**Лабораторна робота 24.**

**Використання класів-репозиторіїв для сутностей для реалізації додаткового функціоналу.**

**Мета роботи:** навчитись створювати класи-репозиторії для сутностей та реалізовувати більш складні запити до БД із використанням DQL та Native SQL.

**Задачі:**

* Розглянути поняття класу репозиторія то створити його
* Реалізувати операції вибірки додаткових даних за допомогою DQL
* Реалізувати операції вибірки додаткових даних за допомогою Native SQL
* Навчитись використовувати додатково створений функціонал у контролері.
* Виконати індивідуальне завдання.

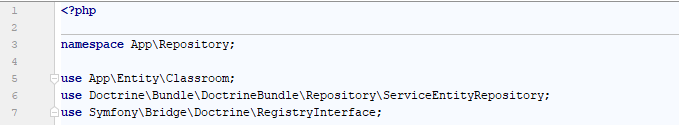
1. Насправді у попередніх роботах ми вже стикалися з об’єктами епозиторіїв та використовували їх для вибірки необхідних нам даних без написання жодного SQL коду. Так, наприклад, пригадаємо, як у попередняй роботі вибирались 2 студенти із кодами 110 та 111 (рис.1)



Рис.1. Використання репозиторію для вибірки даних.

За замовченням нам вже доступні методи find, findBy, findOneBy та findAll. Вони покривають більшість простих задач пошуку даних, але бувають більш складні задачі пошуку, або наприклад одну з найбільш часто використовуваних операцій ми хочемо оформити у метод для більш зручного користування нею. Виходом може стати створення класу-репозиторія для нашої сутності.

Якщо при створенні нового класу-сутності ми використали команду make:entity, то клас репозиторію буде створено автоматично (рис.2) та також буде встановлено зв’язок між основним класом та класом-репозиторієм (рис.3).



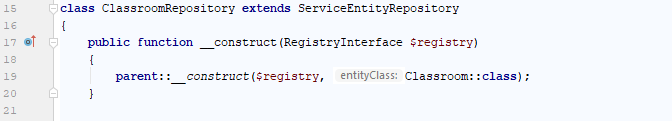


Рис.2. Клас-репозиторій для Classroom.

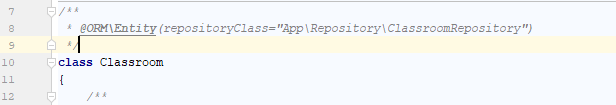
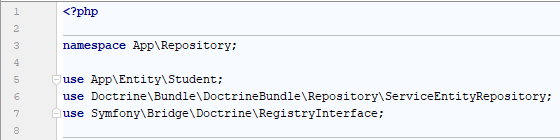
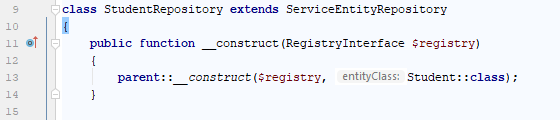


Рис.3. Зв’язок між класом Classroom та його репозиторієм.

Далі у разі необхідності від може бути розширений власними методами для вибірки необхідних даних.

1. Створимо клас репозиторій для сутності «Студент» та реалізуємо у ньому метод, що повертає студентів із певною датою народження. Почнемо із створення у папці «src\Repository» файлу «StudentRepository.php» наступного вмісту (рис.4)





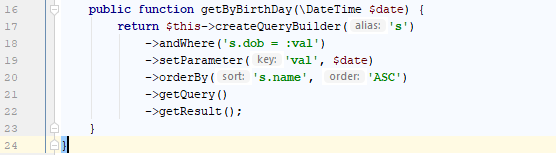


Рис.4. Вміст файлу «StudentRepository.php»

Далі пов’яжемо створений клас із класом-сутністю «студент» (рис.5) та у контролері створимо метод для повернення студентів за датою народження (рис.6). Результат наводимо на рис.7.



Рис.5. Вказання репозиторію у класі «студент»

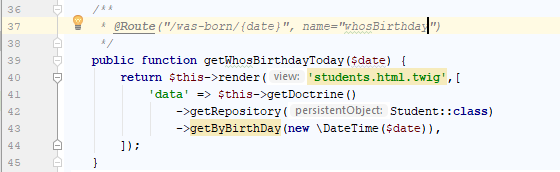


Рис.6. Новий метод у контролері.

