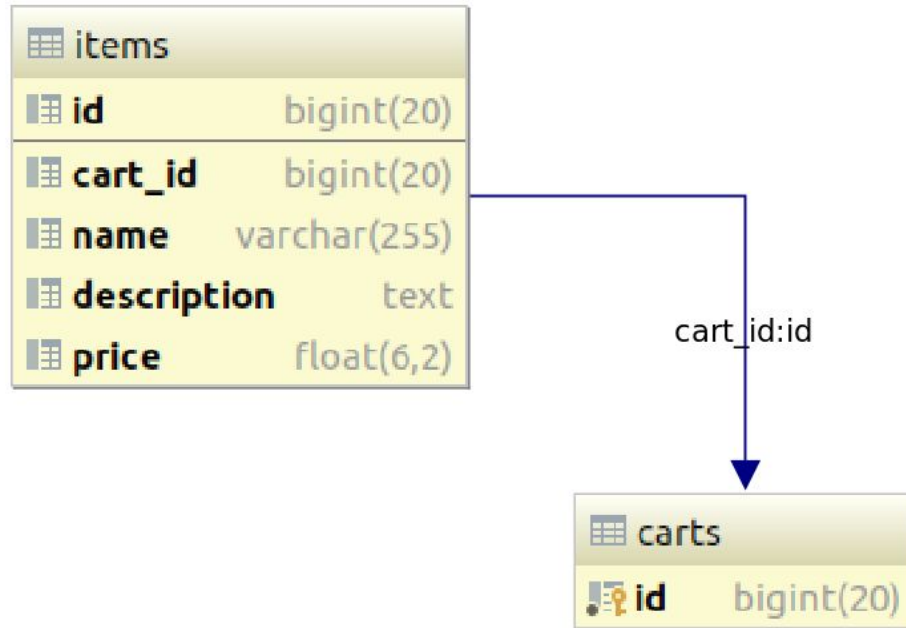


# JAVA II

JPA & debug

Create databases with one to many relation by schema provided below



# Create Cart, Item data models

- Look at you cart table and create relevant Cart entity object for that table. Use @OneToMany annotation.
- Look as your items table and create relevant Item entity object for that table. User @ManyToOne annotation.

Look at hints below :)

```
0 references
@Getter
0 references
@Setter
@OneToMany(mappedBy="cart")
private Set<Item> items;
```

```
@ManyToOne
@JoinColumn(name="cart_id", nullable=false)
private Cart cart;
```

```
0 references
@Getter
0 references
@Setter
@Column(columnDefinition="TEXT")
private String description;
```

# Insert some test data into your tables

```
insert into carts (id) values (default);
```

```
insert into items (name, description, price, cart_id) ·  
· values ("test 1", "test description 1", 8.25, 1);
```

```
insert into items (name, description, price, cart_id) ·  
· values ("test 2", "test description 2", 9.36, 1);|
```

# Write cart and item repositories

- Write CartRepository by extending CrudRepository
- Write ItemRepository by extending CrudRepository

# Write CartRestController

- Write CartRestController that will have cartRepository and return all carts with items by using GetMapping('/api/carts');  
<http://localhost:8080/api/carts> should return
- Write CartRestController that will have cartRepository and return cart by id with items by using GetMapping('/api/carts/{id}');  
<http://localhost:8080/api/carts/1> should return

```
localhost:8080/api/carts
[
  {
    id: 1,
    items: [
      {
        id: 2,
        name: "test 2",
        description: "test description 2",
        price: 9.36
      },
      {
        id: 1,
        name: "test 1",
        description: "test description 1",
        price: 8.25
      }
    ]
  }
]
```

```
localhost:8080/api/carts/1
{
  id: 1,
  items: [
    {
      id: 1,
      name: "test 1",
      description: "test description 1",
      price: 8.25
    },
    {
      id: 2,
      name: "test 2",
      description: "test description 2",
      price: 9.36
    }
  ]
}
```


