

# CSY2028

## Web Programming

### Topic 8

Tom Butler  
thomas.butler@northampton.ac.uk



# Topic 8

- MySQL continued
  - Saving the database
  - DELETE queries
  - Security problems
  - Prepared statements
  - Auto Increment
  - Multiple tables

## Saving the database

- The data from the database is stored *inside the virtual machine*
- If the virtual machine is destroyed or removed (or you re-create the virtual machine on a different PC) the data is lost!
- To save the data you need to copy the files to the **host** operating system

# Saving the database

- You can do this manually but the Virtual Machine is set up to save the database when it is shut down
- Issue ``vagrant halt`` and the database will be exported to ``database.sql``
- When the server boots it is configured to import the database automatically
- Remember to HALT your virtual machine at the end of the session in order to preserve your data.

# DELETE queries

- DELETE queries work exactly the same as SELECT queries
- They take a *WHERE* clause to search for records to delete
- To delete the record from the person table with the email address [john@example.org](mailto:john@example.org) you can execute the query

```
DELETE FROM person WHERE surname="john@example.org"
```

- Note that with a DELETE query there is no \* before FROM!

# DELETE queries

- DELETE queries can be dangerous
- Running the query

```
DELETE FROM person
```

- Will delete all the records from the table
- And there is no way to get them back!

# DELETE queries

- You should always include a WHERE statement

```
DELETE FROM person WHERE email="john@example.org"
```

- It is also good practice to add a LIMIT statement:

```
DELETE FROM person WHERE email="john@example.org" LIMIT 1
```

- When you know that only one record should be deleted using LIMIT 1 will prevent other records being deleted if there is a missing WHERE clause

# MySQL

- Last week you ended with code that looked something like this:

```
<form action="" method="POST">
    <label>Search for a surname:</label>
    <input type="text" name="surname" />
    <input type="submit" name="submit" />
</form>

<?php

$pdo = new PDO('mysql:host:192.168.56.2;dbname=week10', 'student', 'student');

if (isset($_POST['surname'])) {

    $results = $pdo->query('SELECT * FROM person WHERE surname="' . $_POST['surname'] . '"');

    foreach ($results as $result) {
        //...
    }

}
```



# MySQL

```
$results = $pdo->query('SELECT * FROM person WHERE surname="' . $_POST['surname'] . '');
```

- If "Smith" is entered in the surname box then the query that is executed is:

```
SELECT * FROM person WHERE surname="Smith"
```

- This works, but the user can type anything into the box!

# MySQL

- This can be a problem as users cannot be trusted
- What if they entered abc"123 (with the quotes) into the box?
- The query that would be executed is:

```
SELECT * FROM person WHERE surname="abc"123"
```

- Which is invalid and will cause an error

# MySQL

- MySQL Also allows you to run more than one query at once by separating them by a semicolon ( ; )

```
$pdo->query('SELECT * FROM person WHERE surname="Smith";SELECT * FROM PERSON');
```

- This would actually issue **both** queries

# MySQL

```
$results = $pdo->query('SELECT * FROM person WHERE surname="' . $_POST['surname'] . '');
```

- Given the following query, what would happen in the user typed  
– "; DELETE FROM person ; SELECT "
- Into the surname box?

```
SELECT * FROM person WHERE surname=""; DELETE FROM person; SELECT "
```

# SQL Injection

- This is known as SQL Injection
- This is very dangerous and leaving your database exposed to these exploits can be dangerous:
  - People can get unauthorised access to data
  - Change things without you knowing
  - Delete records

# SQL Injection

- There are two ways to prevent SQL injection
  - 1) escaping quotes
  - 2) Prepared statements

# SQL Injection

- To prevent SQL injection you can *escape* special characters with a slash. For example:

```
$pdo->query('SELECT * FROM person WHERE surname="abc\"123";');
```

- Would actually search the database for the string including the quote:
  - abc"123



# Escaping Quotes

- PDO provides a function called *quote*. This does two jobs:
  - Puts quotes around the string
  - Escapes any special characters

```
echo $pdo->quote('Smith'); //Prints 'Smith' (with quotes)
echo $pdo->quote('abc"123'); //Prints 'acb\"123' (with quotes)
```

- This can be used to prevent SQL injection:

```
$results = $pdo->query('SELECT * FROM person WHERE surname=' . $pdo->quote($_POST['surname']));
```



