

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.



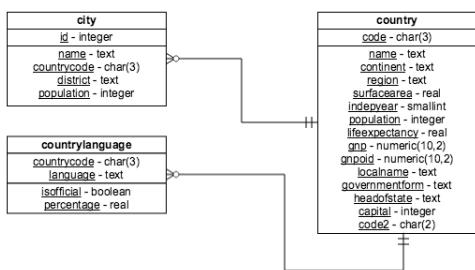
To complete this lab, you will utilize the MySQL relational database service available as part of the IBM Skills Network Labs (SN Labs) Cloud IDE. SN Labs is a virtual lab environment used in this course.

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 screenshot shows the 'SKILLS NETWORK TOOLBOX' interface with the 'DATABASES' section open. The 'MySQL' entry is selected and highlighted with a red circle containing the number '2'. Below it, there's a 'Create' button also highlighted with a red circle containing the number '3'.

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

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

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.

The screenshot shows the 'MySQL' tab in the 'SKILLS NETWORK TOOLBOX' interface. The 'MySQL' entry is active and highlighted with a red circle containing the number '2'. Below it, there's a 'MySQL CLI' button highlighted with a red circle containing the number '1'.

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

▼ Hint (Click Here)

Recall the statement that we used to create a new database in previous labs.

▼ Solution (Click Here)

In the Terminal, we can create a new database **world** with the following:

```
CREATE DATABASE world;
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3039
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database world;
Query OK, 1 row affected (0.01 sec)

mysql>
```

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

▼ Hint (Click Here)

Recall the statement that we used to choose a database to use in previous labs.

▼ Solution (Click Here)

To use the newly created **world** database, we can enter the following into the Terminal:

```
USE world;
```

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)

Recall the command that we used to execute a SQL script file in previous labs.

The name of the SQL script file is **world_mysql_script.sql**, as can be seen in the Cloud IDE file explorer.

▼ Solution (Click Here)

To execute the SQL script file that we downloaded, we can use the following command:

```
SOURCE world_mysql_script.sql;
```

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)

Recall the command that we used to show tables in previous labs.

▼ Solution (Click Here)

We can show the tables in the database with the **SHOW TABLES** command:

```
SHOW TABLES;
+-----+
| tables_in_world |
+-----+
| city           |
| country        |
| countrylanguage|
+-----+
3 rows in set (0.00 sec)

mysql> ■
```

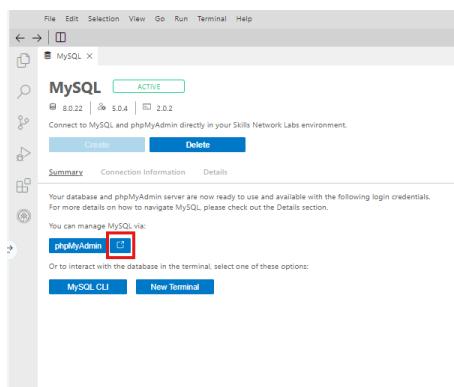
As you can see, there are a total of three tables in the **world** database: **city**, **country** and **countrylanguage**.

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.



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**.

phpMyAdmin

Server: mysql:3306

Databases SQL Status User accounts Export Import Settings

General settings

Server connection collation: utf8mb4_unicode_ci

Appearance settings

Language English Theme: pmahomme

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.

Databases SQL Status User accounts Export Import Settings Binary log Replication

Server Processes Query statistics All status variables Monitor Advisor

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	%
Received	1.7 MiB	5.7 MiB	Max. concurrent connections	4	---	---
Sent	8.7 MiB	29.2 MiB	Failed attempts	203	684.91	15.32%
Total	10.3 MiB	34.9 MiB	Aborted	0	0	0%
			Total	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	

