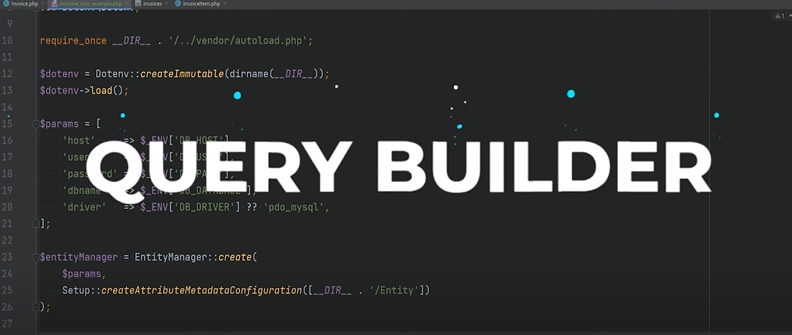
**Video 3.17. Doctrine ORM Query Builder**

1. **Giới thiệu Query builder trong Doctrine ORM**

Tiếp tục với Doctrine ORM và nói về query builder, DQL của nó. Chúng ta có thể tạo một phiên bản query builder bằng cách gọi phương thức createQueryBuilder trên đối tượng EntityManager. Sau đó, chúng ta có thể gọi một số phương thức liên tục trên query builder để thực sự xây dựng câu truy vấn. Vậy hãy xây dựng một số câu truy vấn đơn giản bằng cách sử dụng đối tượng Invoice.

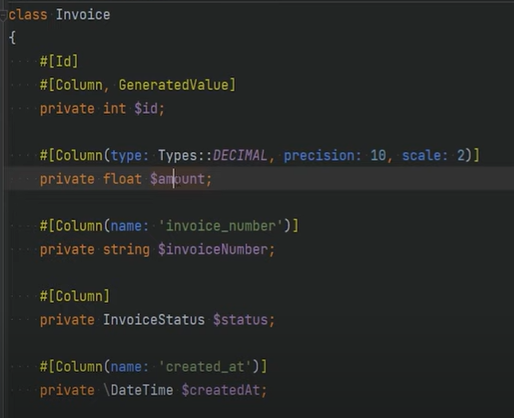
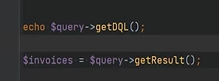
1. **Xây dựng Query builder**

Ví dụ, query builder sẽ chọn (select) và chúng ta sẽ chọn có thể created at và amount từ đối tượng Invoice. Và chúng ta cần truyền bí danh (alias) ở đây như là đối số thứ hai vì vậy đặt giá trị này thành "i". Vì sử dụng bí danh, chúng ta cần thêm nó vào các thuộc tính được chọn ở đây, vì vậy làm i.created at và i.amount. Thêm một số điều kiện đơn giản hoặc mệnh đề WHERE ở đây. Vì vậy chúng ta sẽ thực hiện điều kiện WHERE i.amount lớn hơn một số lượng nào đó. Chúng ta cần đặt tham số này, vì vậy chúng ta sẽ thực hiện set parameter amount và hãy tạm thời thiết lập nó là 100. Sau đó, chúng ta có thể thiết lập kiểu một cách rõ ràng nếu chúng ta muốn và chuyển đối số thứ ba. Nhưng chúng ta không cần nó trong trường hợp này vì kiểu dữ liệu cho một số giá trị được suy ra tự động. Hãy đặt chúng trên từng dòng riêng biệt để dễ đọc hơn. Hãy cũng thêm một số sắp xếp ở đây, vì vậy chúng ta sẽ thêm mệnh đề ORDER BY. Chúng ta sẽ thực hiện ORDER BY và sẽ sắp xếp theo ngày theo thứ tự giảm dần. Sau đó, chúng ta cần thực thi câu truy vấn vì query builder chỉ xây dựng đối tượng câu truy vấn mà không thực sự thực hiện câu truy vấn.



1. **Gọi hàm để khởi chạy**

Chúng ta cần chuyển đổi query builder thành đối tượng câu truy vấn và để làm điều đó, chúng ta có thể gọi phương thức getQuery. Chúng ta sẽ gọi getQuery và ở đây chúng ta có thể đặt giá trị này thành query. Sau đó, chúng ta có thể thực thi câu truy vấn bằng cách gọi phương thức getResult trên đối tượng câu truy vấn. Vì vậy, nếu chúng ta gõ "query" và để IDE tự động hoàn thành, chúng ta thấy rằng chúng ta có một số phương thức có sẵn. Vì vậy, chúng ta có thể chạy phương thức "getResult" và gán kết quả này cho biến "invoices". Tất nhiên, chúng ta có thể nối tiếp cuộc gọi phương thức "getResult" sau phương thức "getQuery" và đặt giá trị này thành "invoices", nhưng tôi chỉ muốn có câu truy vấn như một biến riêng biệt. Mặc dù điều này có vẻ tương tự với query builder của Doctrine DBAL. Mà chúng tôi đã đề cập trước đó, đây không phải là tên bảng ở đây và đây không phải là tên cột, vì trong bảng, "created\_at" được viết là "created\_at" theo quy tắc snake case. Chúng tôi đang sử dụng thuộc tính trên đối tượng Invoice. Nếu chúng tôi xem vào đối tượng Invoice, chúng tôi thấy rằng "amount" và "created\_at" chỉ là các thuộc tính được ánh xạ tới các cột trên bảng. 

Vì vậy, chúng tôi không thực sự xây dựng hay viết SQL ở đây. Chúng tôi đang xây dựng và viết một cái gọi là DQL (Doctrine Query Language). Đừng hoảng, DQL không đáng sợ như có thể nghe.DQL tương tự SQL, nhưng khác với SQL là trong DQL, bạn suy nghĩ dựa trên các thực thể (entities) và các thuộc tính được ánh xạ. 

Chúng ta thực sự có thể kiểm tra câu truy vấn DQL và xem nó là gì bằng cách gọi phương thức getDQL trên đối tượng query hoặc query builder.

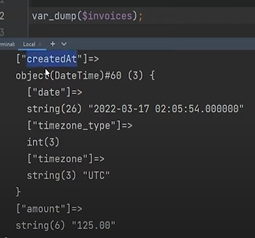
1. **Chạy terminal**

Hãy mở terminal và chạy mã, và như bạn có thể thấy, chúng ta nhận được DQL là kết quả.

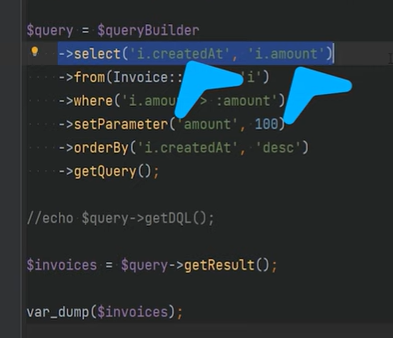
Như bạn có thể thấy, chúng ta đang chọn thuộc tính "created\_at" và "amount" từ thực thể "invoice" và ở đây nó sử dụng tên lớp đầy đủ. Điều kiện là "amount" lớn hơn một giá trị "amount" và chúng ta sắp xếp theo "created\_at".theo thứ tự giảm dần bây giờ, đây là một câu truy vấn DQL khá đơn giản, nhưng chúng ta có thể xây dựng các câu truy vấn phức tạp. Thậm chí chúng ta có thể viết DQL nguyên thủy và thay vì tạo trình tạo câu truy vấn, chúng ta có thể tạo một câu truy vấn và truyền DQL nguyên thủy vào đó, từ đó chúng ta có thể sao chép nó và giả sử rằng chúng ta đã tự viết nó.