# Prepared statements

- A second method of solving this problem is *Prepared Statements*
- These are a special type of query that is compiled before it is executed
- Instead of writing a query by building a string using concatenation, you write the query with placeholders in the string which will be replaced when the query is executed

# Prepared Statements

- This is done using the PDO *prepare* function
- When a query is *prepared* it is only stored ready to be executed
- You must also call the *execute* function to send the query to the database

# Prepared Statements

- When you write a query to be prepared you don't need to include quotes or the value you want to use in the query
- Instead you use a *placeholder*
- This placeholder is a name of your choice prefixed with a colon (:)
- e.g.

```
$pdo->prepare('SELECT * FROM person WHERE surname = :name');
```

# Prepared statements

```
$pdo->prepare('SELECT * FROM person WHERE surname = :name');
```

- The *prepare* function returns a PDOStatement Object
- This object is a prepared query which you can later execute, you must store the object in a variable to execute the query:

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE surname = :name');
```

- *By convention this variable is called \$stmt*

# Prepared Statements

- Once you have a `$stmt` object you can execute the query
- When you execute the query using the *execute* function you must provide it an array of replacements for the placeholders you put in the query

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE surname = :name');
```

- This query has a placeholder called "name"

# Prepared Statements

- To provide the replacements for the placeholder you must create an array with the placeholder names as the keys and the values you wish to use in the query as the values:

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE surname = :name');  
  
$criteria = [  
    'name' => $_POST['surname']  
];  
  
$stmt->execute($criteria);
```

- This will send the query to the database
- However, unlike the *query* function, *execute* does not return the records that matched the query

# Prepared Statements

- Once you have executed the query, you can use a foreach() loop to loop over all the records

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE surname = :name');  
  
$criteria = [  
    'name' => $_POST['surname']  
];  
  
$stmt->execute($criteria);  
  
foreach ($stmt as $row) {  
    echo '<p>' . $row['firstname'] . '</p>';  
}
```



# Prepared Statements

- You can also use `$stmt→fetch()` to retrieve a single record:

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE surname = :name');  
$criteria = [  
    'name' => $_POST['surname']  
];  
$stmt->execute($criteria);  
$row = $stmt->fetch();
```



# Prepared Statements

- Fetch() returns one record then moves on to the next. It returns false if there are no more records
- You can use a while loop to loop over the results:

```
while ($row = $stmt->fetch()) {  
    echo '<p>' . $row['firstname'] . '</p>';  
}
```

- Will do the same as

```
foreach ($stmt as $row) {  
    echo '<p>' . $row['surname'] . '</p>';  
}
```

# Prepared Statements

- Prepared statements are slightly more code but they are:
  - More secure as they're immune to SQL injection
  - Often more readable because you don't need to use string concatenation
  - Faster to execute

# Prepared Statements

- Prepared statements can be used for any query type with as many parameters as you like for example:

```
$stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname, birthday)
                        VALUES (:email, :firstname, :surname, :birthday)
');

$criteria = [
    'firstname' => $_POST['firstname'],
    'surname' => $_POST['surname'],
    'email' => $_POST['email'],
    'birthday' => $_POST['birthday']
];

$stmt->execute($criteria);
```

- Which is a lot easier to read than:

```
$pdo->query('INSERT INTO person (email, firstname, surname, birthday)
            VALUES (" . $_POST['email'] . ', " . $_POST['firstname'] .
            ', " . $_POST['surname'] . ', " . $_POST['birthday'] . "')
');
```

# Prepared Statements

- The code for prepared statements can be cut down significantly if you design your code well
- If you name your placeholders in the query with the same names as the fields on your form the criteria array is created for you:

```
$stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname, birthday)
                        VALUES (:email, :firstname, :surname, :birthday)
');

$criteria = [
    'firstname' => $_POST['firstname'],
    'surname' => $_POST['surname'],
    'email' => $_POST['email'],
    'birthday' => $_POST['birthday']
];
$stmt->execute($criteria);
```

```
$stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname, birthday)
                        VALUES (:email, :firstname, :surname, :birthday)
');
$stmt->execute($_POST);
```

```
<form action="add.php" method="POST">
    <label>First name:</label>
    <input type="text" name="firstname" />

    <label>Surname:</label>
    <input type="text" name="surname" />

    <label>Email:</label>
    <input type="text" name="email" />

    <input type="submit" name="submit" value="Submit" />
</form>
```

- This will cause an error because the submit button will be sent as part of the form
- `$_POST['submit']` is set and there is no placeholder for it:

```
$stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname)
                        VALUES (:email, :firstname, :surname)
                        ');
$stmt->execute($_POST);
```

```
<form action="add.php" method="POST">
    <label>First name:</label>
    <input type="text" name="firstname" />

    <label>Surname:</label>
    <input type="text" name="surname" />

    <label>Email:</label>
    <input type="text" name="email" />

    <input type="submit" name="submit" value="Submit" />
</form>
```

- To remedy this you need to remove the submit button from \$\_POST:

```
$stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname)
                        VALUES (:email, :firstname, :surname)
');
$stmt->execute($_POST);
```

# Prepared Statements

```
$stmt = $pdo->prepare('INSERT INTO person (email, firstname, surname, birthday)
                        VALUES (:email, :firstname, :surname, :birthday)
');

$stmt->execute($_POST);
```

- Is both less code and easier to read than

```
$pdo->query('INSERT INTO person (email, firstname, surname, birthday)
            VALUES ("' . $_POST['email'] . '", "' . $_POST['firstname'] . '
            ', "' . $_POST['surname'] . '", "' . $_POST['birthday'] . '"
            ');
```

# Exercise 1

- 1) Ensure you have completed Topic 7's exercises
- 2) Amend Topic 7's exercises to use prepared statements (do not use `$pdo→query()` anywhere, use `$pdo→prepare()` throughout!)
- 3) **Optional** Add a Delete button next to each entry in the list to allow users to be deleted
- You **should not** have any concatenation in your SQL queries after `=`. e.g.

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE email = "' . $email . '"');
```



- Use placeholders instead!
- (Placeholders cannot be used in place of field names)

```
$stmt = $pdo->prepare('SELECT * FROM person WHERE email = :email');
```





# MySQL – Auto Increment

- Every table in a database should have a *primary key*
- This is a piece of data that is unique to each record
- For the *person* table last week we used the *email address* field as the primary key because each person has a unique email address

# Primary Keys

- Sometimes there is not an obvious field for a *primary key*
- When this happens you can use a number to give each record its own ID.
- This column is usually called *id* and will store sequential numbers for each record added
- The first record added will be given the ID 1, the second 2 and so on

# Auto Increment

- In a lot of databases (such as oracle) you have to manually calculate the next available ID in the table
- MySQL provides a feature called Auto Increment that does this for you
- An auto increment column can be added to any table
- It must be the primary key

## Example table

- Consider a table that stores messages posted by a specific *user* that stores the following information:
- Who posted the message (firstname, surname and email address)
- The time the message was posted
- The text of the message