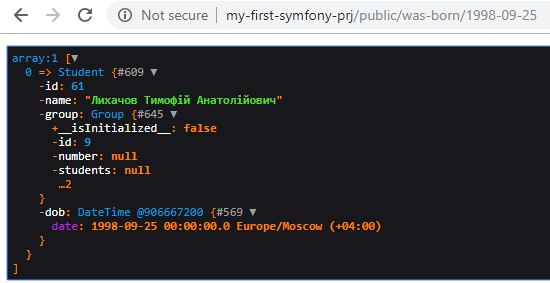


Рис.7. Результат вибірки студентів за датою народження.

1. У попередньому прикладі для написання запиту DQL ми використали метод createQueryBuilder(). Реалізуємо схожу задачу, але із використанням createQuery() та написанням DQL-запиту вручну. Реалізуємо посилання «http://my-first-symfony-prj/public/was-born-between», що повертатиме студентів із датою народження між нижньою та верхньою межею. Додаємо метод у клас-репозиторій (рис.8) та у контролер (рис.9). Перевіряємо результат (рис.10).

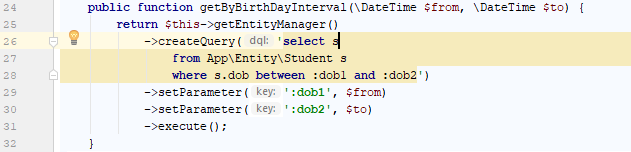


Рис.8. Метод репозиторію для вибірки студентів.

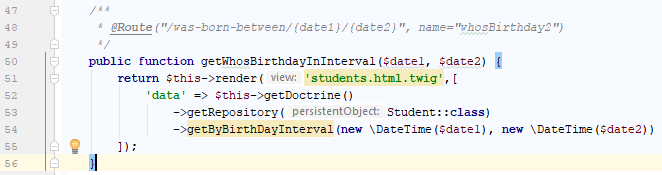


Рис.9. Метод контролеру для повернення результату.



Рис.10. Результат вибірки студентів.

1. Doctrine також підтримує виконання нативних SQL запитів до БД. Якщо у деяких випадках не вистачає функціоналу мови DQL, а таке часто трапляється наприклад у випадках використання деяких специфічних вбудованих функцій або при застосуванні нестандартних зв’язків між таблицями, може бути використаний стандартний SQL запит.

Для прикладу реалізуємо посилання, що буде повертати студентів-іменинників. Як завжди, почнемо із реалізації методу у класі-репозиторії (рис.11), далі додамо обробку адреси у контролер (рис.12) та наведемо зовнішній вигляд отриманої сторінки (рис.13).

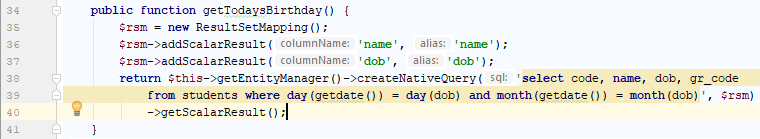


Рис.11. Метод, що повертає студентів-іменинників.

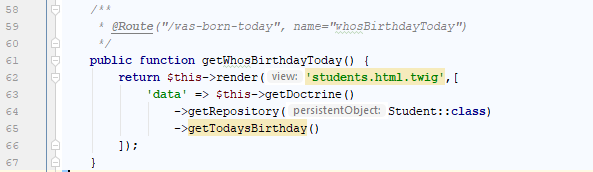


Рис.12. Додавання методу у контролер.

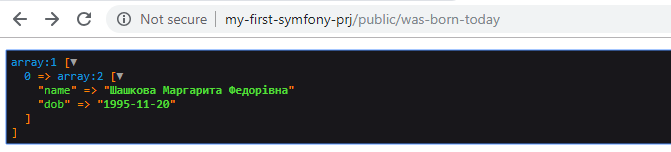


Рис.13. Зовнішній вигляд сторінки.