Hãy kết thúc phần này và thay vì tạo trình tạo câu truy vấn, chúng ta có thể tạo câu truy vấn ở đây và sử dụng entity manager create query, như bạn có thể thấy nó chấp nhận một chuỗi DQL làm đối số, vì vậy chúng ta có thể dán chuỗi DQL vào đây và sau đó có thể nhận kết quả của câu truy vấn đó. Điều này cho phép chúng ta linh hoạt viết các câu truy vấn phức tạp hơn từ đầu. Bây giờ, hãy xóa phần này tạm thời vì chúng ta sẽ không viết DQL từ đầu, và hãy thực sự in ra thông tin của hóa đơn bằng cách sử dụng.

1. **Dữ liệu**

Tôi sẽ comment phần này và sau đó in ra thông tin của hóa đơn, hãy mở terminal, chạy mã và như bạn có thể thấy, chúng ta đang nhận được danh sách hóa đơn với số tiền lớn hơn 100 và mỗi phần tử hóa đơn chứa hai dữ liệu là ngày tạo và số tiền. 

Bây giờ như bạn có thể nhận thấy, kết quả là một mảng, không phải là các đối tượng hay thực thể, chỉ là một mảng đơn giản chứa danh sách hóa đơn, trong đó mỗi phần tử là một mảng khác chứa các mục mà chúng ta đang lựa chọn. Lý do làm việc như vậy là do cách mệnh đề select được tạo thành. Vì vậy, dòng này có tác động đến cách dữ liệu được truy xuất, ngoài việc bạn có thể truy xuất dữ liệu theo cách cụ thể bằng cách gọi các phương thức. Nhưng vì chúng ta đang chỉ định cụ thể ở đây và xác định các thuộc tính như created at và amount, nó không hydrate đối tượng thực thể mà thay vào đó trả về một mảng chứa các trường đó dưới dạng giá trị. Điều này có ý nghĩa vì nếu nó chọn một đối tượng thực thể và đối tượng thực thể hóa đơn, bạn chỉ hydrate hai thuộc tính trên đó là created at và amount và việc đối tượng thực thể hóa đơn hoạt động theo cách đó sẽ không có ý nghĩa.

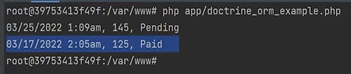


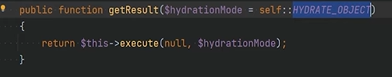
Bởi vì chúng ta có một số thuộc tính khác được ánh xạ vào các cột và nếu chúng ta chỉ lựa chọn amount và created, thì không có ý nghĩa để hydrate một đối tượng thực thể hoàn chỉnh. Đó là lý do tại sao chúng ta nhận được kết quả dưới dạng một mảng để làm việc với các thực thể. Để làm việc với các đối tượng thực thể, chúng ta có thể loại bỏ các chi tiết ở đây và chỉ đơn giản lựa chọn bí danh ở đây. Vì vậy, chúng ta sẽ xóa đi và hãy mở terminal, xóa đi phần này, chạy lại mã và như bạn có thể thấy, bây giờ chúng ta đang nhận được các đối tượng thực thể. Tuy nhiên, điều này không dễ đọc, vì vậy điều tôi sẽ làm là thêm một vòng lặp foreach đơn giản ở đây, với mỗi invoice trong invoices, và chúng ta hãy echo ra thuộc tính created at và chúng ta sẽ định dạng nó theo cách này. Hãy cũng lấy amount và trạng thái của hóa đơn.

Vì vậy, chúng ta sẽ gọi invoice.getAmount() và invoice.getStatus(), và chúng ta có thể gọi toString() trên nó vì kết quả sẽ là trạng thái của hóa đơn trong chúng. Bây giờ, hãy làm sạch một chút và điều này sẽ đủ tốt. Tôi sẽ thêm một khối tài liệu ở đây rằng đây là một thực thể hóa đơn để auto completion hoạt động và chúng ta có thể nhấp vào các phương thức này. 

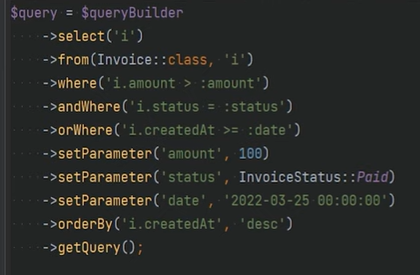
1. **Hydration**

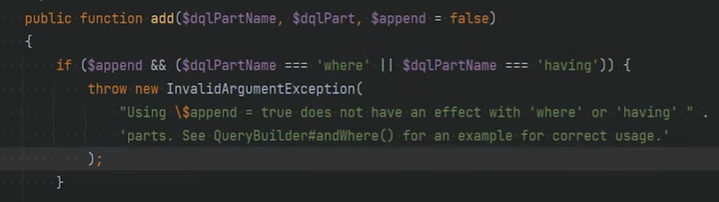
Hãy chạy mã ngay bây giờ và như bạn có thể thấy, nó hoạt động. Điều này có nghĩa là bây giờ chúng ta đang làm việc với các thực thể đã hydrate.

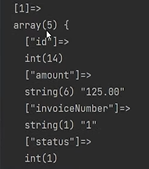
Hydration đơn giản là quá trình chuyển đổi kết quả từ cơ sở dữ liệu thành đối tượng hoặc thực thể. Nó đơn giản là quá trình điền hoặc tải dữ liệu vào một đối tượng.Nếu chúng ta kiểm tra phương thức getResult() ở đây, chúng ta thấy rằng chế độ hydration mặc định được đặt là hydrate object. 

Chúng ta có thể thay đổi nó và hydrate kết quả dưới dạng một mảng nếu chúng ta muốn, hoặc chúng ta có thể đơn giản gọi một phương thức khác được gọi là getArrayResult(), như bạn có thể thấy nó nhận đối số là hydrate array. Vì vậy, thay vì gọi getResult(), chúng ta có thể gọi getArrayResult(). Bây giờ, hãy sử dụng var\_dump(invoices) vì vòng lặp này sẽ không còn hoạt động nữa vì chúng ta không còn làm việc với các thực thể. 

1. **Ứng dụng Hydrate**

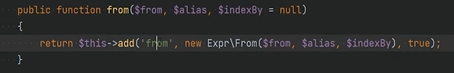
Vì vậy, tôi sẽ comment phần này tạm thời. 

Mở terminal, chạy mã và như bạn có thể thấy, chúng ta quay trở lại với mảng, nhưng bây giờ mảng chứa nhiều thông tin hơn cho mỗi hóa đơn vì đây là các thuộc tính được ánh xạ vào các cột trên thực thể.



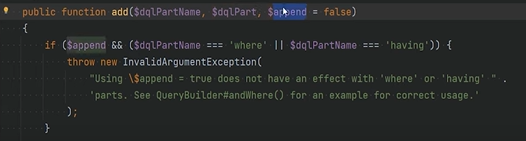
1. **Các phương thức select(), from(), where(), add(), john()...**

Bây giờ, hãy đóng cái này lại và hãy kiểm tra một số phương thức ở đây như select(), from(), và where(). Tôi sẽ kiểm tra phương thức select().