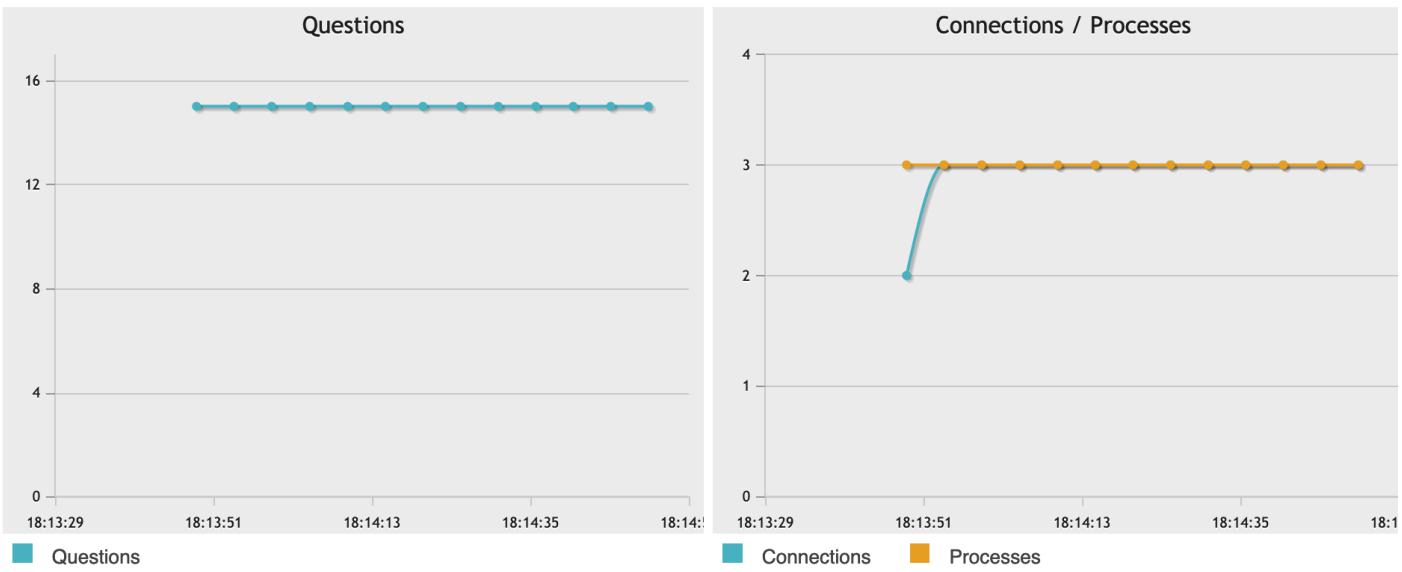
Feel free to explore the rest of the subpages to gain a better understanding of the current status of your server.

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**.

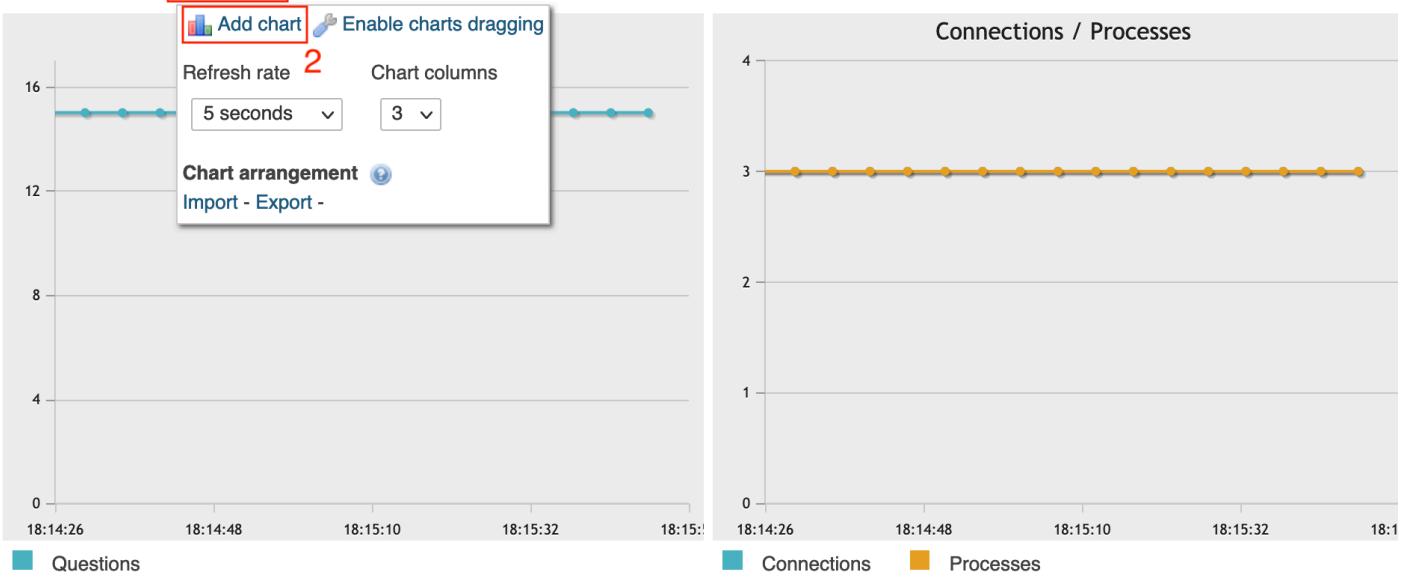
II Pause monitor Settings Instructions/Setup



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**.

