3, Server charset: UTF-8 Unicode (utf8mb4)), "Web server" (Apache/2.4.38 (Debian), Database client version: libmysql - mysqld 7.4.15, PHP extension: mysqli, curl, mbstring, PHP version: 7.4.15), and "phpMyAdmin" (Version information: 5.0.4, latest stable version: 5.1.1, Documentation, Official Homepage, Contribute, Get support, List of changes, License).

3. There are several subpages on the **Status** page: **Server**, **Processes**, **Query Statistics**, **All Status Variables**, **Monitor** and **Advisor**. These subpages will give you an inside look at the current processes on your server.

Network traffic since startup: 10.3 MiB

This MySQL server has been running for 0 days, 0 hours, 17 minutes and 47 seconds. It started up on Oct 07, 2021 at 08:30 PM.

Traffic	#	ø per hour	Connections	#	ø per hour	%
<b>Received</b>	1.7 MiB	5.7 MiB	<b>Max. concurrent connections</b>	4	---	---
<b>Sent</b>	8.7 MiB	29.2 MiB	<b>Failed attempts</b>	203	684.91	15.32%
<b>Total</b>	10.3 MiB	34.9 MiB	<b>Aborted</b>	0	0	0%
			<b>Total</b>	1,325	4,470.48	100.00%

This MySQL server works as **master** in replication process.

#### Replication status

##### Master status

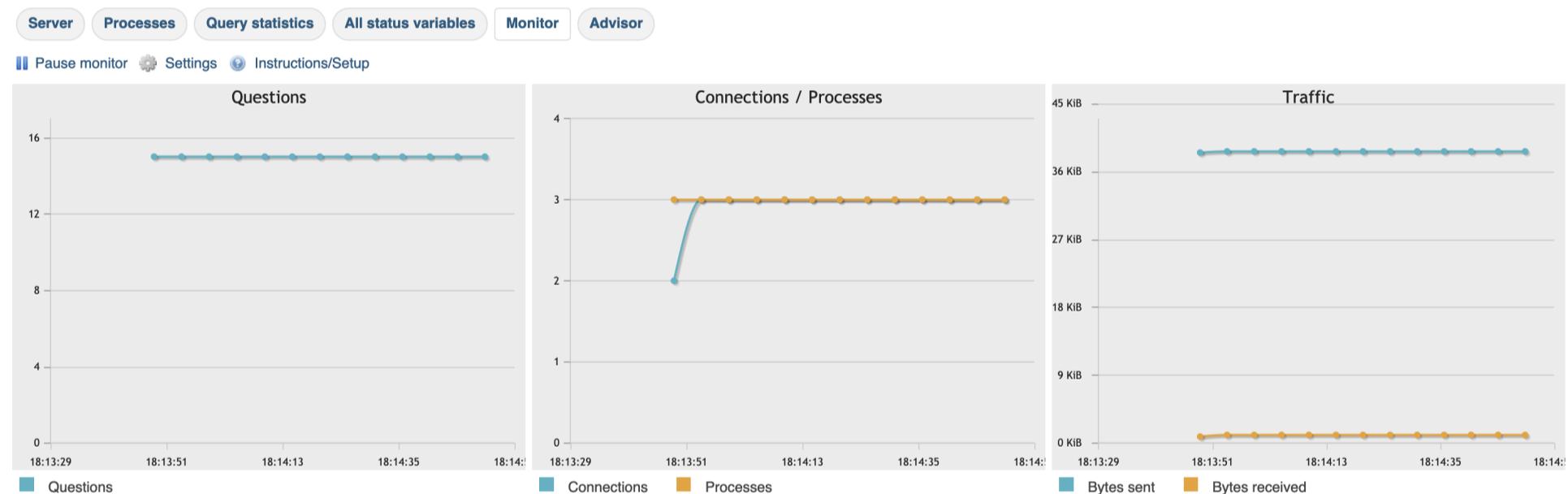
Variable	Value
File	binlog.000002
Position	1690985
Binlog_Do_DB	
Binlog_Ignore_DB	

Feel free to explore the rest of the subpages to gain a better understanding of the current status of your server.

If you'd like a refresher on the information provided by each subpage, you can revisit the previous reading, [Monitoring and Optimizing Your Databases in MySQL](#).