Phương thức này, như bạn có thể thấy, gọi một phương thức khác gọi là add(), và đối số đầu tiên nó truyền vào là select, và đối số thứ hai nó truyền vào là một đối tượng biểu thức nào đó. Hãy kiểm tra phương thức from() ngay bây giờ. 

Lại ở đây, chúng ta gọi phương thức add() và truyền from làm đối số đầu tiên, và một lần nữa nó tạo ra một đối tượng biểu thức nào đó.Hãy kiểm tra phương thức where() và một lần nữa nó gọi phương thức add() với where làm đối số đầu tiên và một đối tượng biểu thức mà nó tạo ra ngay ở đây.

Về cơ bản, các đối tượng biểu thức này hoặc biểu thức là cách để xây dựng các truy vấn phức tạp hơn và các điều kiện phức tạp hơn. Nó thực sự làm cho việc xây dựng các truy vấn và điều kiện phức tạp trở nên linh hoạt hơn và không quá phức tạp. Phương thức add() ở đây chịu trách nhiệm xây dựng DQL. 

Nếu chúng ta kiểm tra phương thức đó, chúng ta thấy rằng nó chấp nhận ba đối số: tên phần DQL, phần DQL và cờ append tùy chọn. Đối số đầu tiên, tên phần DQL, là vị trí hoặc nơi mà phần DQL phải được đặt. Vì như chúng ta đã thấy trong phần select(), from(), và where(), nó chỉ là một chuỗi nơi phần DQL nên được đặt, có thể là select, where, from, group by, và cetera. Đối số thứ hai khá rõ ràng và đó là phần DQL, nó có thể được truyền dưới dạng chuỗi, một mảng hoặc một đối tượng biểu thức.Đối số thứ ba là cờ append tùy chọn, nó quyết định xem liệu nó có nên ghi đè lên các mục đã được xác định trước trong DQL hay không. Lưu ý rằng các phần where và having luôn ghi đè lên, vì vậy cờ này không có tác dụng đối với chúng. Chúng ta có thể thấy trong phần đầu của phương thức này, nó gây ra một ngoại lệ nếu append được đặt thành true và tên phần là where hoặc having. 

Hãy nói rằng chúng ta muốn xây dựng một truy vấn với mệnh đề where như sau: where amount lớn hơn một số lượng nào đó (đó là những gì chúng ta đã có) và status bằng với một trạng thái nào đó hoặc created at lớn hơn hoặc bằng với một ngày nào đó, và chúng ta sẽ đặt nó trong dấu ngoặc đơn để không phức tạp lắm. Nhưng bây giờ, hãy thử xây dựng nó với Query Builder.Bạn có thể nghĩ rằng gọi where nhiều lần sẽ được nối vào truy vấn, nhưng điều đó không hoạt động vì where luôn ghi đè lên.

Vì vậy, nếu chúng ta đặt điều này thành một điều kiện như status bằng với một trạng thái nào đó, điều đó sẽ ghi đè lên phần trước đó. Nếu chúng ta thực sự echo DQL ở đây và chạy mã, chúng ta sẽ chỉ có một mệnh đề where duy nhất, vì vậy chúng ta không thể sử dụng nhiều cuộc gọi phương thức where theo cách này. Lựa chọn khác là Doctrine thực sự có các phương thức gọi là andWhere() và orWhere() mà chúng ta có thể thử và sử dụng. Ví dụ, chúng ta có thể sử dụng andWhere() trong trường hợp này và chạy mã và chắc chắn rồi, bây giờ chúng ta đã có phần end.

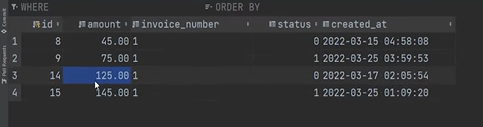
Bây giờ hãy thêm orWhere() vào đây và chúng ta sẽ tạo điều kiện created at lớn hơn hoặc bằng với một ngày nào đó. Chúng ta cũng cần đặt các tham số, vì vậy chúng ta sẽ đặt trạng thái là invoice status paid, sau đó chúng ta sẽ đặt ngày là 25 tháng 3. Hãy chạy mã lại và đây là phần where mà nó tạo ra. Tôi sẽ sao chép nó và đặt vào đây để chúng ta có thể so sánh.

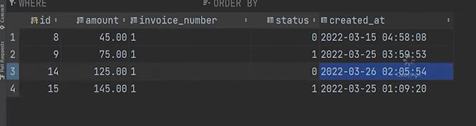
Như bạn có thể thấy, nó không hoàn toàn giống như chúng ta thực sự mong muốn. Chúng ta muốn dấu ngoặc đơn nằm xung quanh điều kiện or ở đây, nhưng thay vào đó, nó đặt dấu ngoặc đơn xung quanh điều kiện end.

Vì vậy, hai truy vấn này không giống nhau và nó có thể gây ra một số lỗi khó chịu. Để khắc phục điều này và tạo ra loại truy vấn như vậy, chúng ta có thể thay vì sử dụng endWhere() và orWhere() ở đây, chúng ta có thể xây dựng biểu thức where tùy chỉnh của chúng ta. Vì vậy, chúng ta sẽ tắt hai phần này và thay vào đó sử dụng biểu thức tùy chỉnh. Chúng ta có thể xây dựng các biểu thức bằng cách gọi phương thức trên query builder, vì vậy chúng ta sẽ làm queryBuilder.expr. Sau đó, nếu chúng ta sử dụng tính năng tự động hoàn thành, chúng ta sẽ thấy nó có một số phương thức trợ giúp mà chúng ta có thể sử dụng để xây dựng biểu thức tùy chỉnh hoặc phức tạp. Đối với phần đầu tiên, chúng ta cần phần end, vì vậy chúng ta sẽ sử dụng endX().

Phương thức này chấp nhận danh sách các biểu thức làm đối số, vì vậy chúng ta cần xây dựng phần này và các phần khác của biểu thức. Một lần nữa, chúng ta hãy thử xây dựng biểu thức và sử dụng phương thức greaterThan() và truyền i amount làm đối số đầu tiên và amount parameter làm đối số thứ hai. Chúng ta sau đó có thể đặt dấu phẩy và tiếp tục xây dựng biểu thức của chúng ta. Bây giờ chúng ta cần xây dựng phần này, như bạn có thể thấy, nó có một điều kiện or ở đây, vì vậy chúng ta cần làm một cái gì đó tương tự như chúng ta đã làm ở đây, nhưng thay vì sử dụng chỉ mục, chúng ta cần gọi orX().Chúng ta sẽ sao chép điều này và đặt vào đây, sau đó sử dụng orX() và đặt trong dấu ngoặc đơn. Bây giờ chúng ta cần truyền điều kiện này và điều kiện này vào, vì vậy chúng ta sẽ làm một cái gì đó tương tự như thế này, và thay vì dùng greaterThan, nó sẽ là equals, và i status sẽ bằng một loại trạng thái nào đó. Và đối với phần thứ hai, nó sẽ là greaterThanOrEqual, và chúng ta sẽ thay đổi điều này thành created at, và điều này thành date. Bây giờ hãy loại bỏ điều này, và chúng ta cũng cần lấy giá trị ở đây để truyền cho trạng thái, vì vậy hãy mở terminal và chạy mã. 

