

COMP 3015

Week 5

Web Application Development





Goals for today

1. Understand the basics of how to use MySQL with PHP
2. Understand SQL injection vulnerabilities and how parameterized prepared statements work
3. Understand how we can switch our Repository persistence layer from using a file to using a database
4. Know what “CRUD” is
5. Understand the basics of using PDO



DB Basics

Databases allow you to save data in a structured format

- Typical DB operations are create, read, update, delete (CRUD)
- Web applications need to be careful when accepting user input and using that input in queries
- We'll be using a relational database called MySQL



MySQL Basics: Installing

- **Ubuntu** (use your default pkg manager on other Linux distros):
 - **\$ sudo apt update && sudo apt install mysql-server**
 - Start and stop using systemctl
- **macOS** (with homebrew):
<https://formulae.brew.sh/formula/mysql>
 - **\$ brew install mysql**
- **Windows:**
 - <https://dev.mysql.com/downloads/mysql/>



MySQL Basics: Login and Create a Database

On my local machine I don't have a password configured, so I can simply run “mysql -uroot” in order to log into MySQL as the root user. For providing a password you can use the “-p” option.

Creating a database: “create database <new database name>;”

```
[~ $ mysql -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9134
Server version: 8.0.23 Homebrew

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

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 new_co;
Query OK, 1 row affected (0.01 sec)
```



MySQL Basics: Use and show tables

Once a database is created and you're logged into the MySQL shell, you need to instruct MySQL about which database you'd like to use:

```
mysql> use new_co;  
Database changed  
mysql> █
```

The show command will tell you what tables exist in the current database:

```
[mysql> show tables;  
Empty set (0.00 sec)  
  
mysql> █
```

Creating tables and running .sql files

```
create database new_co;  
use new_co;  
create table users (  
    id int auto_increment not null primary key,  
    password_digest varchar(255),  
    email varchar(255) not null,  
    name varchar(255) not null  
);
```

Understanding this is critical for understanding how basic DB backups can be done.

```
[~/Work/BCIT/COMP3015/SQLInjectionExample (master) $ mysql -uroot < exampleSchema.sql  
~/Work/BCIT/COMP3015/SQLInjectionExample (master) $
```



Table Structure

A typical table structure:

```
[mysql> describe posts;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
title	varchar(100)	NO		NULL	
body	varchar(512)	NO		NULL	
created_at	datetime	NO		NULL	
updated_at	datetime	YES		NULL	

```
5 rows in set (0.01 sec)
```




Table Structure Intro (cont.)

```
[mysql> describe posts;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| id         | int           | NO   | PRI | NULL    | auto_increment |
| title      | varchar(100)  | NO   |     | NULL    |                |
| body       | varchar(512)  | NO   |     | NULL    |                |
| created_at | datetime      | NO   |     | NULL    |                |
| updated_at | datetime      | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0.01 sec)
```

- **PRI:** primary key. A unique identifier for each record
 - auto_increment in this case. First record will have ID 1, second will have ID 2, nth will have ID n.
- Columns can be nullable (Null set to YES for updated_at) meaning no value is required
- varchar(n) means a string of maximum length n



Reading/Selecting Records

```
mysql> SELECT * FROM posts WHERE id=1\G
***** 1. row *****
      id: 1
     title: test
    body: test2
created_at: 2022-09-04 07:20:59
updated_at: NULL
1 row in set (0.00 sec)

mysql> █
```

This command selects all fields (*), where the record ID is equal to 1.



Reading/Selecting Records (cont.)

Selecting only the fields we need:

```
mysql> SELECT id, title, body FROM posts WHERE id=1\G
***** 1. row *****
    id: 1
  title: test
   body: test2
1 row in set (0.00 sec)

mysql> █
```

In some instances, a field in a record might contain a lot of data. Excluding it can improve performance. For example, WYSIWYG editor contents being saved with base64 images.



Inserting Records

```
mysql> INSERT INTO posts (id, title, body, created_at, updated_at) VALUES (NULL, "title here", "body here", NOW(), NULL);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|                5 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT * FROM posts WHERE id=5\G
***** 1. row *****
      id: 5
    title: title here
    body: body here
created_at: 2022-10-10 10:20:37
updated_at: NULL
1 row in set (0.01 sec)

mysql> █
```

Note: NULL is passed in for the ID of the auto incrementing primary key (unique ID for each record)

LAST_INSERT_ID() returns the ID of the most recently inserted record on a per-connection basis



Updating Records

Updates should be made with a WHERE clause that will ensure only the intended record is changed.

Tip: run a SELECT query with the WHERE clause before running the UPDATE query to ensure you're only going to update what you intend to.