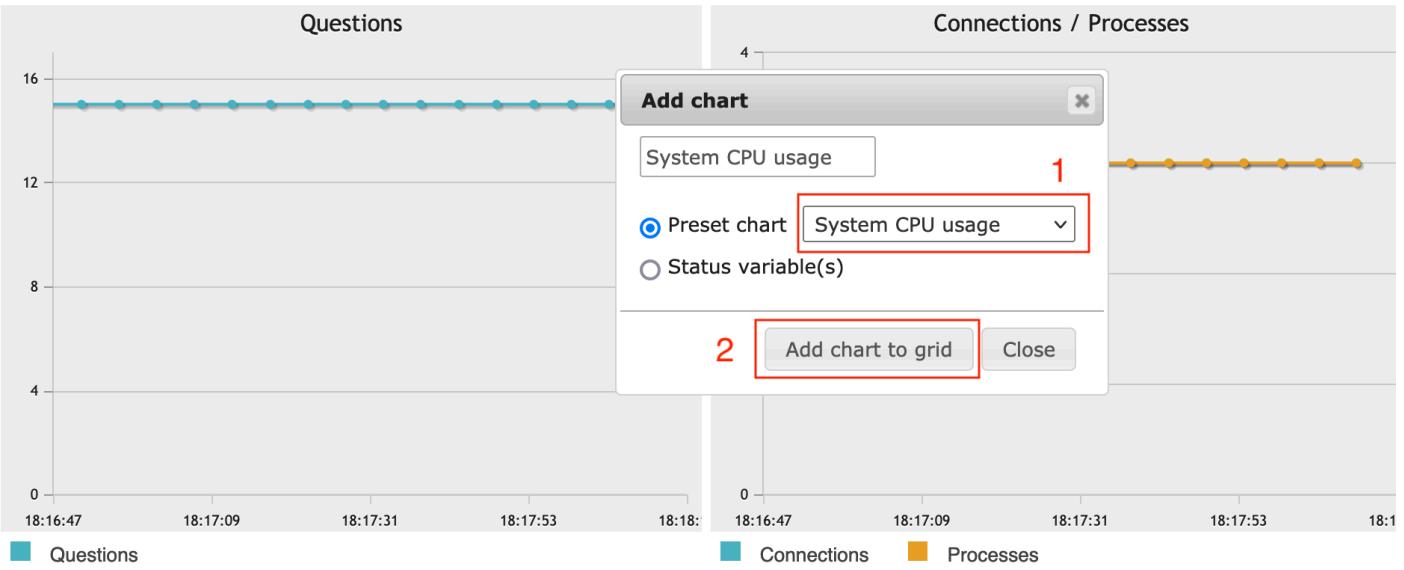
II Pause monitor **Settings** **Instructions/Setup**