Như bạn có thể thấy, bây giờ phần where trông chính xác. Chúng ta có thể sao chép nó và đặt ngay dưới đây, và như bạn có thể thấy, nó hoàn toàn giống nhau. Sự khác biệt duy nhất là ở đây tôi sử dụng tên cột, nhưng ở đây chúng ta có tên thuộc tính vì đó là DQL. Hãy loại bỏ điều này và hãy thử lấy kết quả. Tôi sẽ thay thế điều này để lấy kết quả, chúng ta sẽ loại bỏ var\_dump(), hãy bỏ comment ở đây, hãy chạy mã và như bạn có thể thấy, nó hoạt động và chúng ta đang nhận được một hóa đơn duy nhất vì đó là duy nhất hóa đơn thỏa mãn các kiểm tra này. Nếu chúng ta mở bảng hóa đơn ở đây, chúng ta thấy rằng chúng ta chỉ có hai hóa đơn có giá trị lớn hơn 100 và chỉ có một hóa đơn đã thanh toán và chỉ có một hóa đơn thỏa mãn kiểm tra ngày.

Nếu chúng ta thay đổi điều này thành ngày 26 tháng 3 ví dụ, và cập nhật nó, sau đó chúng ta chạy mã, chúng ta thấy rằng bây giờ chúng ta đang nhận được hai hóa đơn, vì vậy mọi thứ đều hoạt động như mong đợi.

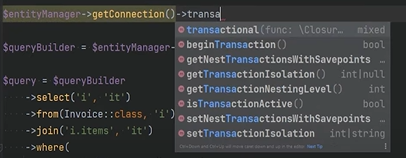
Điều này rất tuyệt đúng không? Chúng ta thậm chí có thể xây dựng một trình tạo truy vấn tùy chỉnh bao bọc xung quanh trình tạo truy vấn của Doctrine và sử dụng biểu thức để xây dựng các truy vấn phức tạp bằng cách sử dụng các phương thức trôi chảy để không phải viết quá nhiều biểu thức thủ công như thế này.Vậy giờ chúng ta hãy làm thêm một ví dụ nữa nơi chúng ta chọn các mục liên quan để chúng ta có thể gọi phương thức join() ở đây và làm đối số đầu tiên chúng ta cần truyền tên của mối quan hệ hoặc liên kết. 

Nếu chúng ta kiểm tra thực thể invoice, chúng ta thấy rằng tên là items, vì vậy chúng ta có thể làm i.items và đối số thứ hai là bí danh, vì vậy chúng ta sẽ để nó tạm thời. Sau đó, trong phần select, chúng ta cũng có thể chọn các mục, vì vậy chúng ta sẽ làm ở đây. Và bây giờ nó sẽ chọn các mục hóa đơn cũng. Vậy giờ chúng ta hãy xuống đây và var\_dump() kết quả mảng được lấy và hãy thêm câu lệnh exit. Hãy xóa điều này, chạy mã và như bạn có thể thấy, mỗi hóa đơn cùng với các thuộc tính của nó cũng chứa các thuộc tính liên quan được gọi là items chứa các thuộc tính của thực thể invoice item. Nếu chúng ta muốn chỉ chọn một số phần từ invoice items, chúng ta có thể làm điều đó bằng cách chỉ định các thuộc tính ở đây, vì vậy chúng ta có thể làm it.description ví dụ. Chạy mã lại và như bạn có thể thấy, chúng ta chỉ nhận được mô tả. Bây giờ nếu chúng ta muốn làm việc với các thực thể, tất nhiên nó cũng sẽ hoạt động, vì vậy hãy loại bỏ điều này ở đây, sau đó trong vòng lặp, tôi sẽ dán một đoạn mã nhỏ để in ra một số thông tin về mục. Như bạn có thể thấy, chúng ta đang lặp qua từng mục và chúng ta đang in ra mô tả, số lượng và đơn giá. Hãy mở terminal, xóa điều này, chạy mã và như bạn có thể thấy, nó hoạt động như mong đợi. 

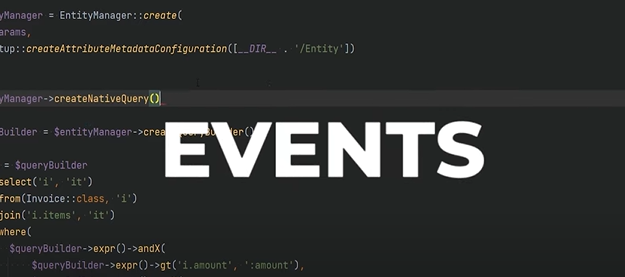
1. **Các phương thức kết nối**

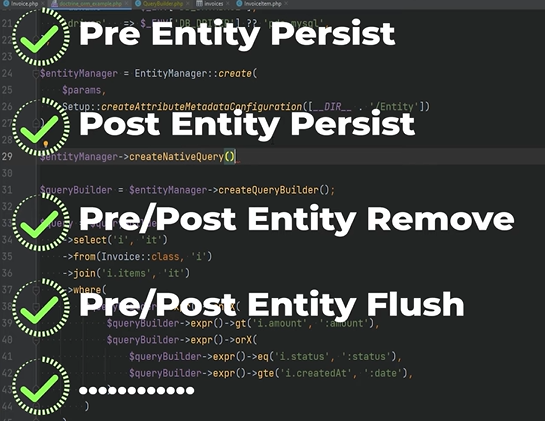
****

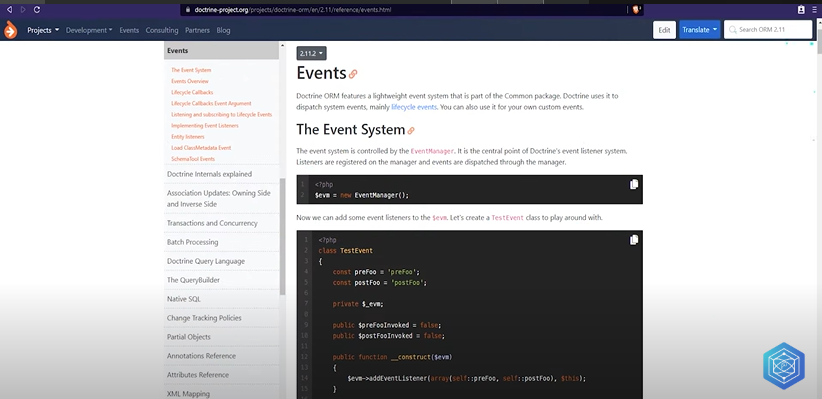
Ngoài khả năng xây dựng truy vấn, chúng ta cũng có thể gọi các phương thức kết nối trên EntityManager như beginTransaction(), commit() và rollback().