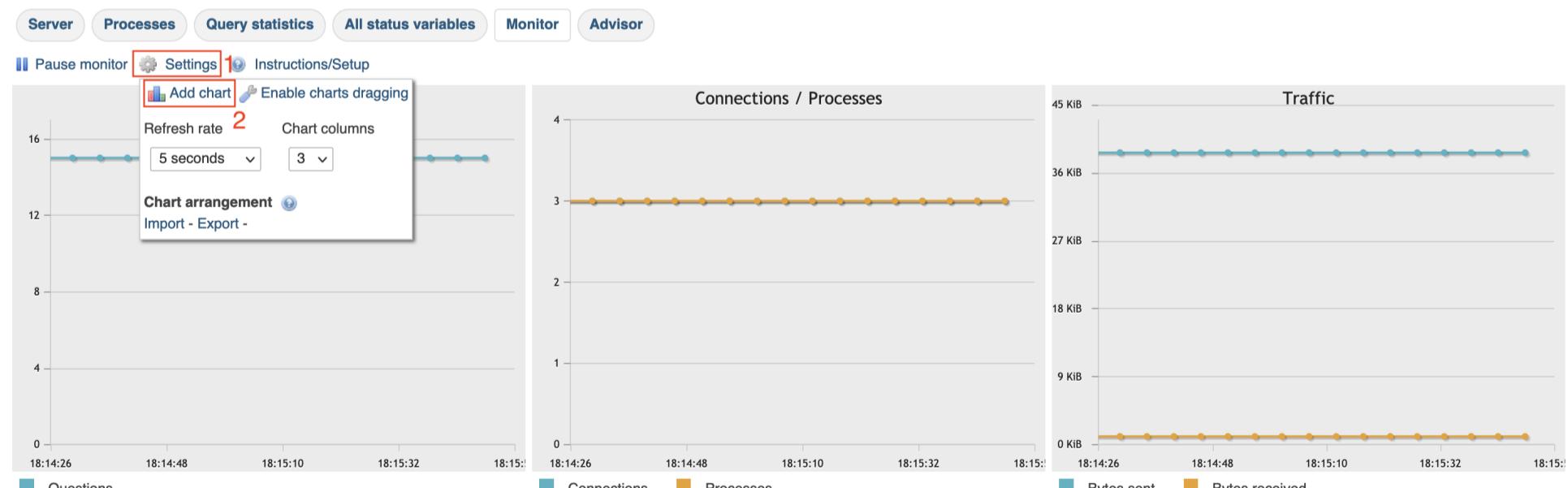
In this lab, we'll be taking a look at the **Monitor** subpage.

4. From the available sections, click **Monitor**.



When you first arrive at this page, you may only have three charts: **Questions**, **Connections / Processes** and **Traffic**. We can add a few more graphs to help track important metrics.

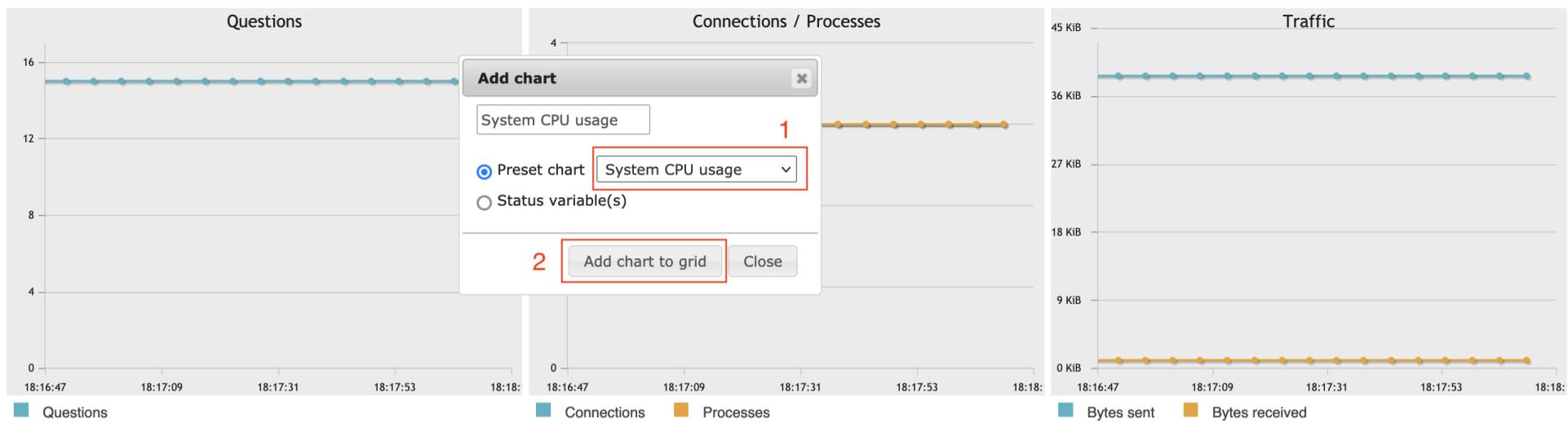
5. Select **Settings** then **Add chart**.