```
mysql> SELECT * FROM posts WHERE id=5\G
***** 1. row *****
      id: 5
     title: title here
      body: body here
created_at: 2022-10-10 10:20:37
updated_at: NULL
1 row in set (0.00 sec)

mysql> UPDATE posts SET title="Updated title", body="Updated body" WHERE id=5\G
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM posts WHERE id=5\G
***** 1. row *****
      id: 5
     title: Updated title
      body: Updated body
created_at: 2022-10-10 10:20:37
updated_at: NULL
1 row in set (0.00 sec)

mysql> █
```



Deleting Records

```
[mysql> DELETE FROM posts WHERE id=5;  
Query OK, 1 row affected (0.01 sec)
```

```
[mysql> SELECT * FROM posts WHERE id=5\G  
Empty set (0.00 sec)
```

```
mysql> █
```



Foreign Keys

- Imagine a system with users and posts
 - **Post** model and **User** model (classes/objects in our application code)
 - **posts** table and **users** table
- We need to know which users made what posts
 - Need a link between the **posts** table and the **users** table



Foreign Key Relations

author_id on the posts table references the users table.

- MUL: multiple
 - Multiple occurrences of the same value are allowed
 - For example, your user can create 10 posts, so in the posts table your user ID would have 10 occurrences

```
mysql> describe posts;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
title	varchar(255)	NO		NULL	
body	varchar(255)	NO		NULL	
author_id	int	NO	MUL	NULL	
created_at	datetime	NO		NULL	
updated_at	datetime	YES		NULL	

```
6 rows in set (0.00 sec)
```

```
mysql> describe users;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
password_digest	varchar(255)	YES		NULL	
email	varchar(255)	NO	UNI	NULL	
name	varchar(255)	NO		NULL	

```
4 rows in set (0.00 sec)
```

```
mysql> █
```




Foreign Key Relations

```
mysql> show create table posts\G
***** 1. row *****
      Table: posts
Create Table: CREATE TABLE `posts` (
  `id` int NOT NULL AUTO_INCREMENT,
  `title` varchar(255) NOT NULL,
  `body` varchar(255) NOT NULL,
  `author_id` int NOT NULL,
  `created_at` datetime NOT NULL,
  `updated_at` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `author_id` (`author_id`),
  CONSTRAINT `posts_ibfk_1` FOREIGN KEY (`author_id`) REFERENCES `users` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=7 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.00 sec)