Ví dụ, nếu chúng ta cuộn lên đây, chúng ta có thể làm một cái gì đó như entityManager->getConnection()->beginTransaction() hoặc entityManager ->transactional(function() { ... }) và cũng có thể gọi các phương thức liên quan đến giao dịch trực tiếp trên EntityManager như beginTransaction()commit(), rollback() hoặc thay vì transactional(), chúng ta có thể sử dụng wrapInTransaction() và bọc khối mã trong giao dịch. Chúng ta cũng có thể chạy các truy vấn SQL nguyên thủy (Native SQL) nếu chúng ta muốn thay vì xây dựng DQL hoặc viết DQL thô. Thay vì gọi createQueryBuilder(), chúng ta có thể gọi một phương thức được gọi là createNativeQuery() và truyền chuỗi SQL làm đối số đầu tiên, sau đó truyền kết quả ánh xạ làm đối số thứ hai. 

Chúng ta không có thời gian để bao hết tất cả những gì Doctrine ORM hoặc DQL có thể cung cấp như chạy các truy vấn cập nhật và xóa, lưu cache repository, các truy vấn phức tạp hơn và nhiều hơn nữa. Đó là lý do tại sao tôi rất khuyến khích và đề xuất bạn kiểm tra tài liệu nếu bạn muốn tìm hiểu và biết thêm về Doctrine nói chung. Như thường lệ, tôi sẽ để lại các liên kết cần thiết trong phần mô tả. Mục đích chính của việc tìm hiểu Doctrine trong loạt bài này là giới thiệu cho bạn về ORM và một số khía cạnh trừu tượng cấp cao mà bạn gặp trong các framework. Nếu bạn muốn xem một hướng dẫn hoặc khóa học đầy đủ về Doctrine, hãy để lại một bình luận bên dưới và nếu có đủ yêu cầu, tôi có thể thêm nó vào danh sách của mình. Một điều mà tôi muốn đề cập là các sự kiện vòng đời (life cycle events).

Các sự kiện vòng đời là các sự kiện kích hoạt trước hoặc sau khi một điều gì đó xảy ra. Điều này có thể là sự thay đổi trạng thái của một thực thể ví dụ như trước khi lưu trạng thái, sau khi lưu trạng thái, trước và sau khi xóa hoặc trước và sau khi gửi tin nhắn flash và cùng với nhiều sự kiện khác.

Do đó, điều này cho phép chúng ta kết nối vào các sự kiện này và cung cấp các bộ lắng nghe tùy chỉnh để thực hiện một số việc cụ thể. Hệ thống Doctrine 7 được điều khiển bởi lớp EventManager, tương tự như EntityManager, nó là một trung tâm điều khiển các sự kiện. 

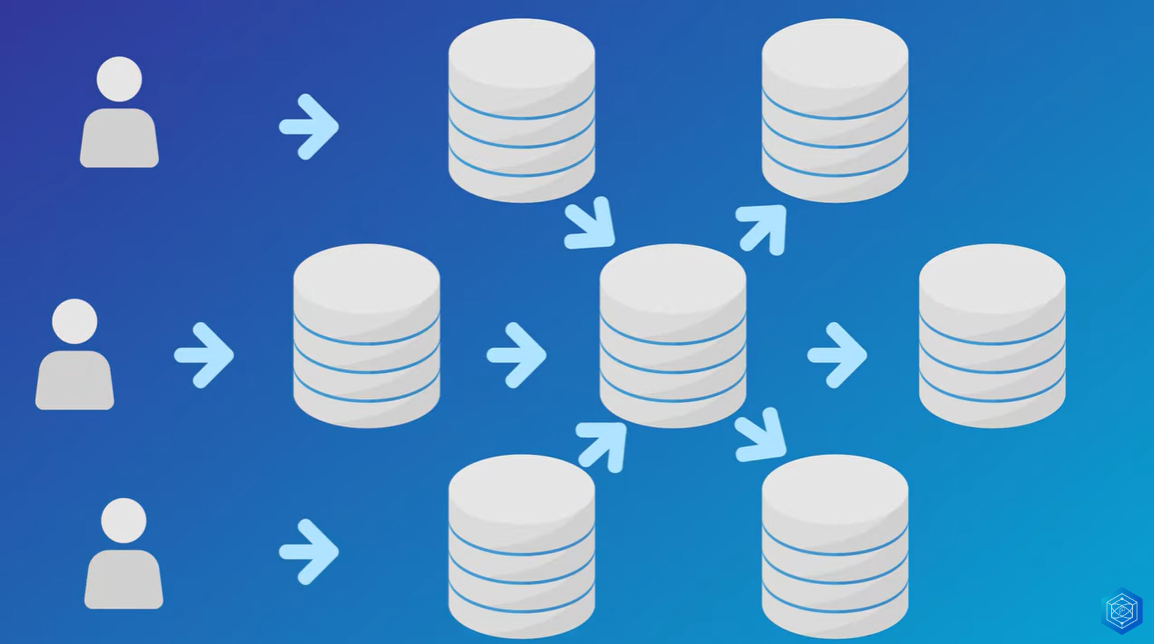
Nếu chúng ta xem tài liệu, chúng ta có thể thấy rằng chúng ta có thể tạo các lớp sự kiện tùy chỉnh và phát chúng bằng cách sử dụng EventManager. Nếu chúng ta cuộn xuống một chút nữa, chúng ta thấy rằng chúng ta có thể đăng ký các bộ lắng nghe sự kiện và người đăng ký. Đây là các lớp mà cơ bản nhận được thông báo khi xảy ra các sự kiện cụ thể. Khi đến với các thực thể, chúng ta có một số phương thức gọi lại vòng đời mà chúng ta có thể tạo như được hiển thị trong bảng ở đây. Chúng được tự động phát đi mỗi khi xảy ra các sự kiện cụ thể, chẳng hạn như prePersist() xảy ra trong lần gọi persist ban đầu, postPersist() được kích hoạt trong quá trình flush và cùng với nhiều sự kiện khác. Hãy mở lớp Invoice và xem làm thế nào chúng ta có thể sử dụng các phương thức vòng đời. Chúng ta có thuộc tính createdAt được ánh xạ vào cột createdAt trong bảng.Nó tương tự như một dấu thời gian, nơi chúng ta muốn đặt ngày và giờ hiện tại khi hóa đơn được tạo. Như bạn có thể nhớ trong bài học trước, chúng ta có ví dụ này, trong đó chúng ta đang tạo một thực thể hóa đơn với các mục hàng và chúng ta đang thiết lập thuộc tính createdAt bằng cách gọi phương thức này và đặt nó thành ngày và giờ hiện tại. Bây giờ, nếu chúng ta muốn tự động hóa điều này và không cần thiết lập nó bằng tay, nếu chúng ta chỉ xóa phần này đi và để cho hóa đơn tự động thiết lập thời gian createdAt ngay trước khi nó được tạo. Vì vậy, nếu chúng ta đến đây, chúng ta muốn kết nối vào một trong các phương thức vòng đời gọi là prePersist, để chúng ta có thể thêm một phương thức nào đó ở đây gọi là onPrePersist, và phương thức này nhận đối số là lifecycleEventArgs, và bên trong đây chúng ta sẽ thiết lập createdAt theo cách này, vì vậy chúng ta sẽ làm như sau: createdAt = new DateTime(). Bây giờ chúng ta có thể loại bỏ phương thức setCreatedAt vì chúng ta không còn cần nó nữa, và chúng ta cũng cần thêm thuộc tính ở đây để ghi rõ rằng điều này được kích hoạt trước, vì vậy chúng ta sẽ thêm thuộc tính @PrePersist. Chúng ta cũng cần thêm một thuộc tính vào lớp thực thể để đánh dấu rằng nó có các phương thức gọi lại vòng đời.Vậy giờ chúng ta hãy mở terminal và kiểm tra điều này. Chúng ta sẽ chạy mã, vậy hãy mở bảng hóa đơn, làm mới và đúng như dự kiến, hóa đơn đã được tạo và createdAt được thiết lập tự động. Đó là tất cả cho bài học này.