# Write ItemRestController

- Write ItemRestController that will have itemRepository and return all carts with items by using GetMapping('/api/items');  
<http://localhost:8080/api/items> should return
- Write ItemRestController that will have itemRepository and return cart by id with items by using GetMapping('/api/items/{id}');  
<http://localhost:8080/api/items/1> should return



```
localhost:8080/api/items

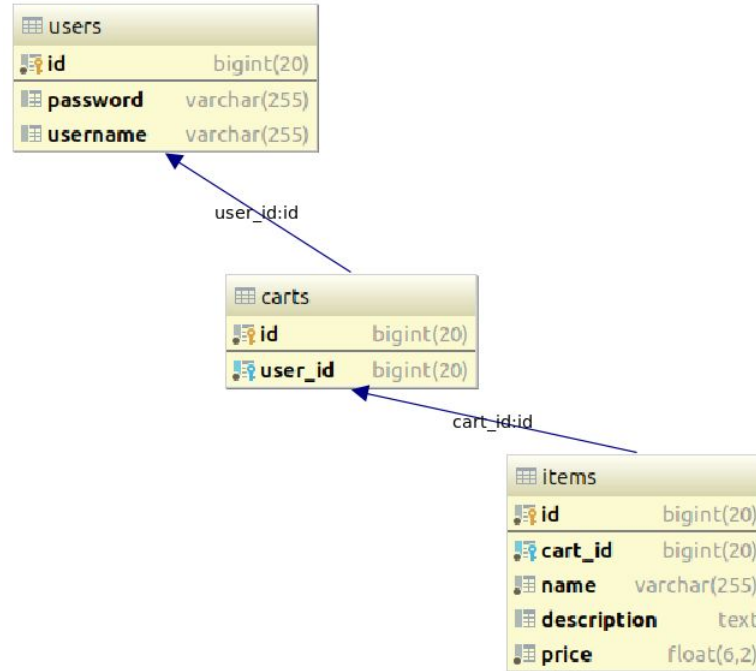
[
  {
    id: 1,
    name: "test 1",
    description: "test description 1",
    price: 8.25
  },
  {
    id: 2,
    name: "test 2",
    description: "test description 2",
    price: 9.36
  }
]
```



```
localhost:8080/api/items/2

{
  id: 2,
  name: "test 2",
  description: "test description 2",
  price: 9.36
}
```

# Create one to many relation between users and carts tables in database





How migration can look with flyway :)

# Create oneToMany relation between users and carts models

- Look at you users table and create relevant relation by using @OneToMany annotation.
- Look as your carts table and create relevant relation by using @ManyToOne annotation.

Look at hints below :)

```
· @ManyToOne  
· @JoinColumn(name="user_id", nullable=false)  
· private User user;
```

```
· @OneToMany(mappedBy="user")  
· private Set<Cart> cart;
```

# Modify spring security config

Modify spring security class in such way that:

/api/carts

/api/users

/api/items

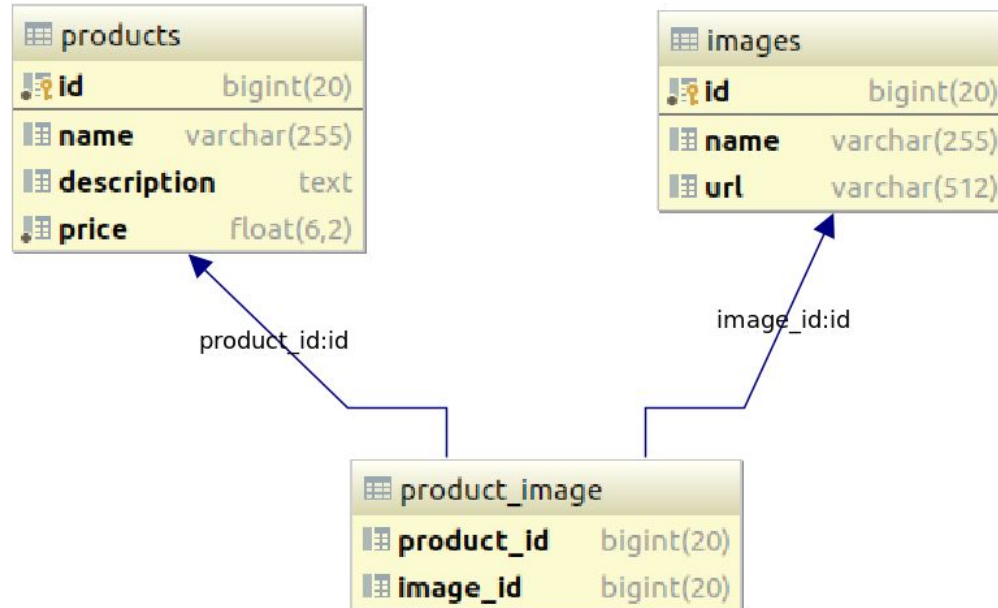
could be accessed by user with `USER` role / authority