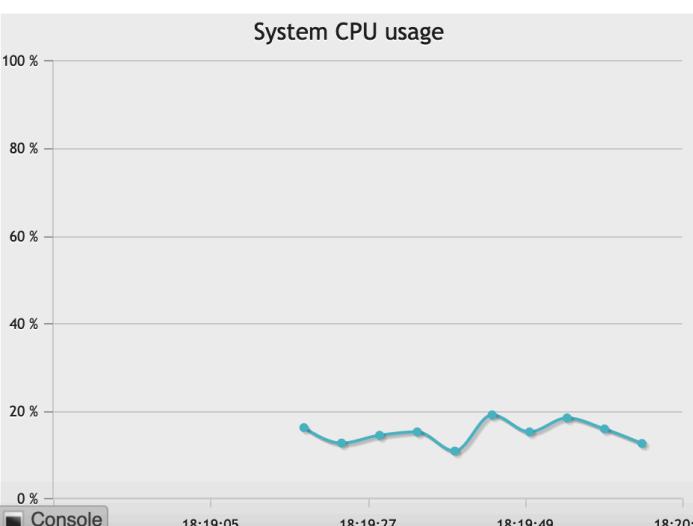
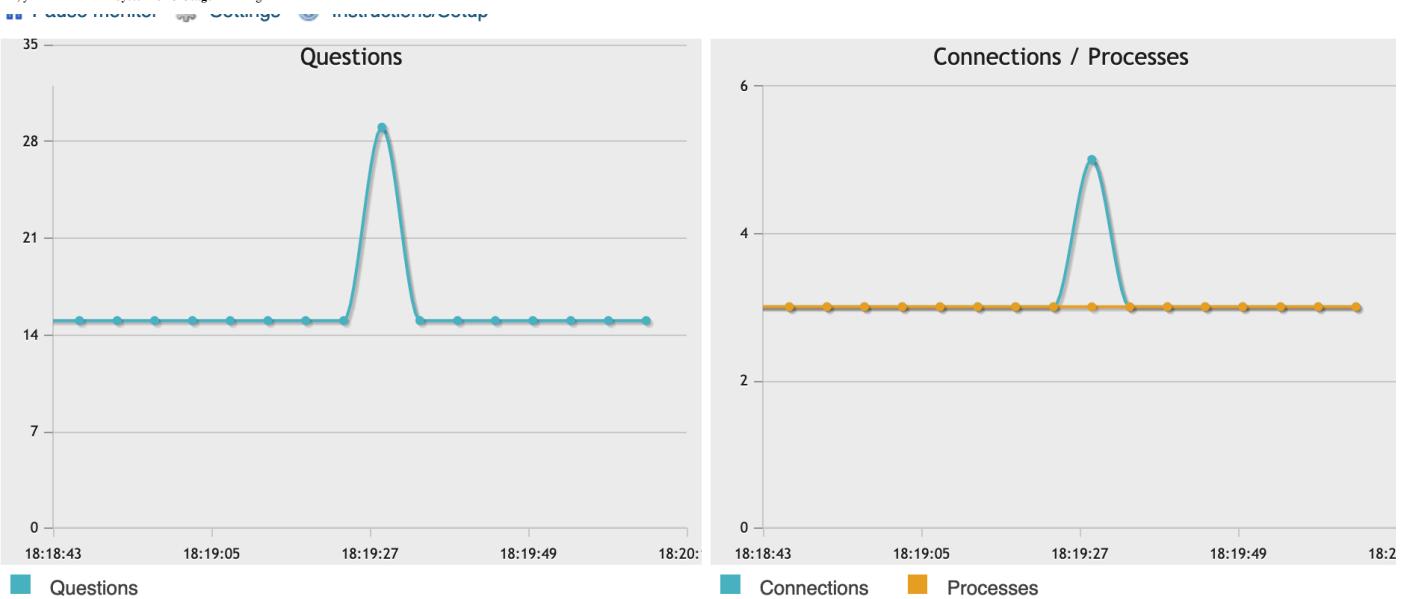
6. In the dialog box, you can choose a chart based on a status variable or select **Preset chart**. In this case, we'll be adding two preset charts.

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.

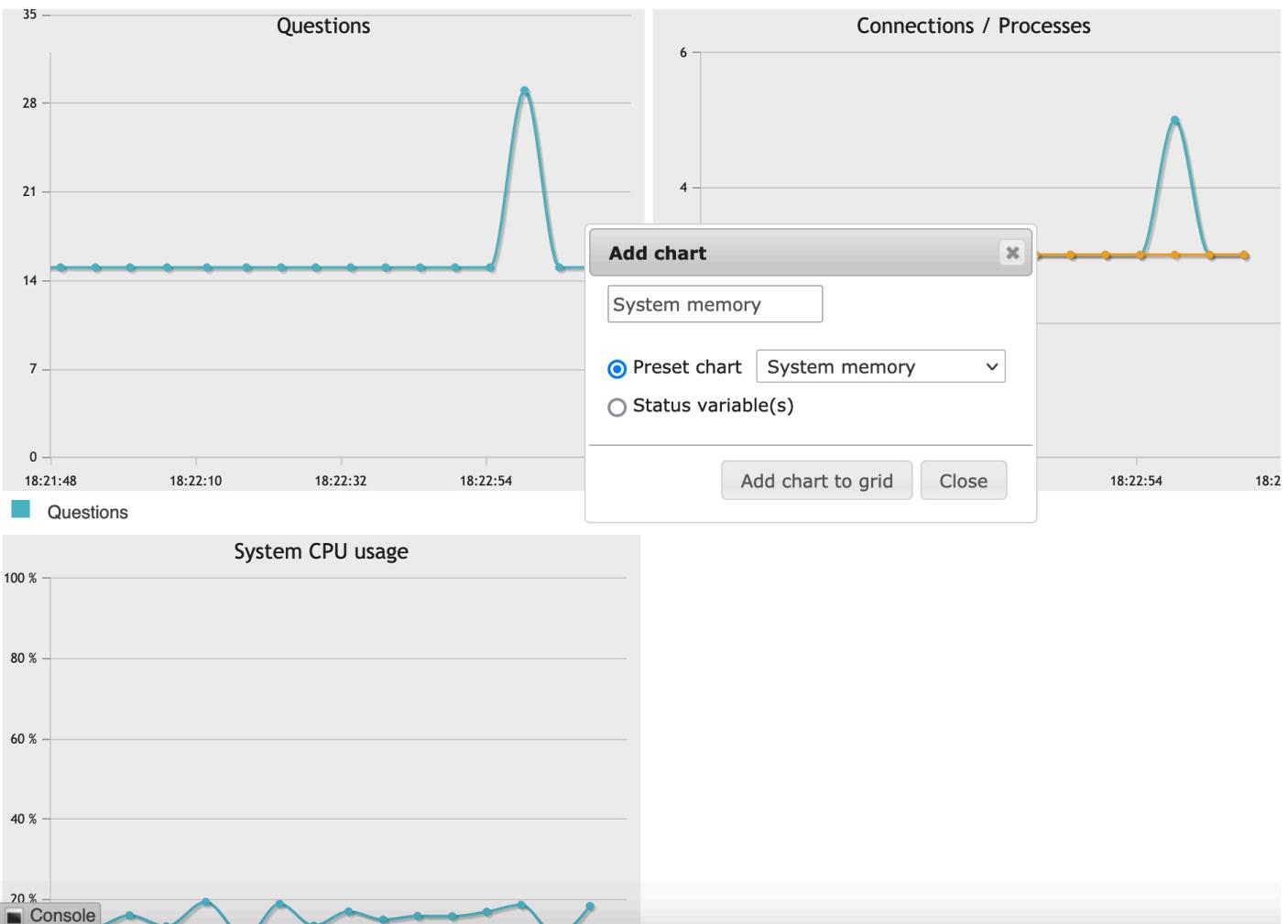
7. Repeat this process to add the **System memory** chart.