Можемо звернути увагу, що на зараз наш SQL запит повертає результат у вигляді звичайного масиву. Однак можна виконати перетворення рядків масиву у об’єкти типу «студент». Для цього при виконанні запиту потрібно у якості другого параметру методу createNativeQuery() передати ResultSetMapping, у якому прописати відповідність полів результату запиту полям нашого об’єкту. Змінимо відповідним чином методи для повернення сьогоднішній іменинників (рис.14-15). Звернімо увагу, що метод контролеру, який повертає адресу, змін не зазнає.

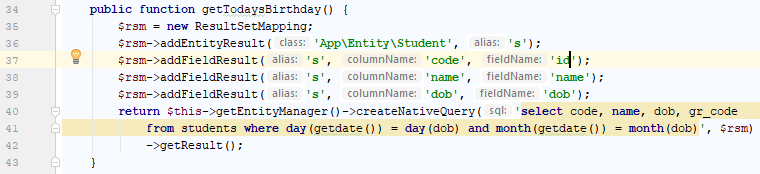


Рис.14. Змінений метод, що повертає студентів-іменинників.

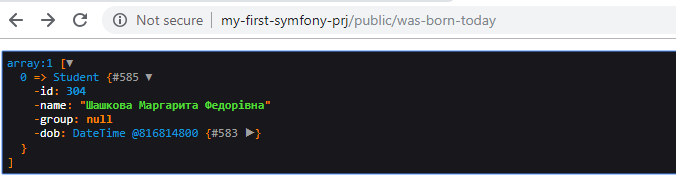


Рис.15. Зовнішній вигляд сторінки після змін.

Додамо до нашого ResultSetMapping обробку вкладеного об’єкту «група» для додавання до об’єкту «студент» інформації про групу, до якої він відноситься (рис.16-17).

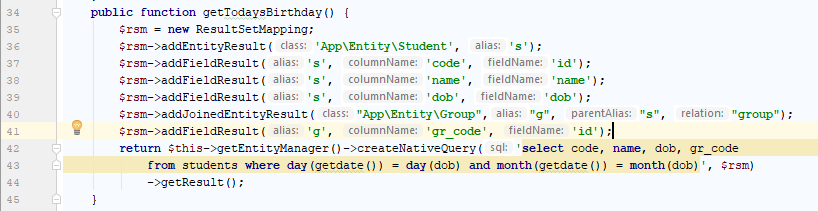


Рис.16. Студенти-іменинники із вказанням групи.

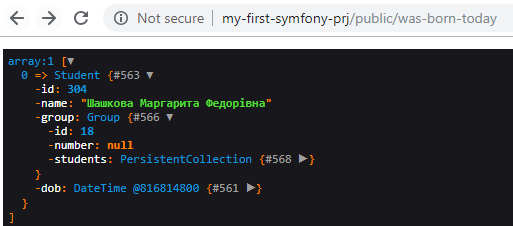


Рис.17. Зовнішній вигляд сторінки після змін.

Звернімо увагу, що у групі на даний момент заповнюється лише її код, як у випадку із LazyLoad. Для повного завантаження атрибутів групи необхідно буде якним чином оновити їх в коді, наприклад у контролері (рис.18) після чого вони стануть видимі у нашому масиві об’єктів (рис.19). Також скориставшись Symfony toolbar можна побачити новий запит до БД (рис.20).

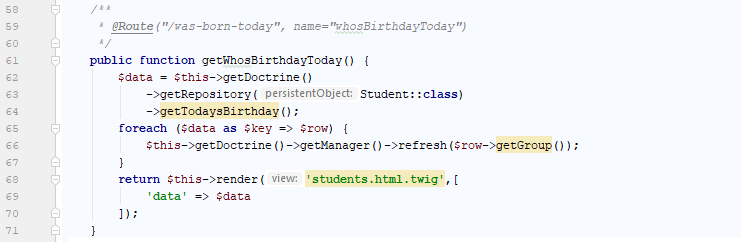


Рис.18. Читання даних по групі для кожного іменинника.



Рис.19. Зовнішній вигляд сторінки після змін.