# Debugging

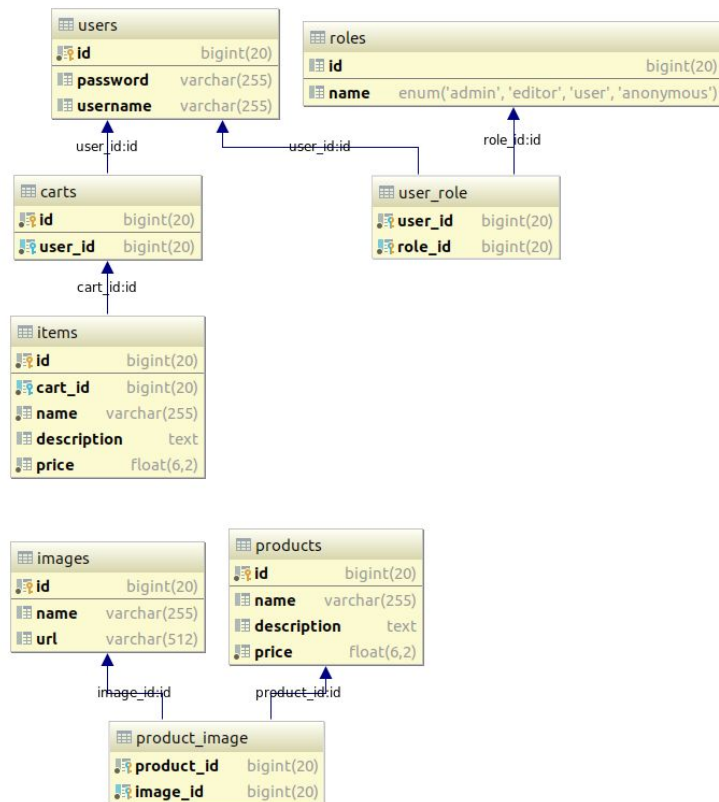
**Debugging** is the process of finding and resolving defects or problems within a computer program that prevent correct operation of [computer software](#) or a [system](#).

Debugging tactics can involve [interactive](#) debugging, [control flow](#) analysis, [unit testing](#), [integration testing](#), [log file analysis](#), monitoring at the [application](#) or [system](#) level, [memory dumps](#), and [profiling](#).

Create products and images tables, models and repositories. Create many to many relations between them. Use schema below.



# More or less final database with relations



# How to debug spring boot application?

1. Add configuration for spring boot plugin (maven example below):

```
<configuration>  
  <jvmArguments>  
    -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=y,address=5005  
  </jvmArguments>  
</configuration>
```

2. Run application: mvn spring-boot:run
3. Press F5 (in visual studio code)

# Spring boot debug example

**VARIABLES**

- Local
  - this: UsersRestController (i...
  - userRepository: \$Proxy109 (...)
    - constructorParams: Class[1...
    - key0: Object (id=100)
    - m0: Method (id=101)
    - m1: Method (id=102)
    - m10: Method (id=103)
    - m11: Method (id=104)
    - m12: Method (id=105)
    - m13: Method (id=106)
    - m14: Method (id=107)

**WATCH**

**CALL STACK**

- Thread [http-nio-8080-exec-8] RUNN...
- Thread [http-nio-8080-exec-7] RUNN...
- Thread [http-nio-8080-exec-6] RUNN...
- Thread [http-nio-8080-exec-5] RUNN...
- Thread [http-nio-8080-exec-4] RUNN...
- Thread [http-nio-8080-exec-3] RUNN...
- Thread [http-nio-8080-exec-2] RUNN...
- Thread [http-nio-8080-exec-1] PAUS...
  - UsersRestController.getUsers()

**BREAKPOINTS**

- ☐ Uncaught Exceptions
- ☐ Caught Exceptions
- ☒ UsersRestController.java src/... 22

```
10 import lombok.val;
11 import lt.baltictalents.lessons910.model.User;
12 import lt.baltictalents.lessons910.repository.auth;
13
14 @RestController
15 @RequestMapping("/api")
16 public class UsersRestController {
17     @Autowired
18     UserRepository userRepository;
19
20     @GetMapping("/users")
21     public List<User> getUsers() {
22         val users = userRepository.findAll();
23         return users;
24     }
25 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

Spring Boot :: (v2.1.2.RELEASE)