▼ Hint (Click Here)

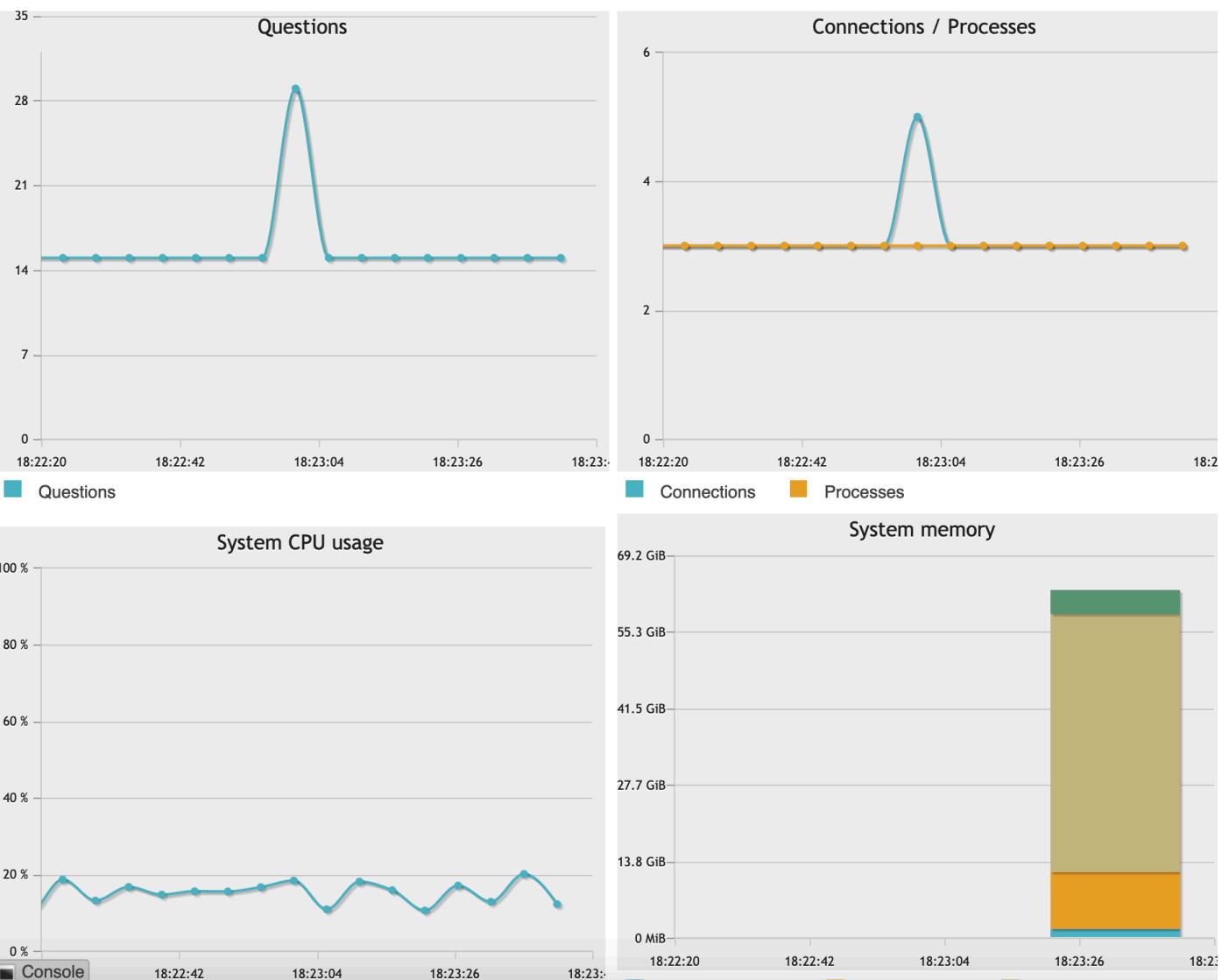
Review how we added the **System CPU Usage** chart. The process is similar, save for one step.