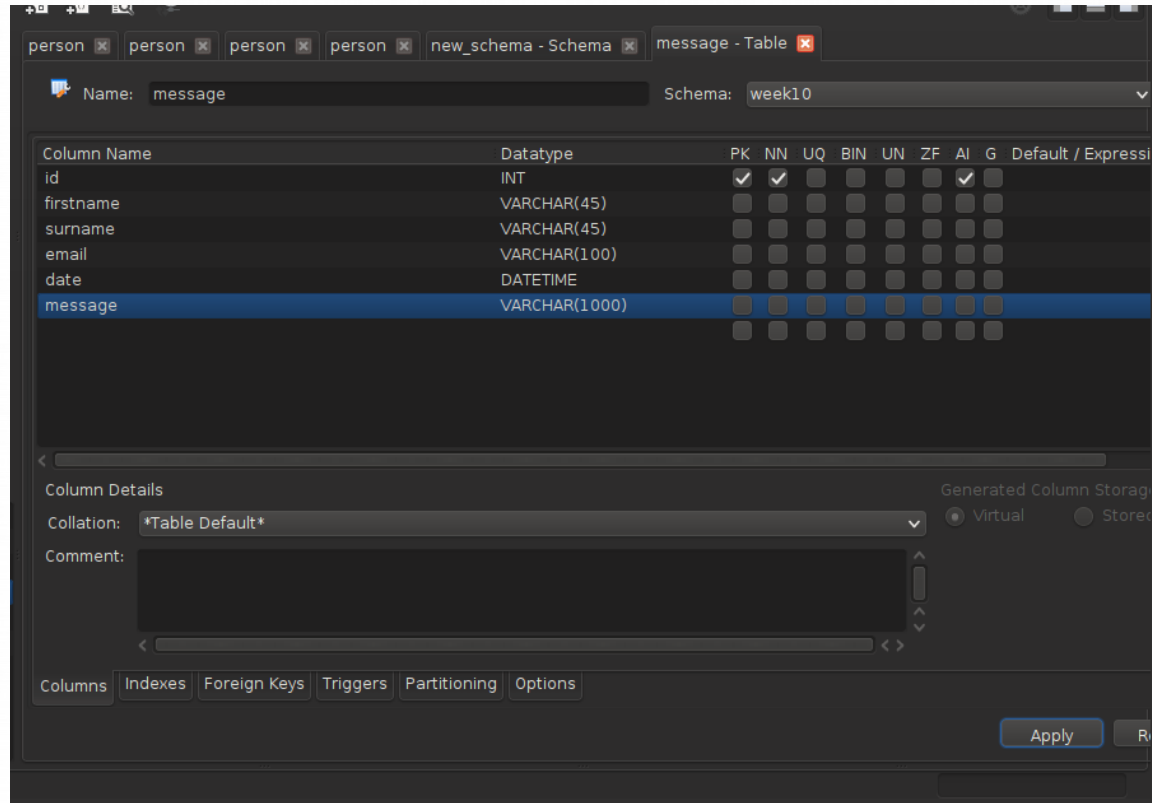
# Messages table

- This table will need the following fields:
  - First name
  - Surname
  - Email address
  - Date of message
  - Message text
- None of these are good candidate for a primary key:
  - Multiple people could share the same name
  - The same person could post two messages (so their email cannot be used)
  - Multiple people could post messages at the same date/time
  - Multiple people could post the same message

## Auto increment

- Because no field is *unique* there is no obvious choice for the primary key
- In this case it would be better to use a sequential number for each record
- This can be done using MySQL's auto\_increment feature
- To create an auto increment column create an INT column and tick the PK and AI boxes in MySQL Workbench

# Auto increment



person x person x person x person x new\_schema - Schema x message - Table x

Name: message Schema: week10

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expressi
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
firstname	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
surname	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
email	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
date	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
message	VARCHAR(1000)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Details

Collation: \*Table Default\*

Comment:

Generated Column Storage: ☒ Virtual ☐ Stored

Columns Indexes Foreign Keys Triggers Partitioning Options

Apply

# Adding records

- Some records can now be added
- When issuing an INSERT query and using auto increment, **if you do not supply a value** for the auto increment column, a number is generated automatically:

```
$stmt = $pdo->prepare('INSERT INTO message (email, firstname, surname, date, message)
                        VALUES (:email, :firstname, :surname, :date, :message)
');
$date = date('Y-m-d H:i:s');
$criteria = [
    'firstname' => 'John',
    'surname'   => 'Smith',
    'email'     => 'john@example.org',
    'date'      => $date,
    'message'   => 'I posted a message!'
];
$stmt->execute($criteria);
```



Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

#	id	firstname	surname	email	date	message
1	John	Smith	john@example.org	2015-12-09 11:38:56	I posted a message!	
	NULL	NULL	NULL	NULL	NULL	

message 1

Apply

# Auto increment

- Each time you add a record it adds to the list.
- You can call execute multiple times without re-writing the query:

```
$stmt = $pdo->prepare('INSERT INTO message (email, firstname, surname, date, message)
                      VALUES (:email, :firstname, :surname, :date, :message)
');

$date = date('Y-m-d H:i:s');

$criteria = [
    'firstname' => 'Sue',
    'surname'   => 'Evans',
    'email'     => 'sue@example.org',
    'date'      => $date,
    'message'   => 'I posted a message!'
];

$stmt->execute($criteria);

$criteria = [
    'firstname' => 'John',
    'surname'   => 'Smith',
    'email'     => 'john@example.org',
    'date'      => $date,
    'message'   => 'I posted another message!'
];

$stmt->execute($criteria);
```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

#	id	firstname	surname	email	date	message
	1	John	Smith	john@example.org	2015-12-09 11:38:56	I posted a message!
	2	Sue	Evans	sue@example.org	2015-12-09 21:44:36	I posted a message!
3	3	John	John	john@example.org	2015-12-09 21:44:36	I posted another m...
	NULL	NULL	NULL	NULL	NULL	

message 4

Apply

# Avoiding duplication

- This causes duplication. Each time someone posts a message their information is stored repeatedly
- If you want to update John's email address you have to update all the records that contain it!
- A better way to handle this is separate tables for each set of information:
  - A table for people
  - A table for messages