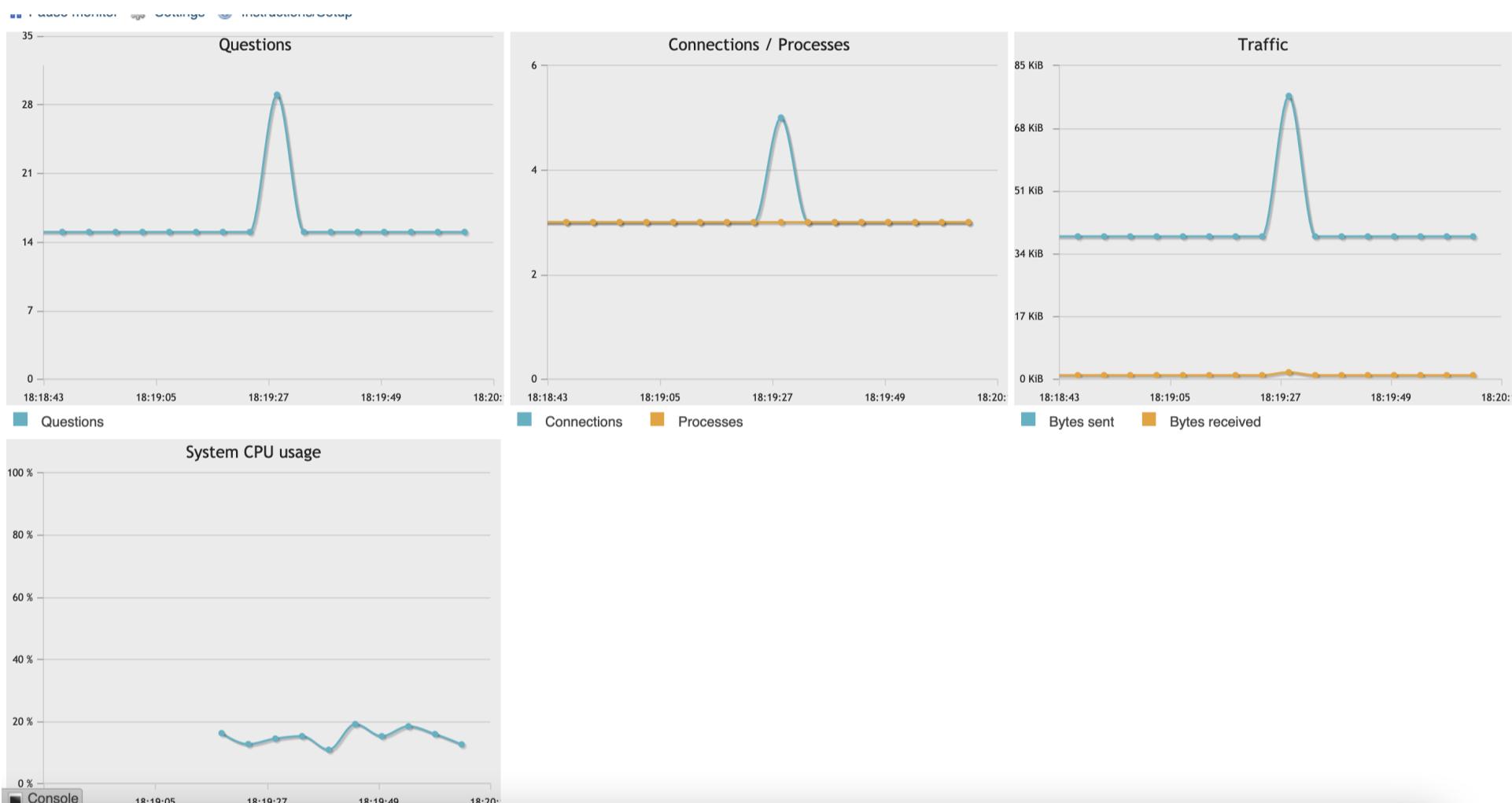
6. In the dialog box, you can choose a chart based on a status variable or select **Preset chart**. In this case, we'll be adding two preset charts.