**Video 3.18. Database Migrations là gì và làm cách nào để sử dụng Doctrine Migrations**

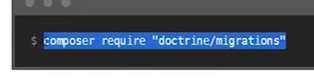
1. **Giới thiệu**



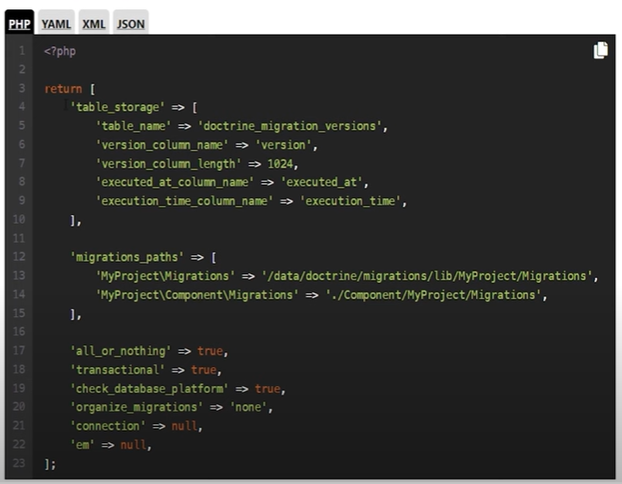
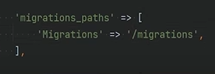
Doctrine có một thư viện khác gọi là 'migrations' cung cấp các tính năng và chức năng bổ sung trên nền tảng DBAL và ORM để quản lý phiên bản của cấu trúc cơ sở dữ liệu. Migrations giống như một hệ thống phiên bản cho cơ sở dữ liệu của bạn, giống như khi sử dụng Git, bạn có thể có nhiều thành viên trong nhóm kéo mã nguồn về và làm việc trên nó với migrations, bây giờ họ cũng có thể thiết lập cơ sở dữ liệu đúng cách và luôn đồng bộ được.



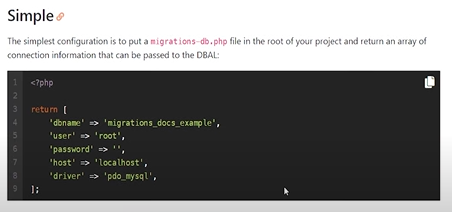
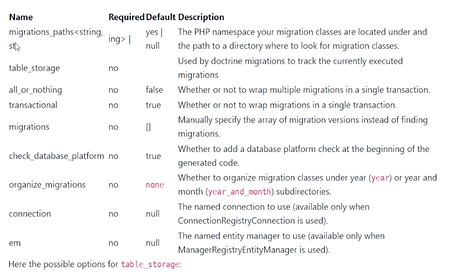
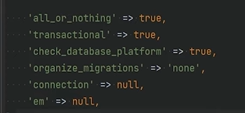
1. **Hướng dẫn cài đặt môi trường**

Chúng ta có thể cài đặt thư viện migrations bằng cách sử dụng composer, vì vậy chúng ta sẽ mở cửa sổ terminal và chạy lệnh 'composer require doctrine migrations'. 

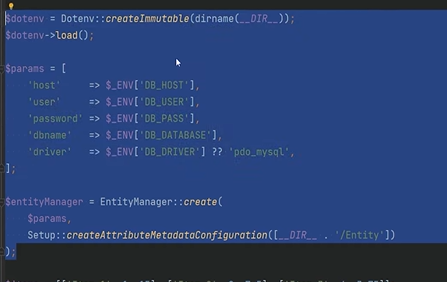
Tiếp theo, chúng ta cần tạo tệp cấu hình migrations gọi là migrations.php và chúng ta cần sao chép đoạn mã nhỏ này và dán vào đó.Sau đó, tôi sẽ sao chép nội dung này và mở mã nguồn dự án trong thư mục gốc của dự án.

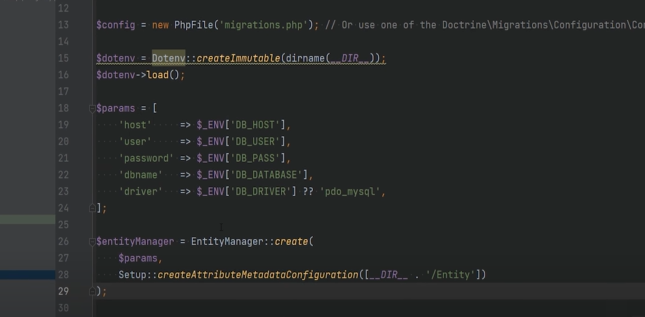
Chúng ta sẽ tạo tệp migrations.php và dán nội dung đó vào đó. Table storage là nơi lưu trữ các phiên bản migration. Doctrine sử dụng nó để theo dõi xem những migration nào đã được thực thi. Chúng ta sẽ để giá trị mặc định cho các tùy chọn này, nhưng chúng ta cần cập nhật các đường dẫn migration. Doctrine sẽ tìm kiếm các tệp migration từ các đường dẫn này, vì vậy chúng ta sẽ thay đổi "my project migrations" thành chỉ "migrations" và cũng sẽ thay đổi đường dẫn thành chỉ "migrations" nữa. 

Đây là không gian tên và đây là đường dẫn thực tế. Chúng ta sẽ giữ nguyên các tùy chọn còn lại với giá trị mặc định của chúng. Bạn có thể kiểm tra tài liệu để biết mô tả về các tùy chọn này và một số tùy chọn khác mà bạn có thể sử dụng, nhưng hiện tại chúng ta đang sử dụng giá trị mặc định và đã đủ.

Tiếp theo, chúng ta cần cấu hình kết nối cho migrations của chúng ta, vì Doctrine cần biết nơi và cách kết nối vào cơ sở dữ liệu. Cách đơn giản là tạo tệp migrationsdbconfig.php trả về một mảng các thông số kết nối như được hiển thị trong tài liệu. Đây là các tùy chọn cấu hình giống như khi chúng ta thiết lập kết nối dbal trong bài học về dbal. Vì chúng ta đang làm việc với orm và có một ứng dụng sử dụng entitymanager, chúng ta có thể sử dụng kết nối từ ứng dụng hiện có sử dụng entitymanager. Vì vậy, nếu chúng ta cuộn xuống một chút ở đây, chúng ta thấy một đoạn mã chứa entitymanager mà chúng ta có thể sử dụng. 