mysql> █
```



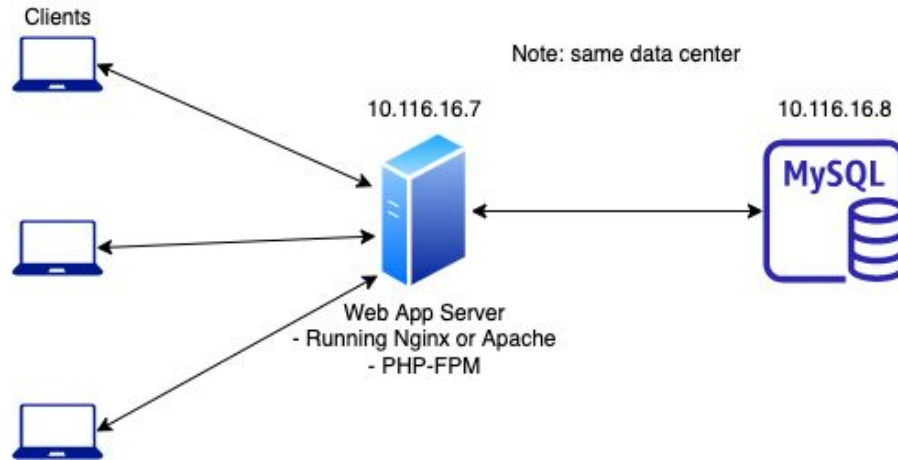
Additional DB topics to know

These are outside the scope of this course but you should learn them eventually. Some of these are covered in COMP 4669.

- Indexing
- Joins
- Relationship types
 - Many-to-many, one-to-many, one-to-one, etc.
- Transactions
 - ACID (atomicity, consistency, isolation, durability)

Database Driven Application Architectures

Web App Architecture with a DB

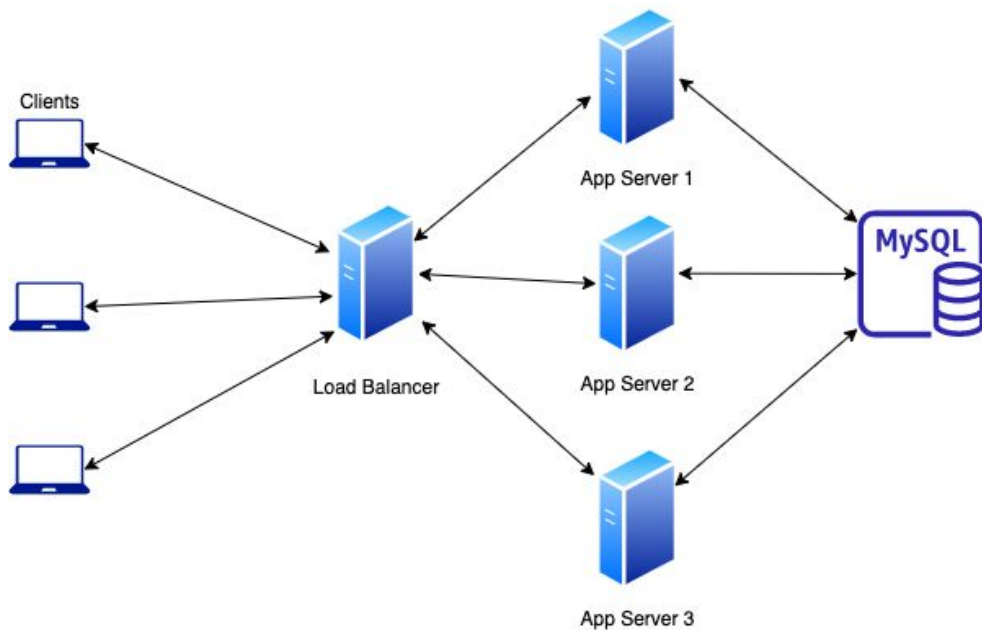


Good idea to separate the web server from the database for scaling:

Security: limit exposure of the DB to the outside world

Scalability: by keeping the DB on the same machine as the web server it becomes easier to “horizontally scale” - see next slide.

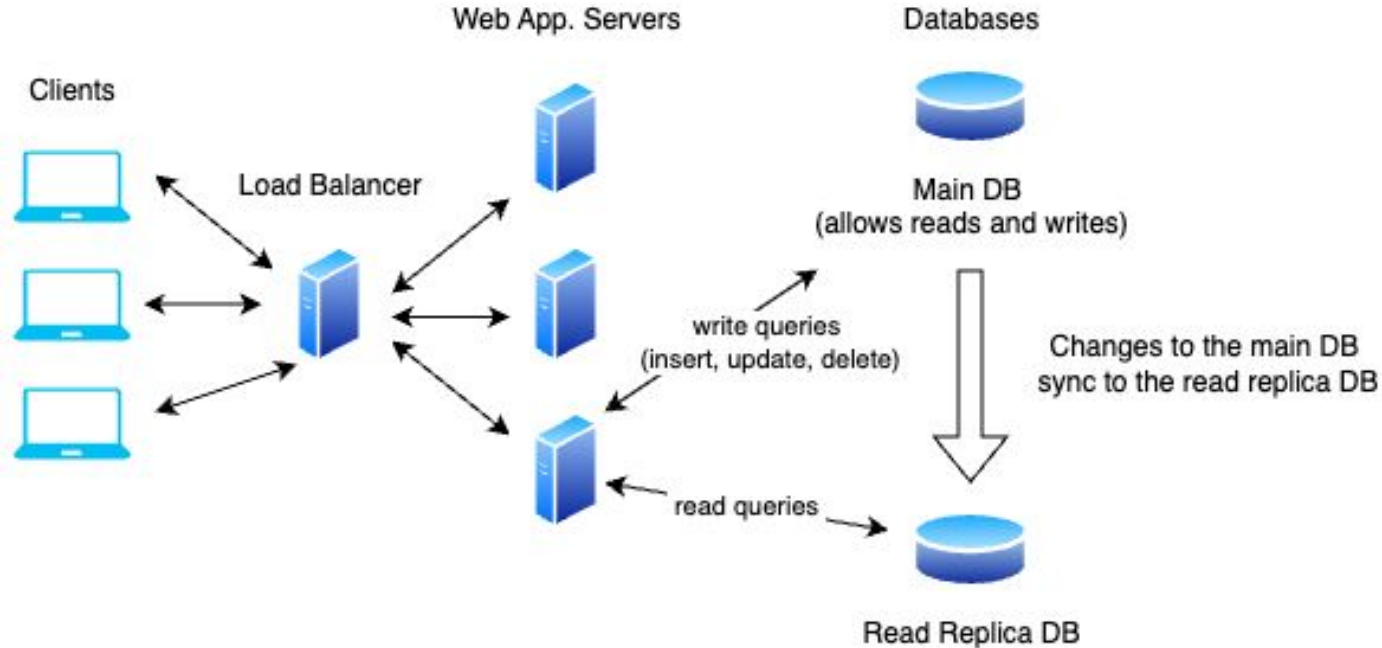
Web App Architecture with a DB



Horizontal scalability: app server 1, 2, 3 all run the the PHP application you're writing

Need to reduce load off of the database?

Idea: create a read-only (replica) database and direct read queries away from the main database which now only needs to handle queries modifying data.



Various cloud service providers such as AWS have out-of-the-box support for this. Eg. using AWS RDS <https://aws.amazon.com/rds/features/read-replicas/>

Database Drivers





Available PHP Database APIs for MySQL

PHP can connect to a MySQL database using your choice of API: mysqli or PDO (PDO_MYSQL, in this course)

- <https://www.php.net/manual/en/book.mysqli.php>
 - MySQL specific
- <https://www.php.net/manual/en/book.pdo.php>
 - Consistent interface between database types: PostgreSQL, MySQL, MS SQL Server, SQLite, etc.
 - Additional supported drivers can be found at:
<https://www.php.net/manual/en/pdo.drivers.php>
- The driver acts as an intermediary between PHP and MySQL



Checking for and Installing DB drivers

- The driver we're going to use is [PDO_MYSQL](#).
- Depending on how PHP is installed and configured, you might already have it available on your system.
- Run “**php -m**” to see the PHP modules that are installed
 - If PDO_MYSQL is installed you should see “pdo_mysql” in the output
- If it's not installed, open your **php.ini** file, search for and uncomment **extension=pdo_mysql**

SQL Injection Attacks and Parameterized Prepared Statements



SQL Injection

General idea with php-ish-pseudocode:

```
$name = $_POST['name'];
```

```
$sql = "SELECT * FROM users WHERE name = '" . $name . "'";
```

With an expected "happy path" input, the resulting SQL query would look like:

```
$name = 'Chris';
```

Resulting in:

```
$sql = "SELECT * FROM users WHERE name = 'Chris'";
```

An unexpected input (from the developer perspective) may be:

```
$name = "` OR 1=1;--";
```

Resulting in:

```
$sql = "SELECT * FROM users WHERE name = "` OR 1=1;--";
```

What happens?

Warning

Security warning: SQL injection

If the query contains any variable input then [parameterized prepared statements](#) should be used instead. Alternatively, the data must be properly formatted and all strings must be escaped using the [mysqli_real_escape_string\(\)](#) function.

A common web service vulnerability caused by mixing user input with database queries.

“SQL Injection is best prevented through the use of parameterized queries.”

src: OWASP,

https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html



Selected PDO Functions

[exec](#) - execute a SQL statement, return the number of rows affected

[prepare](#) - prepare a SQL statement for execution, return a [PDOStatement](#)

[execute](#) - execute a prepared statement against the DB

[fetch](#) - fetch the next row from a result set

[fetchAll](#) - fetch all rows from the result set

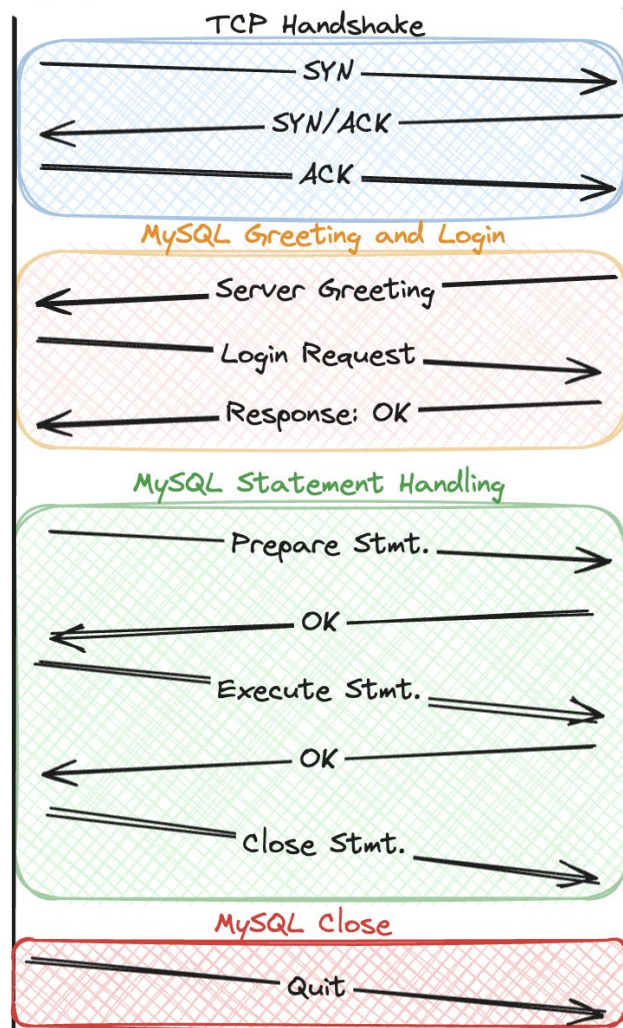
PDO options info: <https://www.php.net/manual/en/pdo.setattribute.php>

(these constants are used in our Repository class)



Parameterized Prepared Statements

- In order to have queries executed in a secure manner, databases must be able to distinguish between (SQL) code and data.
- Using parameterized prepared statements the web server will send the SQL query, and data for that query in two separate requests.



How parameterized prepared statements work

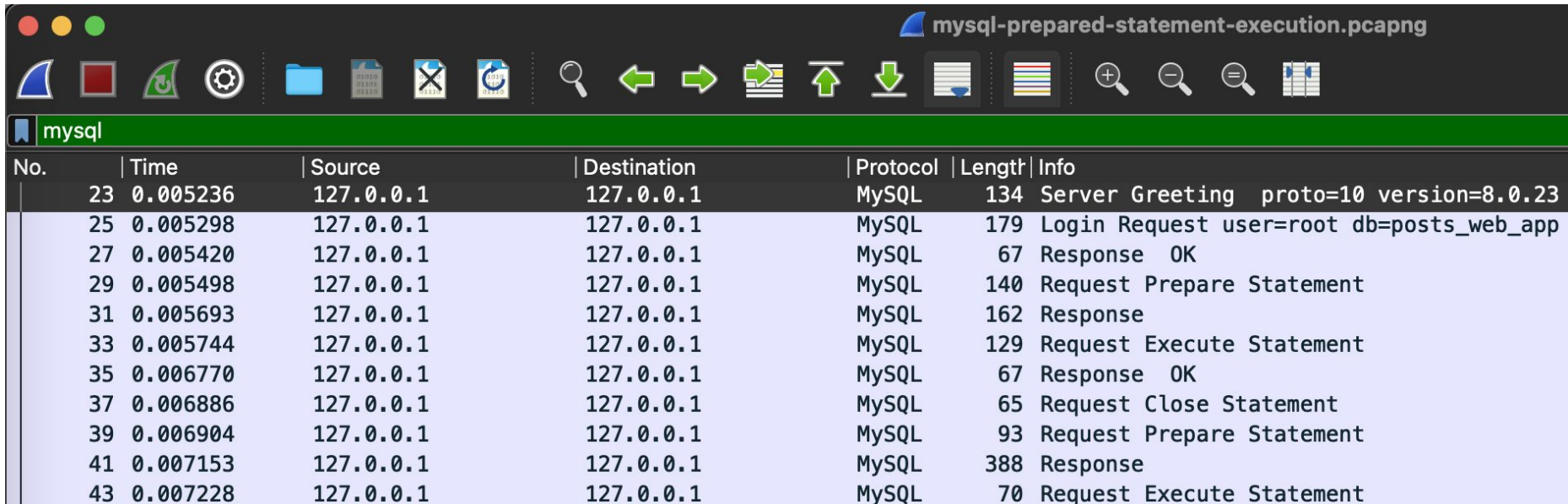
The **prepare statement** request contains the SQL query code. eg.
`INSERT INTO posts (created_at, updated_at, body, title)
VALUES (?, NULL, ?, ?);`

The **execute statement** request contains the data for the query. eg.
"2024-05-25 20:23:09",
"Test Post Body",
"Test Post Title"

Extra Material (outside the scope of the course)