First, let's select **System CPU Usage**.

Press **Add chart to grid**.



Now, you should see the **System CPU Usage** following chart.



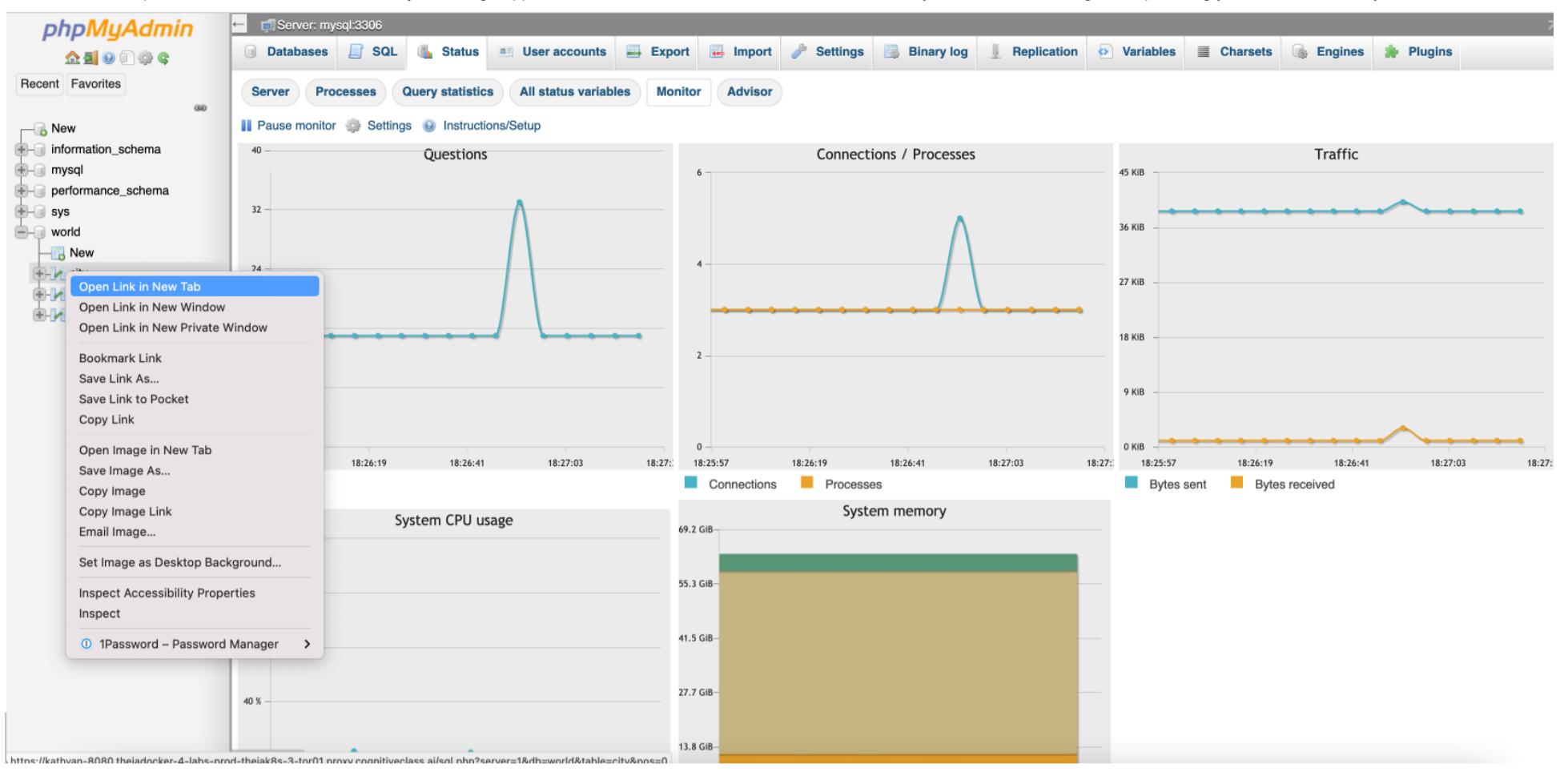
Notice how there was a spike in your charts. Recall that **Questions** includes any queries run in the background by phpMyAdmin. In this case, the interaction was MySQL bringing up the **System CPU Usage** chart.