▼ Solution (Click Here)

When adding the chart, the dialog box should look like the following:

[Pause monitor](#) [Settings](#) [Instructions/Setup](#)

Your Monitor page should now look like the following:



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**.

The screenshot shows the phpMyAdmin interface with the database **world** selected. A context menu is open over the **city** table, with the option **Open Link in New Tab** highlighted.

Server: mysql:3306

- Databases:** information_schema, mysql, performance_schema, sys, world
- Tables:** city, New
- Recent:** New, information_schema, mysql, performance_schema, sys, world, New
- Favorites:** None

Server **Processes** **Query statistics** **All status variables** **Monitor** **Advisor**

Instructions/Setup

Questions

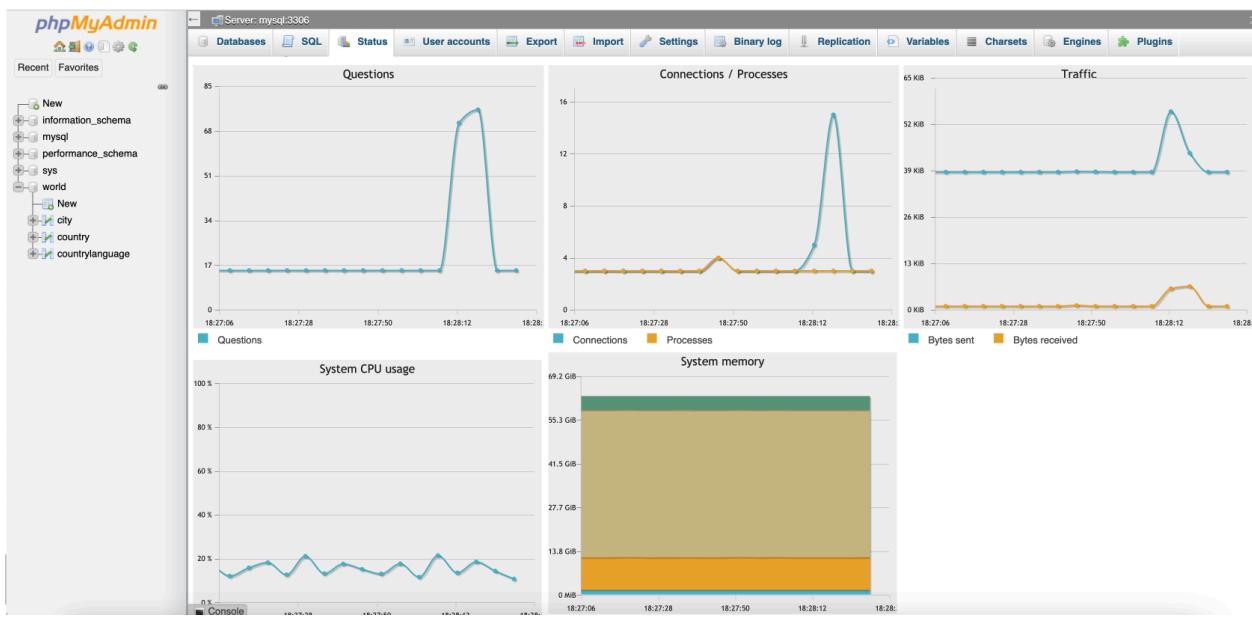
Connections / Processes

Traffic

System CPU usage

System memory

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 queries, 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 'city' table in the 'world' database. The table has the following columns: ID, Name, CountryCode, District, and Population. The data is as follows:

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	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

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

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

The screenshot shows the phpMyAdmin interface with the 'General settings' tab selected. On the left sidebar, there is a 'Recent' section with a 'New' button highlighted by a red box. The main area displays 'Appearance settings' with language set to English and theme to pmahomme. On the right, there are sections for 'Database server' (MySQL via TCP/IP, version 8.0.22), 'Web server' (Apache/2.4.38, PHP 7.4.15), and 'phpMyAdmin' (version 5.0.4).