Without protocol documentation how could we check what messages MySQL sends?

Tools such as [Wireshark](#) allow you to analyze messages sent over a network. Since the source and destination IP addresses are the same in this case, checking the source and destination port for each message you'll be able to see which software initiated the request (MySQL is 3306 by default). See **mysql-prepared-statement-execution.pcapng**



No.	Time	Source	Destination	Protocol	Length	Info
23	0.005236	127.0.0.1	127.0.0.1	MySQL	134	Server Greeting proto=10 version=8.0.23
25	0.005298	127.0.0.1	127.0.0.1	MySQL	179	Login Request user=root db=posts_web_app
27	0.005420	127.0.0.1	127.0.0.1	MySQL	67	Response OK
29	0.005498	127.0.0.1	127.0.0.1	MySQL	140	Request Prepare Statement
31	0.005693	127.0.0.1	127.0.0.1	MySQL	162	Response
33	0.005744	127.0.0.1	127.0.0.1	MySQL	129	Request Execute Statement
35	0.006770	127.0.0.1	127.0.0.1	MySQL	67	Response OK
37	0.006886	127.0.0.1	127.0.0.1	MySQL	65	Request Close Statement
39	0.006904	127.0.0.1	127.0.0.1	MySQL	93	Request Prepare Statement
41	0.007153	127.0.0.1	127.0.0.1	MySQL	388	Response
43	0.007228	127.0.0.1	127.0.0.1	MySQL	70	Request Execute Statement



Changing our Repository from JSON Files → Database

- Change the function bodies of **getArticleById**, **updateArticleById**, **deleteArticleById**, etc.
- The interface to your Repository class (function signatures) should not need to change

Lab 4 on D2L



Review

- What is a primary key?
- What is a foreign key?
- What does CRUD stand for?
- What is SQL injection caused by?
- How does using parameterized prepared statements ensure the database can differentiate between code and data?
- Why might we want to have our DB on a separate server?

Todo

- Look into PHP namespaces
 - Read:
 - <https://phptherightway.com/#namespaces>
 - <https://www.php.net/manual/en/language.namespaces.rationale.php>
 - Example of namespaces in the lab this week
- Complete lab 4
- Gain familiarity with how to write PHP applications that interact with a database
 - Assignment 2 (assigned in week ~6) will require these skills
 - Recorded SQL injection demo:
<https://www.youtube.com/watch?v=5SR4891jT3E>