7. Repeat this process to add the **System memory** chart.

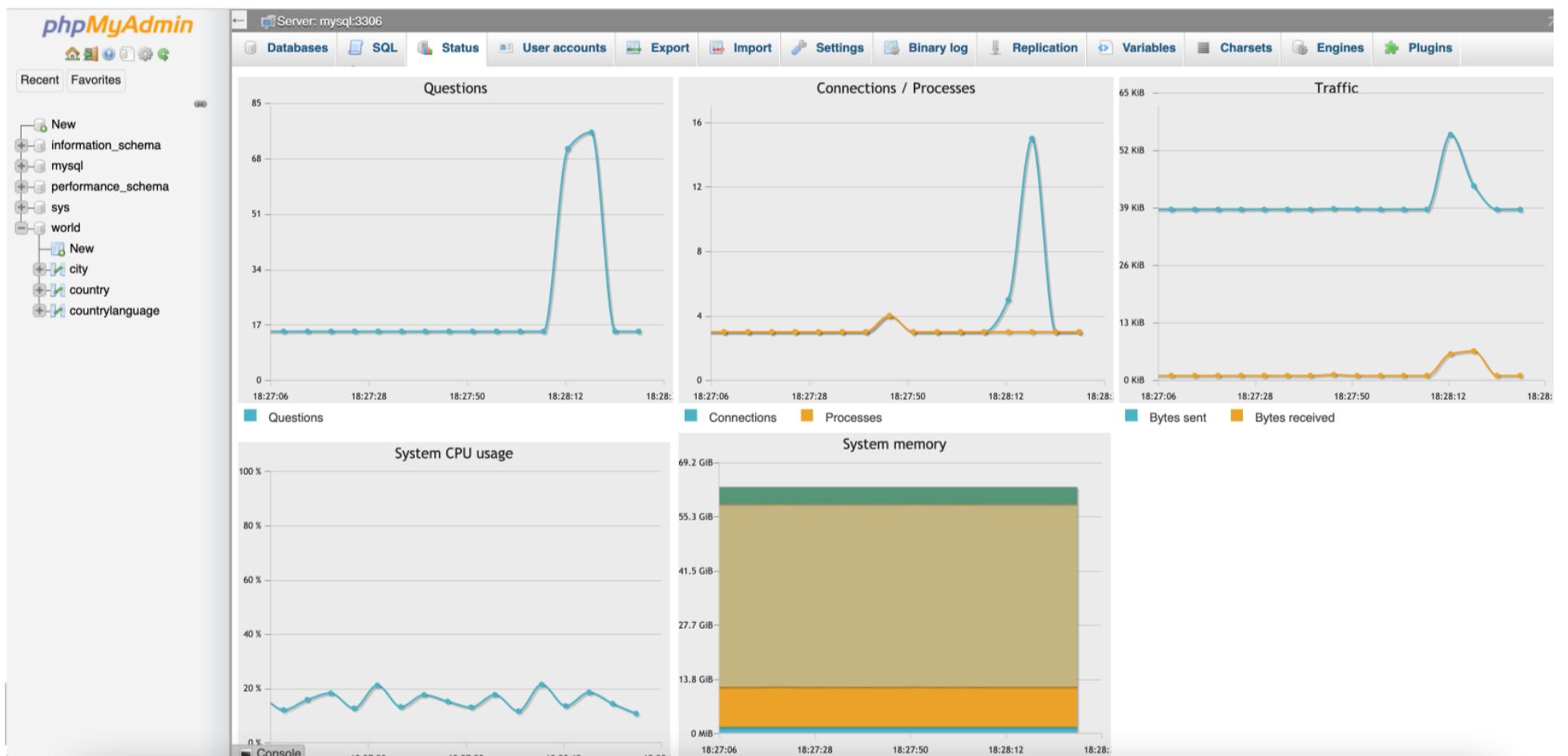
- ▶ Hint (Click Here)
- ▶ Solution (Click Here)

What happens when we run a query? Let's take a look!

8. On the left sidebar, press the + sign next to the database **world**. Then, right-click **city** and select **Open link in new tab**.



