

# View Order History



# View Order History

- Allow users to view their previous order history
- This feature is only available to logged in / authenticated users

The screenshot shows the luv2shop website interface. At the top, there is a navigation bar with the logo 'luv2shop', a search bar, and a 'Search' button. To the right of the search bar, it says 'Welcome back Demo Darby!' followed by 'Logout', 'Member', and 'Orders'. The 'Orders' button is highlighted with a red dashed circle and a red arrow pointing to it from the bottom left. On the far right of the top bar, it shows '\$0.00' and a shopping cart icon with '0' items. Below the navigation bar, on the left side, there is a sidebar with links: 'Books', 'Coffee Mugs', 'Mouse Pads', and 'Luggage Tags'. The main content area is titled 'Your Orders' and contains a table with two rows of order data.

Order Tracking Number	Total Price	Total Quantity	Date
51542a04-e16d-4e3e-9549-7e3394558f8d	\$36.98	2	Feb 13, 2021, 10:39:57 AM
0e8014f2-0cd9-4ce4-bb5a-e451154ee9f0	\$17.99	1	Feb 13, 2021, 11:59:58 AM

# Front End and Back End

- Split up the development across multiple videos
- We'll start working on Spring Boot back end
- Then we'll work on Angular front end

# Development Process - Spring Boot

*Step-By-Step*

1. Create OrderRepository REST API
2. Update configs to make OrderRepository REST API read only

# Step 1: Create OrderRepository REST API

- Expose endpoint to search for orders by the customer's email address

HTTP Method	Endpoint
GET	<code>/api/orders/search/findByCustomerEmail?email=demo@luv2code.com</code>

# Step 1: Create OrderRepository REST API

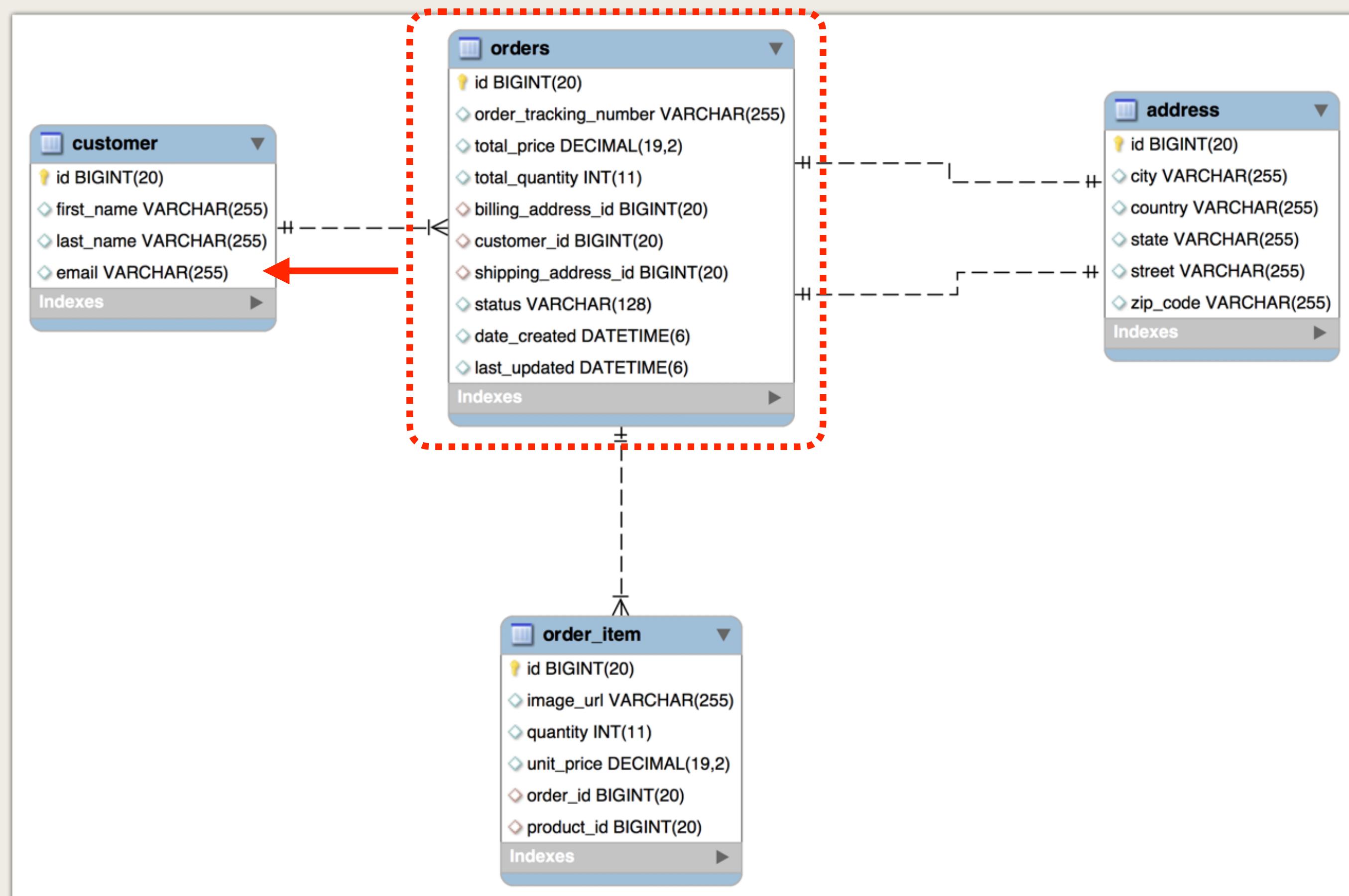
- Spring Data REST and Spring Data JPA supports "query methods"
- Spring will construct a query based on method naming conventions