Hãy sao chép đoạn mã đó và chúng ta cần tạo tệp cli config.php và dán mã đó vào đó.Chúng ta cần nhập tệp php ở đây và chúng ta thực sự cần chỉnh sửa một chút. Như bạn nhớ trong các bài học trước, chúng ta đã làm việc với tệp php của playground mà chúng ta đã thiết lập entitymanager và những thứ tương tự, vì vậy chúng ta có thể sao chép một số mã từ đó. Chúng ta có thể sao chép phần entitymanager, sao chép phần parameters và sao chép phần tải các biến môi trường. Các bước:

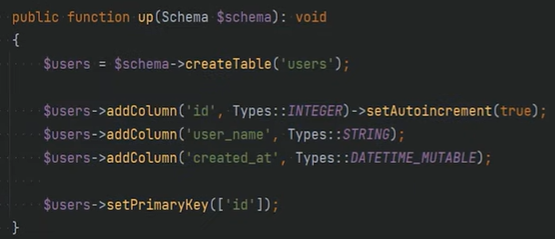
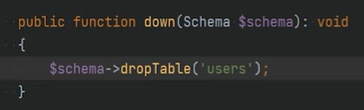
* Bước 1: Copy đoạn code này ở file doctrine\_orm\_example.php
* Bước 2: Sang file Cli\_config.php vừa tạo và bôi đoạn mã sau
* Bước 3: Paste đoạn mã vừa copy vào vị trí bôi đen



* Bước 4: Vào terminal gõ lệnh và bạn sẽ có thông tin địa chỉ của version

1. **Up() và down()**

Hãy mở lớp migration để chúng ta có thể xem ý nghĩa thực sự của các tùy chọn up và down. Các lớp migration phải mở rộng lớp migration trừu tượng và chúng phải cung cấp ít nhất các phương thức up và down. Phương thức up được chạy khi migration được thực thi theo chiều thuận, vì vậy chúng ta đặt logic tạo bảng ở đây hoặc sửa đổi cột và vân vân.

Phương thức down được thực thi mỗi khi migration được hoàn nguyên hoặc rollback, vì vậy chúng ta sẽ đặt mã ở đây để hoàn ngược những gì đã được thực hiện trong phương thức up của ứng dụng. 

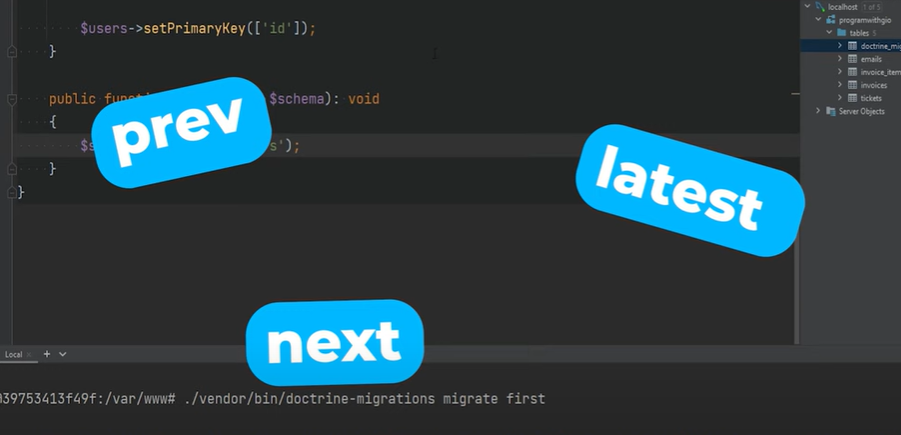
Ví dụ, nếu chúng ta đang tạo một bảng trong phương thức up của ứng dụng, chúng ta sẽ muốn xóa bảng đó trong phương thức down. Nếu chúng ta đang xóa một bảng hoặc cột trong phương thức up, chúng ta sẽ muốn tạo bảng hoặc cột đó trong phương thức down, và cứ thế tiếp tục.

Chỉ một lưu ý nhanh ở đây là bạn không nên hoàn nguyên hay rollback migrations trên môi trường sản phẩm vì nó có thể làm mất dữ liệu trực tiếp. Tuy nhiên, trong quá trình phát triển, nó rất hữu ích vì vậy hãy ghi nhớ rằng khi sử dụng migrations trong môi trường sản phẩm, bạn không muốn hoàn nguyên hay rollback bất kỳ migrations nào. Chúng ta có thể ghi đè lên một số phương thức được cung cấp bởi lớp migration trừu tượng để mở rộng chức năng.Ví dụ, nếu chúng ta cuộn xuống đây, chúng ta sẽ thấy rằng chúng ta có một số phương thức như pre-app và post-app được kích hoạt trước và sau khi phương thức up được thực thi. Chúng ta cũng có các phương thức như pre-down và post-down được thực thi trước và sau khi phương thức down được thực thi và cứ thế tiếp tục. Vậy hãy thực tạo bảng với migration này. Chúng ta sẽ tạo một bảng mới là users mà chúng ta chưa có hiện tại. 

Như bạn có thể thấy, các phương thức up và down có quyền truy cập vào đối tượng schema, và chúng ta có thể sử dụng đối tượng schema để tạo bảng, cột và các thành phần khác. Vì vậy, chúng ta có thể làm như sau: schema create table users. Chúng ta có thể gán giá trị này cho một biến, sau đó chúng ta có thể thêm các cột vào bằng cách gọi phương thức add column trên đối tượng table. Vì vậy, hãy thêm các cột id, username và created\_at. Chúng ta có thể truyền kiểu dữ liệu đúng ở đây và như đối số thứ ba, chúng ta có thể truyền một số tùy chọn nếu muốn. Hãy đặt primary key cho cột id và chúng ta cũng cần đặt cột id để tự động tăng dần.Bây giờ trong phương thức down, chúng ta cần hoàn ngược những gì đã được thực hiện trong phương thức up và trong trường hợp này chúng ta đang tạo bảng users, vì vậy chúng ta cần xóa bảng users trong phương thức down. Một lần nữa, chúng ta sẽ sử dụng đối tượng schema và gọi phương thức drop table, chúng ta truyền users làm đối số. Hãy chạy migrations. Chúng ta có thể chạy một tệp migration đơn hoặc chúng ta có thể chạy tất cả các migration cần thiết để đạt đến phiên bản mới nhất. Hãy chạy nhiều migrations ngay cả khi chúng ta chỉ có một tệp migration duy nhất. Lệnh cho điều đó là doctrine migrations migrate. Hãy nhấn Enter và dường như nó được thực thi mà không có lỗi nào. Hãy mở và làm mới cơ sở dữ liệu và chắc chắn rằng chúng ta có bảng users. Ngoài bảng users, chúng ta cũng thấy bảng doctrine migration versions được tạo ra. Đây là một bảng mà doctrine sử dụng để theo dõi các phiên bản. Nếu chúng ta mở nó, chúng ta sẽ thấy rằng lớp migration này đã được thực thi vào thời gian này và mất bao lâu để thực thi migration đó.Điều này sẽ chạy migrations cho đến điểm đó nếu chúng ta không cung cấp đối số. Mặc định, nó sẽ chạy phương thức up cho tất cả các migrations từ phiên bản hiện tại cho đến phiên bản mới nhất. Chúng ta có thể rollback hoặc revert migrations bằng cách cung cấp một phiên bản cũ hơn so với phiên bản hiện tại, hoặc chúng ta có thể truyền bí danh phiên bản đầu tiên làm đối số, điều đó sẽ rollback migrations cho đến phiên bản đầu tiên. Điều đó có nghĩa là nó sẽ chạy phương thức down trên tất cả các migrations cho đến phiên bản đầu tiên. Nếu chúng ta làm mới cơ sở dữ liệu bây giờ, chúng ta sẽ thấy rằng bảng users đã biến mất và bảng doctrine migration versions trống rỗng.