Doing this will run a `SELECT * FROM city` query. Notice the spike in the charts. Hovering over the number will provide you more insight into the number of questions, or queries, that are run.



To confirm that we ran the query, we can navigate to the new tab we opened and take a look at the success message:

The screenshot shows the phpMyAdmin interface for the 'world' database. The 'city' table is selected. The table has columns: ID, Name, CountryCode, District, and Population. The data includes cities like Kabul, Qandahar, Herat, Mazar-e-Sharif from AFG, and Amsterdam, Rotterdam, Haag, Utrecht, Eindhoven, Tilburg, Groningen, Breda, Apeldoorn, Nijmegen, Enschede from NLD.

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabol	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843
10	Tilburg	NLD	Noord-Brabant	193238
11	Groningen	NLD	Groningen	172701
12	Breda	NLD	Noord-Brabant	160398
13	Apeldoorn	NLD	Gelderland	153491
14	Nijmegen	NLD	Gelderland	152463
15	Enschede	NLD	Overijssel	149544

Since phpMyAdmin limits queries to displaying the first 25 rows (similar to the `LIMIT` clause), the query did not take long to load.

The **Status** page is helpful in monitoring the health and performance of your server and database. Particularly with the **Monitor** page's charts, you'll be able to see real-time information on how your databases are doing. If there are unexpected irregularities, such as a sudden spike in queries or a high volume of incoming traffic, then that may point to a problem that needs to be attended to.

## Exercise 3: Optimize Your Database

Database optimization refers to maximizing the speed and efficiency of retrieving data from your database. When you optimize your database, you are improving its performance, which can also improve the performance of slow queries. By optimizing your database, you'll improve the experience for both yourself and your users.

We can see this in practice with the following scenario:

Imagine that you are the organizer of a sports tournament with a total of 120 participating teams. There's three pieces of information that you've collected from each team: their (unique) team number, their three-letter team code for the system, and the date of the team's creation.

How can we go about optimizing a database containing that information? Let's take a look!

### Task A: Create Your Table

- To start, we'll create a database in phpMyAdmin.

On the left sidebar, select the **New** button to create a new database.

2. On the **Databases** page, you can enter a Database name and leave it as the **UTF-8** encoding. UTF-8 is the most commonly used character encoding for content or data.

Let's call our database **tournament\_teams**.

Press **Create**.

Database	Collation	Master replication	Action
information_schema	utf8_general_ci	Replicated	<a href="#">Check privileges</a>
mysql	utf8mb4_0900_ai_ci	Replicated	<a href="#">Check privileges</a>
performance_schema	utf8mb4_0900_ai_ci	Replicated	<a href="#">Check privileges</a>
sys	utf8mb4_0900_ai_ci	Replicated	<a href="#">Check privileges</a>
world	utf8mb4_0900_ai_ci	Replicated	<a href="#">Check privileges</a>

Total: 5

Check all   With selected: Drop

3. Now, we can make a table. Let's call it **teams**.

Recall the information we want to store in this database: team number, team code and team creation date. We can make three columns for that.

Press **Go**.

4. We can now add in the column names and data types.

We'll enter the following information into the table. Below are only the values that you'll need to fill out. Anything else can be left as is:

Name	Type	Length/Values
team_no	INT	
team_code	CHAR	100
creation_date	CHAR	100

Since the team number is unique, with each of the 120 teams having their own number ranging from 1 to 120, that will be our primary key. We will set that later. It is a numeric data type, so we use **INT** for now.

The team code and creation date are both set as the **CHAR** data type. Notice that for the **CHAR** type, we must include the length of the value. To be safe, we've set that to 100.

You might be thinking that these data types aren't the best choices for this database. You'd be correct! Given what we know, there are better choices for each of the entries, but let's first take a look at an unoptimized database and how we can optimize it.

When all the values are entered, press **Save**.