File: OrderRepository.java

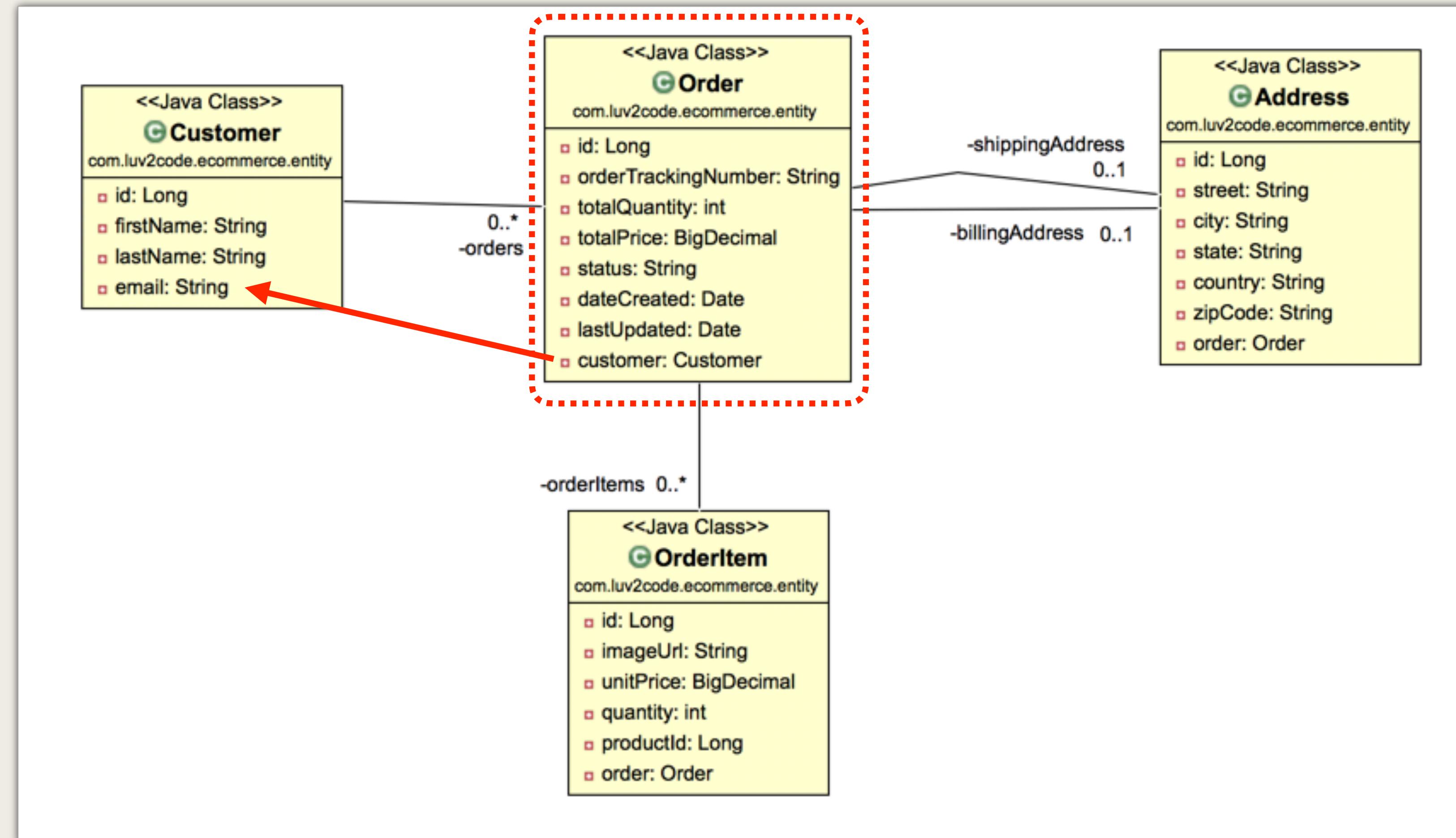
```
@RepositoryRestResource
public interface OrderRepository extends JpaRepository<Order, Long> {

    Page<Order> findByCustomerEmail(@Param("email") String email, Pageable pageable);
}
```

# Database Diagram



# Class Diagram - Entity Classes



# Step 1: Create OrderRepository REST API

File: OrderRepository.java

```
@RepositoryRestResource  
public interface OrderRepository extends JpaRepository<Order, Long> {  
  
    Page<Order> findByCustomerEmail(@Param("email") String email, Pageable pageable);  
}
```

Behind the scenes, Spring will execute a query similar to this

SELECT \* FROM orders  
LEFT OUTER JOIN customer  
ON orders.customer\_id=customer.id  
WHERE customer.email=:email

# Step 1: Create OrderRepository REST API

File: OrderRepository.java

```
@RepositoryRestResource  
public interface OrderRepository extends JpaRepository<Order, Long> {  
  
    Page<Order> findByCustomerEmail(@Param("email") String email, Pageable pageable);  
}
```

<http://localhost:8080/api/orders/search/findByCustomerEmail?email=demo@luv2code.com>

To pass data to REST API

# Step 2: Update configs to make OrderRepository REST API read only

File: MyDataRestConfig.java

```
@Configuration
public class MyDataRestConfig implements RepositoryRestConfigurer {

    @Override
    public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config, CorsRegistry cors) {

        HttpMethod[] theUnsupportedActions = {HttpMethod.PUT, HttpMethod.POST,
                                              HttpMethod.DELETE, HttpMethod.PATCH};

        // disable HTTP methods for ProductCategory: PUT, POST and DELETE
        disableHttpMethods(Product.class, config, theUnsupportedActions);
        disableHttpMethods(ProductCategory.class, config, theUnsupportedActions);
        ...
        disableHttpMethods(Order.class, config, theUnsupportedActions); ←
    }

    ...
}
```