1. **Prev, next, latest**

Các bí danh phiên bản khác mà chúng ta có thể sử dụng là prev, next và latest, trong đó prev sẽ rollback về phiên bản trước, next sẽ migrate đến phiên bản tiếp theo và latest sẽ migrate cho đến phiên bản mới nhất.

Chúng ta cũng có thể lấy trạng thái của các migrations bằng cách chạy lệnh status. Vì vậy, thay vì migrate chúng ta sẽ dùng status và như bạn có thể thấy, nó cung cấp cho chúng ta một bảng tóm tắt về trạng thái của các migrations của chúng ta.

1. **Chạy thử và sửa lỗi**

Ngoài việc viết migrations bằng tay như thế này, chúng ta cũng có thể tạo ra migrations dựa trên sự khác biệt giữa các entities và schema cơ sở dữ liệu thực tế, điều này rất hữu ích khi làm việc với ORM và entities. Vậy hãy chạy lệnh div và xem nó làm gì và như bạn có thể thấy, nó đã tạo ra một tệp migration mới.Vậy hãy mở tệp migration đó và như bạn có thể thấy, nó đang xóa bảng emails và tickets, và lý do cho điều đó là vì chúng ta không có các entities tương ứng cho các bảng emails và tickets.

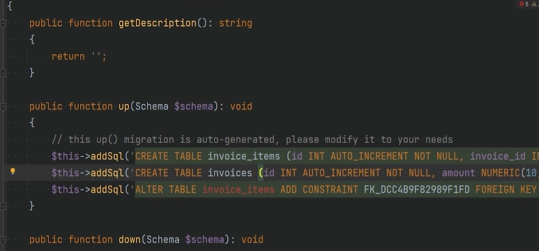
Đây là các bảng được tạo ra trong một số bài học trước đó nhưng chúng ta chưa tạo các entities cho chúng và khi chúng ta chạy lệnh div, doctrine có thể xác định sự khác biệt giữa schema cơ sở dữ liệu hiện tại và schema cơ sở dữ liệu có thể được tạo ra từ entities hiện tại của chúng ta và vì chúng ta không có các entities cho emails và tickets, nó cho chúng ta biết rằng chúng ta cần xóa các bảng này để đồng bộ entities và schema cơ sở dữ liệu hiện tại. Điều này rất hữu ích để giữ cho schema cơ sở dữ liệu đồng bộ với các entities vì nếu chúng ta thay đổi các entities như tạo ra một mapping khác hoặc thay đổi mapping hoặc tạo ra một entity hoàn toàn mới, chúng ta không cần phải tạo thủ công các lớp migration cho nó, chúng ta chỉ cần tạo ra sự khác biệt và để nó tự động tạo ra cho chúng ta. Bây giờ điều tôi sẽ làm là xóa tất cả các bảng và chạy lệnh div một lần nữa để tạo ra schema cơ sở dữ liệu ban đầu, sau đó chúng ta sẽ thực hiện một số thay đổi và chạy lệnh div một lần nữa để xem điều gì xảy ra. Tôi sẽ xóa tất cả các bảng này từ đây. 

Chúng ta cũng sẽ xóa các tệp migration này. 

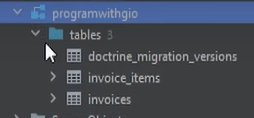
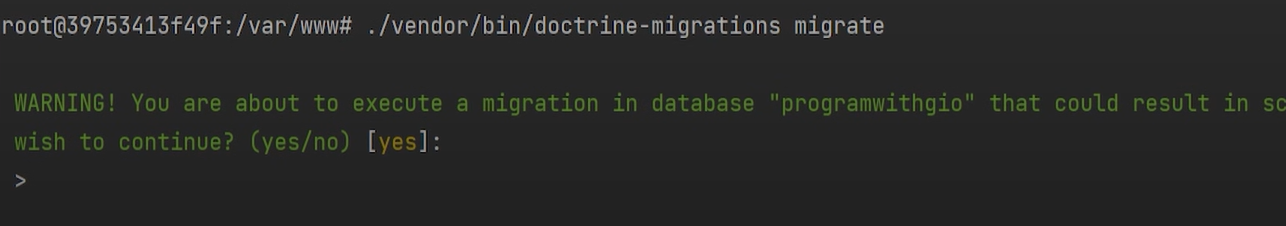
Hãy tạo lại lệnh div và chắc chắn nó đã tạo ra tệp migration. 

Hãy mở tệp migration đó và như bạn có thể thấy, nó đang tạo bảng invoices và invoice items với foreign key đúng.





Hãy chạy lệnh migrate sau đó làm mới cơ sở dữ liệu và như bạn có thể thấy, bảng invoice và invoice items đã được tạo cùng với bảng migration versions và giờ đây schema cơ sở dữ liệu và entities của chúng ta đã đồng bộ.



Bây giờ chúng ta hãy thêm một mapping mới vào entity invoice, giả sử rằng chúng ta muốn thêm trường due date cho hóa đơn. Vì vậy, hãy mở entity invoice và chúng ta sẽ nhân bản trường created\_at và chỉnh sửa thành due\_date. Tất nhiên, nếu chúng ta cố gắng tạo entity này, nó sẽ thất bại vì chúng ta không có cột due\_date trong bảng database.







Một lựa chọn là tạo migration thủ công hoặc chúng ta có thể chạy lệnh div để xem liệu điều này có tự động được tạo ra không. Vậy hãy chạy lệnh div một lần nữa. Nó đã tạo ra một tệp migration mới. Hãy mở tệp migration đó và như bạn có thể thấy, nó đã phát hiện sự khác biệt một cách chính xác và đang thêm một cột mới due\_date vào bảng invoices. Hãy migrate cơ sở dữ liệu, mở bảng invoices và cột đã được thêm vào. Như bạn có thể đã nhận thấy, ngoài việc tạo bảng và cột. Vậy hãy mở tệp migration đó và như bạn có thể thấy, nó đang xóa các bảng emails và tickets. Lý do cho việc này là vì chúng ta không có các entities tương ứng cho các bảng emails và tickets. Đây là các bảng được tạo trong một số bài học trước đó, nhưng chúng ta chưa tạo các entities cho chúng. Khi chúng ta chạy lệnh div, Doctrine có thể phát hiện sự khác biệt giữa schema cơ sở dữ liệu hiện tại và schema cơ sở dữ liệu có thể được tạo ra từ các entities hiện tại của chúng ta. Vì chúng ta không có các entities cho emails và tickets, Doctrine thông báo cho chúng ta rằng chúng ta cần xóa các bảng này để đồng bộ hóa các entities và schema cơ sở dữ liệu hiện tại. Điều này rất hữu ích để giữ cho schema cơ sở dữ liệu đồng bộ với các entities, vì nếu chúng ta thay đổi các entities như tạo một mapping khác, thay đổi mapping hoặc tạo một entity mới hoàn toàn, chúng ta không cần phải tạo thủ công các lớp migration cho nó. Chúng ta chỉ cần tạo ra sự khác biệt và để Doctrine tự động tạo ra các lớp migration.