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**.

The screenshot shows the 'Databases' page. A 'Create database' form is open, with the database name 'tournament_teams' highlighted by a red box (1) and the 'Create' button highlighted by a red box (2). Below the form is a table listing existing databases: information_schema, mysql, performance_schema, sys, and world, all with utf8mb4_0900_ai_ci encoding.

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.

The screenshot shows the 'Structure' tab for the 'tournament_teams' database. A 'Create table' dialog is open, with the table name 'teams' highlighted by a red box (1) and the 'Number of columns:' dropdown highlighted by a red box (2). Below the dialog, a message says 'No tables found in database.'

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**.

Table name: teams Add 1 column(s) Go

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

Structure

Table comments: Collation: Storage Engine: InnoDB

PARTITION definition:

Partition by: (Expression or column)

Partitions:

5. On the Structure page, select the checkbox next to team_no and click Primary.

On the Structure page, select the checkbox next to team_no and click Primary.

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

Check all With selected: Browse Change Drop Primary Unique Index Fulltext

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`);
```

On the Structure page, select the checkbox next to team_no and click Primary.

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

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
Edit Drop	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`

Browse your computer: tournament_teams_data.sql (Max: 2,048KiB)

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

Character set of the file:

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 be dangerous.)*

Skip this number of queries (for SQL) starting from the first one:

Other options:

Enable foreign key checks

Format:

Format-specific options:

SQL compatibility mode:

Do not use AUTO_INCREMENT for zero values

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.

← Server: mysql:3306 > Database: tournament_teams > Table: teams

Browse Structure SQL Search Insert Export Import Privileges Operations Trigger

Showing rows 0 - 24 (120 total, Query took 0.0006 seconds.)

```
SELECT * FROM `teams`
```

1 < > >> | Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

+ Options

	team_no	team_code	creation_date
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	1	ACE	2021-01-01
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	2	FAX	2021-01-03
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	3	RED	2021-01-04
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	4	MAZ	2021-01-05
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	5	AMS	2021-01-06
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	6	ROT	2022-02-10
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	7	HAS	2021-01-08
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	8	BLU	2021-01-09
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	9	EIN	2021-01-10
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	10	TIL	2021-01-11
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	11	GRO	2021-01-12
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	12	BRE	2021-01-13
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	13	APE	2021-01-14
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	14	NIJ	2021-01-15
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	15	ENS	2021-01-16
<input type="checkbox"/> Edit <input type="button" value="Copy"/> <input type="button" value="Delete"/>	16	HAR	2021-01-17

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.

← Server: mysql:3306 » Database: tournament_teams » Table: teams

Browse Structure SQL Search Insert Export Import Privileges Operations Trigger

Showing rows 0 - 119 (120 total, Query took 0.0008 seconds.)

