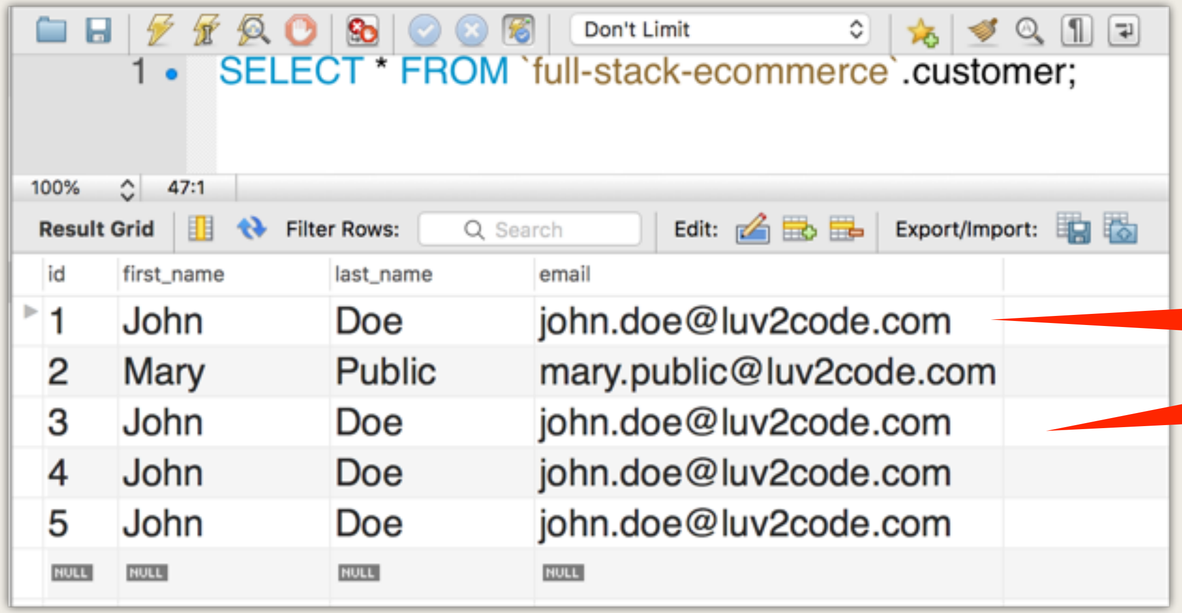
# Handle Customers by Email

# Code Refactoring

- Currently, if we perform multiple checkouts with **same email address**
- We have multiple customer entries in customer table



# Our Solution

- A single customer is associated with multiple orders
- On the backend, in our CheckoutServiceImpl
  - Check database if customer already exists based on email
  - If so then use the existing customer from database
  - Else we have a new customer

# Development Process


1. Remove existing data from database tables

2. Modify database schema to only allow unique email addresses

3. Add new method to CustomerRepository
    1. findByEmail(...)

4. Update CheckoutServiceImpl
    1. Check if customer already exists ... if so then use the existing customer

# Step 1: Remove existing data from database tables

***Run this SQL in MySQL Workbench***

```
USE `full-stack-ecommerce`;

-- clean up previous database tables

SET FOREIGN_KEY_CHECKS=0;

TRUNCATE customer;
TRUNCATE orders;
TRUNCATE order_item;
TRUNCATE address;

SET FOREIGN_KEY_CHECKS=1;
```

**the `truncate` command removes ALL rows from a table**

**In general**
**- `truncate` is faster than `delete`**
**- `truncate` resets auto_increment back to starting value**

# Step 2: Modify database schema to only allow unique email addresses

***Run this SQL in MySQL Workbench***

```
USE `full-stack-ecommerce`;

-- clean up previous database tables

SET FOREIGN_KEY_CHECKS=0;

TRUNCATE customer;
TRUNCATE orders;
TRUNCATE order_item;
TRUNCATE address;

SET FOREIGN_KEY_CHECKS=1;


-- make the email address unique

ALTER TABLE customer ADD UNIQUE (email);
```

**Database constraint**

**MySQL will throw an error if you attempt to insert duplicate email**

# Step 3: Add new method to CustomerRepository

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

File: CustomerRepository.java

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {

    Customer findByEmail(String theEmail);

}
```

Magic!

# Step 3: Add new method to CustomerRepository

- Find a customer based on email

File: CustomerRepository.java

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {

    Customer findByEmail(String theEmail);

}
```

Method returns null if not found

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

```
SELECT * FROM Customer c WHERE c.email = theEmail
```

# Step 4: Update CheckoutServiceImpl

File: CheckoutServiceImpl.java

```
public PurchaseResponse placeOrder(Purchase purchase) {

    // retrieve the order info from dto
    Order order = purchase.getOrder();
    ...
    ...
    // populate customer with order
    Customer customer = purchase.getCustomer();

    // check if this is an existing customer ...
    String theEmail = customer.getEmail();

    Customer customerFromDB = customerRepository.findByEmail(theEmail);

    if (customerFromDB != null) {
        // we found them ... let's assign them accordingly
        customer = customerFromDB;
    }

    // add the order to the customer
    customer.add(order);

    // save to the database
    customerRepository.save(customer);

    // return a response
    return new PurchaseResponse(orderTrackingNumber);
}
```

Method returns null if not found