## Avoiding duplication

- To achieve this you will need some way of linking between each table so you know who posted which message
- This is done using a *foreign key*
- A foreign key is a field in the table that stores the value of a records *primary key* from another table

# Avoiding duplication

- Firstly, the person table needs to store information about who posted the message:
  - ID (Primary Key, Auto Increment)
  - firstname
  - Surname
  - Email



Name: person

Schema: week10



Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
firstname	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
surname	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
email	VARCHAR(100)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

#### Column Details

Collation: \*Table Default\*

Comment:

#### Generated Column Storage Type

☒ Virtual

☐ Stored

Columns

Indexes

Foreign Keys

Triggers

Partitioning

Options

Apply

Revert

# Avoiding duplication

- The message table can then store the information specific to messages:
  - ID (Primary key, auto increment)
  - The message text
  - The time it was posted
- As well as:
  - UserId (The ID of the user who posted the message)





Name: message

Schema: week10



Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	G	Default / Expression
id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
date	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
message	VARCHAR(1000)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
userId	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

#### Column Details

Collation: \*Table Default\*

Comment:

Generated Column Storage Type

☒ Virtual

☐ Stored

Columns

Indexes

Foreign Keys

Triggers

Partitioning

Options

Apply

Revert

## Removing duplication

- The people's information can be added to the *person* table
- And the *message* information can stay in the *message* table with a reference to the relevant user ID in the *person* table

# Person table

# Message table

Result Grid

Filter Rows:

Edit: Export/Import: Wrap Cell Content:

#	id	date	message	userid
1	1	2015-12-09 11:38:56	I posted a message!	1
2	2	2015-12-09 21:44:36	I posted a message!	2
3	3	2015-12-09 21:44:36	I posted another m...	1
	NULL	NULL	NULL	NULL

Result Grid

Form Editor

Field Types

Query Stats

Execution Plan

message 1

Apply Revert

Message table

#	id	date	message	userId
1	1	2015-12-09 11:38:56	I posted a message!	1
2	2	2015-12-09 21:44:36	I posted a message!	2
3	3	2015-12-09 21:44:36	I posted another m...	1
	NULL	NULL	NULL	NULL

Person table

id	firstname	surname	email
1	John	Smith	john@example.org
2	Sue	Evans	sue@example.org
NULL	NULL	NULL	NULL

# Using multiple tables in PHP

- To use the information in PHP you need two queries, one for each table
  - One to get all of the messages
  - One to query the person table to get the information about the person who posted each message:

```

$pdo = new PDO('mysql:dbname=csy2028;host:v.je', 'student', 'student');

$messageQuery = $pdo->prepare('SELECT * from message');
$userQuery = $pdo->prepare('SELECT * FROM person WHERE id = :id');

$messageQuery->execute();
echo '<ul>';
while ($message = $messageQuery->fetch()) {

    $userCriteria = [
        'id' => $message['userId']
    ];

    $userQuery->execute($userCriteria);

    $user = $userQuery->fetch();

    echo '<li>' .
        $user['firstname'] . ' ' . $user['surname'] .
        ' posted the message <strong>' . $message['message'] . '</strong>' .
        ' on ' . $message['date'] .
        '</li>';

}

echo '</ul>';

```

Connect to database

Prepare a query for each table

Execute the query that returns  
all the messages

Loop through each record  
in the message table

Build the criteria to search for  
a user by their id

Execute the user query and  
search for the user with the ID  
from the message table

Fetch the information from the  
person table that was retrieved  
by the query

Generate HTML using both  
the \$user record and the  
\$message record

- John Smith posted the message **I posted a message!** on 2015-12-09 11:38:56
- Sue Evans posted the message **I posted a message!** on 2015-12-09 21:44:36
- John Smith posted the message **I posted another message!** on 2015-12-09 21:44:36

# Exercise 2

- 1) Add an `id` column to the `person` table and make it the primary key and auto\_increment
- 2) Create a table **message** with the following fields:
  - id (auto increment, int)
  - userId (int)
  - messageDate (dateTime)
  - messageText (varchar)
- 3) Add a form for allowing users to post a message. The form should have two fields:
  - Message text (textarea)
  - User (select box with all the users from the database)
- 4) Add a page for displaying all the messages that have been posted with the message text, date the message was posted and who posted it
- 5) **Optional** Make it so users can **reply** to other messages.
  - Difficulty: Medium/hard – can you use the same HTML form for replies and new messages?