Name	Type	Length/Values	Default	Collation	Attributes	Null	Index	A_I	Comments	Virtuality
team_no	INT		None			<input type="checkbox"/>	---	<input type="checkbox"/>		
team_code	CHAR	100	None			<input type="checkbox"/>	---	<input type="checkbox"/>		
creation_date	CHAR	100	None			<input type="checkbox"/>	---	<input type="checkbox"/>		

5. On the **Structure** page, select the checkbox next to **team\_no** and click **Primary**.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	team_no	int		No	None				<input type="checkbox"/> Change <input type="checkbox"/> Drop <input type="checkbox"/> More
2	team_code	char(100)	utf8mb4_0900_ai_ci	No	None				<input type="checkbox"/> Change <input type="checkbox"/> Drop <input type="checkbox"/> More
3	creation_date	char(100)	utf8mb4_0900_ai_ci	No	None				<input type="checkbox"/> Change <input type="checkbox"/> Drop <input type="checkbox"/> More

This will set **team\_no** as the primary key of the table. If you were successful in doing so, you'll receive a success message, like the following:

Your SQL query has been executed successfully.

```
ALTER TABLE `teams` ADD PRIMARY KEY( `team_no`);
```

[Edit inline] [Edit] [Create PHP code]

**Table structure** **Relation view**

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	team_no	int			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	team_code	char(100)	utf8mb4_0900_ai_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
3	creation_date	char(100)	utf8mb4_0900_ai_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

Check all With selected: [Browse](#) [Change](#) [Drop](#) [Primary](#) [Unique](#) [Index](#) [Fulltext](#)

[Print](#) [Move columns](#) [Normalize](#)

Add 1 column(s) after creation\_date [Go](#)

**Indexes**

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
<a href="#">Edit</a> <a href="#">Drop</a>	PRIMARY	BTREE	Yes	No	team_no	0	A	No	

Create an index on 1 columns [Go](#)

Notice that **team\_no** was automatically added as a PRIMARY index as well.

6. Let's import the data! This sample teams data has been prepared specifically for this lab and is composed of team numbers, team code names, and the creation date of these teams.

You can download the file by clicking the following link: [tournament\\_teams\\_data.sql](#).

To import it, navigate to **Import**, then select **Browse** and find the file you just downloaded.

File to import:

File may be compressed (gzip, bzip2, zip) or uncompressed.  
A compressed file's name must end in **[format].[compression]**. Example: **.sql.zip**

2 [Browse...](#) tournament\_teams\_data.sql (Max: 2,048KiB)

You may also drag and drop a file on any page.

Character set of the file: [utf-8](#)

1

Partial import:

Allow the interruption of an import in case the script detects it is close to the PHP timeout limit. (This might be a good way to import large files, however it can break transactions.)

Skip this number of queries (for SQL) starting from the first one: [0](#)

Other options:

Enable foreign key checks

Format:

[SQL](#)

Format-specific options:

SQL compatibility mode: [NONE](#)

Do not use AUTO\_INCREMENT for zero values

3 [Go](#)

Press **Go**.

When your import is successful, you'll see the following message:

✓ Import has been successfully finished, 120 queries executed. (tournament\_teams\_data.sql)

7. Now, we can take a look at the imported data by navigating back to **Browse**. This automatically loads a `SELECT * FROM teams` statement, which will load the first 25 rows from the **teams** table.

The screenshot shows the phpMyAdmin interface for the 'teams' table. At the top, a message says "Showing rows 0 - 24 (120 total, Query took 0.0006 seconds.)". Below is a SQL query: "SELECT \* FROM `teams`". The results table has columns: team\_no, team\_code, creation\_date. The data shows 16 rows from 1 to 16, with the last two rows being LAM and HAR.

team_no	team_code	creation_date
1	ACE	2021-01-01
2	FAX	2021-01-03
3	RED	2021-01-04
4	MAZ	2021-01-05
5	AMS	2021-01-06
6	ROT	2022-02-10
7	HAS	2021-01-08
8	BLU	2021-01-09
9	EIN	2021-01-10
10	TIL	2021-01-11
11	GRO	2021-01-12
12	BRE	2021-01-13
13	APE	2021-01-14
14	NIJ	2021-01-15
15	ENS	2021-01-16
16	HAR	2021-01-17
17	LAM	2021-01-18

The limit is implemented by phpMyAdmin, but you can easily show all rows by checking the **Show all** checkbox.

You will receive a warning asking if you'd like to see all rows. Since there's only 120 rows in this example, you can click **OK**. In larger datasets, loading all the rows will likely crash your browser.