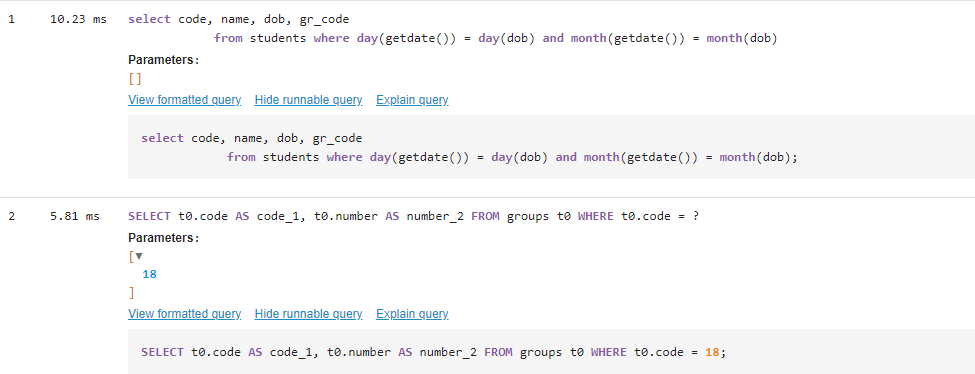
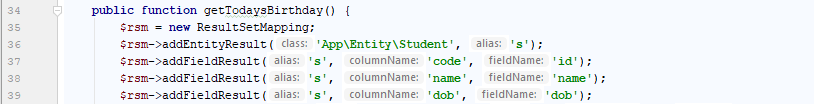


Рис.20. Symfony Profiler. Queries.

Якщо іменинників буде дільше, наприклад 20, то у нас уже буде 21 запит до БД, що зрозуміло не є ефективним підходом по роботі із даними. Бажано виконати за 1 раз вибірку всіх необхідних даних, а вже потім працювати із ними. Реалізуємо це, трохи змінивши метод класу-репозиторію, що повертає іменинників (рис.21) та прибравши операцію оновлення групи, що ми додали на рис.18. Як бачимо на рис.22, група має атрибут number при тому, що всі необхідні дані вибираються із БД за один запит (рис.23).





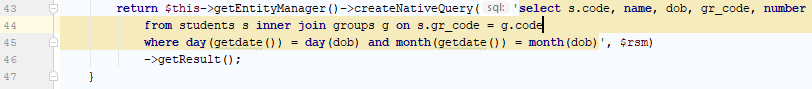


Рис.21. Читання даних по групі для кожного іменинника у запиті.

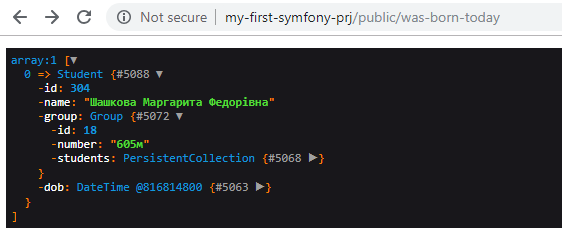


Рис.22. Зовнішній вигляд сторінки після змін.

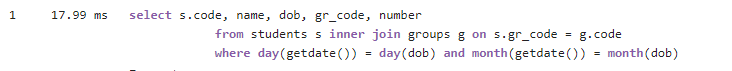


Рис.23. Symfony Profiler. Queries.

1. Використовуючи NaviteQuery Виконаємо збережену процедуру. Створимо процедуру, що повертає сьогоднішніх іменинників (рис.24).

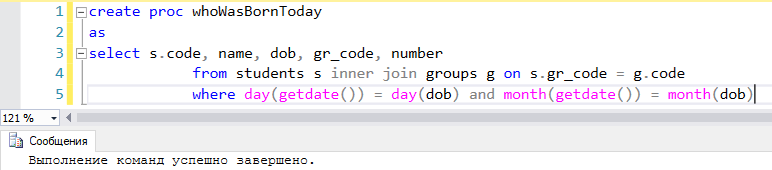


Рис.24. Процедура, що повертає студентів-іменинників.

Далі змінимо метод класу репозиторію, щоб він використовував створену процедуру (рис.25)

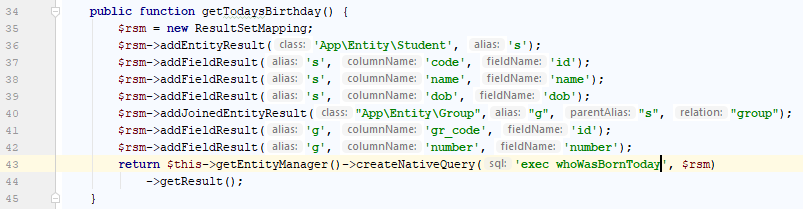


Рис.25. Виконання процедури та обробка результату.