```
SELECT * FROM `teams`
```

Show all | Number of rows: All Filter rows: Search this table Sort by key: None

+ Options

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

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

8. On the left sidebar, select the **tournament_teams** database.

phpMyAdmin

Server: mysql:3306 > Database: tournament_teams

Structure SQL Search Query Export Import Operations Privileges

Filters

Containing the word:

Table	Action	Rows	Type	Collation
teams	Browse Structure Search Insert Empty Drop	120	InnoDB	utf8mb4_09
1 table Sum		120	InnoDB	utf8mb4_09

Check all With selected:

[Print](#) [Data dictionary](#)

Create table

Name: Number of columns: 4

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

1. 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.

← Server: mysql:3306 > Database: tournament_teams > Table: teams

Browse Structure SQL Search Insert Export Import Privileges Operations Trigger

Table structure **Relation view**

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	team_no ↗	int			No	None			Change Drop More
2	team_code	char(100)	utf8mb4_0900_ai_ci		No	None			Change Drop More
3	creation_date	char(100)	utf8mb4_0900_ai_ci		No	None			Change Drop More

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

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

▼ Hint (Click Here)

Consider the available numeric integer data types: **TINYINT**, **SMALLINT**, **MEDIUMINT**, **INT** and **BIGINT**. Which one would be most fitting for this situation?

▼ Solution (Click Here)

If you were thinking **TINYINT**, you would be correct! As we know the maximum signed value of **TINYINT** is 127, this data type would work for our maximum number of 120 teams.

Please note, if you'd like to add an unsigned integer type, you can do that under the **Attributes** column. In this case, we only have 120 teams, so we are within the **TINYINT** maximum signed value range.

Your entry should now look like this:

← Server: mysql:3306 > Database: tournament_teams > Table: teams

Name	Type	Length/Values	Default	Collation	Attributes	Null
team_no	TINYINT		None			<input type="checkbox"/>

Structure

3. Change the data type for **team_code**!

▼ Hint (Click Here)

Consider the string data types: **CHAR** and **VARCHAR**. Given what we know about this data, that all teams have a three-letter team code, which data type and length would be best?

▼ Solution (Click Here)

If you were thinking **CHAR** with a length of **3**, you would be correct! Since we know for sure that every team code is exactly 3 letters, the fixed length **CHAR** data type would be the most suitable in this case.

Your entry should now look like this:

← Server: mysql:3306 > Database: tournament_teams > Table: teams

Name	Type	Length/Values	Default	Collation	Attributes	Null
team_code	CHAR	3	None	utf8mb4_0900_ai_		<input type="checkbox"/>

Structure

4. Finally, let's change the **creation_date**.

▼ Hint (Click Here)

Consider the available data types. Which would be the best choice to contain the values in this column?

▼ Solution (Click Here)

If you were thinking **DATE**, you would be correct! As we know, **DATE** is optimized to contain date values, which is exactly what **creation_date** is.

Your entry should now look like this:

← Server: mysql:3306 > Database: tournament_teams > Table: teams

Name	Type	Length/Values	Default	Collation	Attributes	Null
creation_date	DATE		None	utf8mb4_0900_ai_		<input type="checkbox"/>

Structure

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

← Server: mysql:3306 > Database: tournament_teams > Table: teams

Name	Type	Length/Values	Default	Collation	Attributes	Null
team_no	tinyint		None			<input type="checkbox"/>
team_code	char(3)	utf8mb4_0900_ai_ci	None			<input type="checkbox"/>
creation_date	date		None			<input type="checkbox"/>

Table teams has been altered successfully.

```
ALTER TABLE `teams` CHANGE `creation_date` `creation_date` DATE NOT NULL;
```

Table structure **Relation view**

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	team_no	tinyint		No	None				
2	team_code	char(3)	utf8mb4_0900_ai_ci	No	None				
3	creation_date	date		No	None				

Check all With selected:

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.

← Server: mysql:3306 > Database: tournament_teams

Structure **SQL** **Search** **Query** **Export** **Import** **Operations** **Privileges** **Routines** **Event**

Filters

Containing the word:

Table	Action	Rows	Type	Collation	Size	Overhead
<input type="checkbox"/> teams	★ Browse Structure Search Insert Empty Drop	120	InnoDB	utf8mb4_0900_ai_ci	16.0 KiB	-
1 table	Sum				120 InnoDB utf8mb4_0900_ai_ci 16.0 KiB	0 B

Check all **With selected:**

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.

phpMyAdmin

Server: mysql:3306 > Database: tournament_teams

Structure **SQL** **Search** **Query** **Export** **Import** **Operations** **Privileges**

Filters

Containing the word:

Table	Action	Rows	Type	Collation
<input checked="" type="checkbox"/> teams	★ Browse Structure Search Insert Empty Drop	120	InnoDB	utf8mb4_0900_ai_ci
1 table	Sum			120 InnoDB utf8mb4_0900_ai_ci

Check all

[Print](#) [Data dictionary](#)

Create table

Name:

With selected:

- ✓ Copy table
- Show create
- Export
- Delete data or table**
- Empty
- Drop
- Table maintenance**
- Analyze table
- Check table
- Checksum table
- Optimize table**
- Repair table
- Prefix**
- Add prefix to table
- Replace table prefix
- Copy table with prefix

Rows:

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.

phpMyAdmin

Server: mysql:3306 > Database: tournament_teams

Structure SQL Search Query Export Import Operations Privileges

Your SQL query has been executed successfully.

```
OPTIMIZE TABLE `teams`
```

+ Options

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

Query results operations

[Print](#) [Copy to clipboard](#) [Create view](#)

Filters

Containing the word:

Table	Action	Rows	Type	Collation
<input type="checkbox"/> teams	Browse Structure Search Insert Empty Drop	120	InnoDB	utf8mb4_09
1 table	Sum	120	InnoDB	utf8mb4_09

[Print](#) [Data dictionary](#)

Create table

Name: Number of columns: 4

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

© IBM Corporation 2023. All rights reserved.