The screenshot shows the phpMyAdmin interface for the 'teams' table. At the top, a message says "Showing rows 0 - 119 (120 total, Query took 0.0008 seconds.)". Below is a SQL query: "SELECT \* FROM `teams`". The results table has columns: team\_no, team\_code, creation\_date. The data shows 17 rows from 1 to 17, with the last row being LAM.

team_no	team_code	creation_date
1	ACE	2021-01-01
2	FAX	2021-01-03
3	RED	2021-01-04
4	MAZ	2021-01-05
5	AMS	2021-01-06
6	ROT	2022-02-10
7	HAS	2021-01-08
8	BLU	2021-01-09
9	EIN	2021-01-10
10	TIL	2021-01-11
11	GRO	2021-01-12
12	BRE	2021-01-13
13	APE	2021-01-14
14	NIJ	2021-01-15
15	ENS	2021-01-16
16	HAR	2021-01-17
17	LAM	2021-01-18

We can now confirm that all 120 rows have been loaded.

- On the left sidebar, select the **tournament\_teams** database.

The screenshot shows the phpMyAdmin interface with the 'Structure' tab selected for the 'teams' table. The table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	team_no	int			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	team_code	char(100)	utf8mb4_0900_ai_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
3	creation_date	char(100)	utf8mb4_0900_ai_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

With this view, you can see all the tables in the database, including the **teams** table that we just created. As you can see, the size of the database is currently 64 KiB, that is, 64 kilobytes. While this is a small size, do remember that this would not be the case with larger datasets as this sample table only has 120 entries.

Keep this number in mind as we start optimizing this table.

## Task B: Optimize Your Data Types

- Under **Action** for the **teams** table, select **Structure**. This will bring you back to the **Structure** page.

Next to **team\_no**, click **Change**. This function is helpful when you need to make adjustments to a column, for any reason.

The screenshot shows the 'Structure' page for the 'teams' table. The 'team\_no' column is selected for modification, indicated by a red box around the 'Change' link in the 'Action' column.

- In this editor, let's change the **Type** to one that would be better suited for this column.

- ▶ Hint (Click Here)
- ▶ Solution (Click Here)

- Change the data type for **team\_code**!

- ▶ Hint (Click Here)
- ▶ Solution (Click Here)

- Finally, let's change the **creation\_date**.

- ▶ Hint (Click Here)
- ▶ Solution (Click Here)

- After updating your data types, your **Structure** page should look like the following:

The screenshot shows the 'Structure' page for the 'teams' table after changes have been applied. A message at the top indicates that the table has been altered successfully. The 'creation\_date' column has been changed from 'char(100)' to 'date'. The updated table structure is as follows:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	team_no	tinyint			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
2	team_code	char(3)	utf8mb4_0900_ai_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
3	creation_date	date			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

## 6. We can look at our rows by clicking **Browse**.

As you can see, we still have all 120 entries. Great! So, let's take a look at the size of this table.

## 7. Return to the **tournament\_teams** database in the left sidebar.

Table	Action	Rows	Type	Collation	Size	Overhead
teams	Browse Structure Search Insert Empty Drop	120	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
<b>Sum</b>		<b>120</b>	<b>InnoDB</b>	<b>utf8mb4_0900_ai_ci</b>	<b>16.0 KiB</b>	<b>0 B</b>

Upon first glance, we can see that the size of the database has reduced to 16 KiB, four times smaller than the original unoptimized database!

## 8. We can take optimization a step further by selecting the **teams** table and, in the dropdown, selecting **optimize table**. This is equivalent to performing the **OPTIMIZE TABLE** function in the command-line interface. This command reorganizes the table data and indexes to reduce storage and make accessing the table more efficient. The exact changes made to the table depend on the storage engine in use.

With the InnoDB engine, which is what we are currently using, the table is actually rebuilt, with the index statistics updated to free unused space.

Table	Op	Msg_type	Msg_text
tournament_teams.teams	optimize	note	Table does not support optimize, doing recreate + ...
tournament_teams.teams	optimize	status	OK

That is what the returned table tells us: InnoDB "optimizes" tables by recreating and analyzing them, and that the optimization had been completed.

## Conclusion

Congratulations! You now know how to monitor and optimize your database with the help of the handy tool, phpMyAdmin. Now, you can apply what you have learned to any database that you create or modify in the future.

## Author(s)

Kathy An

## Other Contributor(s)

Rav Ahuja

## Changelog

Date	Version	Changed by	Change Description
2021-10-12	1.0	Kathy An	Created initial version
2021-10-13	1.1	Steve Hord	Copy edits

© IBM Corporation 2021. All rights reserved.