1. Ще один варіант роботи із БД – напряму через об’єкт з’єднання із сервером. Але тут всі результати будуть повертатись у вигляді масивів, і при необхідності працювати із об’єктами класів їх доведеться створювати вручну. Реалізуємо метод, що вибирає студентів за частиною прізвища (рис.26) та використаємо його у контролері (рис.27). Наведемо результат на рис.28.

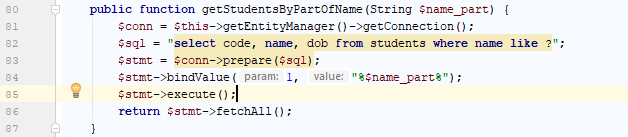


Рис.26. Вибір студентів за частиною прізвища.

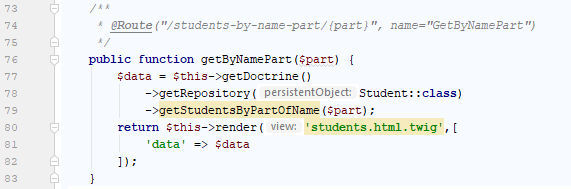


Рис.27. Метод контролера для обробки посилання.

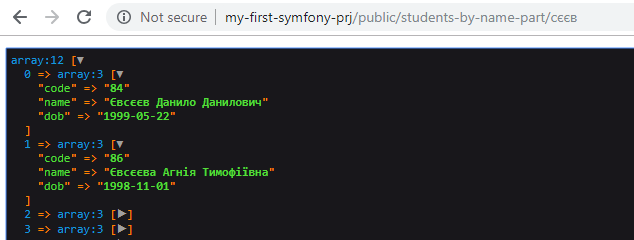


Рис.28. Результат вибірки.

Для повернення даних у вигляді об’єктів класу студент можемо трохи змінити метод контролера, наприклад як наведено нижче (рис.29-30)

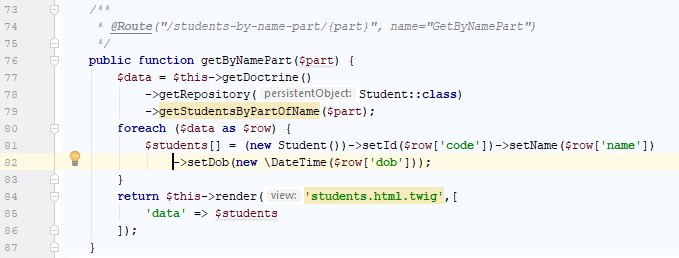


Рис.29. Повернення результату вибірки у вигляді масиву об’єктів.

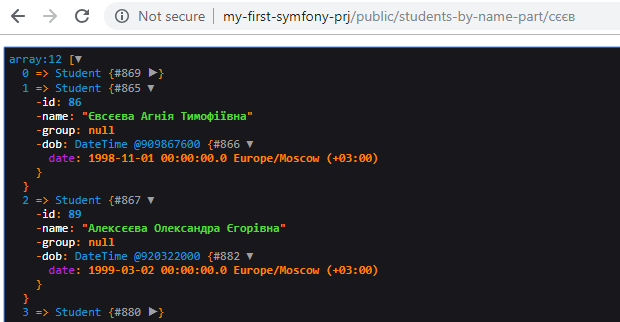


Рис.30. Результат повернення даних у вигляді об’єктів.

1. Кінець роботи.

**Завдання для індивідуального виконання.**

.Для одного з реалізованих класів-сутностей створити клас-репозиторій та реалізувати у ньому методи, що повертають деякі набори даних таблиці класу-сутності із використанням:

* Запиту мовою DQL згенерованого за допомогою QueryBuilder
* Запиту мовою DQL, написаного вручну
* Запит мовою SQL із використанням NaviteQuery
* Запит мовою SQL із використанням getConnection
* Збереженої процедури

Продемонструвати можливість отримання даних у вигляді звичайного масиву та набору об’єктів. Реалізувати контролер, що викликатиме всі реалізовані методи через посилання на сторінці.