

One-to-Many Unidirectional

- One entity associated with multiple records in another entity like: User can have many Orders.
- Reference exist only in 1 direction i.e. from Parent to Child.
- Since its 1:Many, means 1 parent have multiple child and we can not store multiple child ids in 1 parent row, so it creates a **NEW TABLE** and stores the mapping.
- By default its **Lazy** loading, means when query parents, child rows are not fetched.

```
@Table(name = "user_details")
@Entity
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String phone;

    @OneToMany(cascade = CascadeType.ALL)
    private List<OrderDetails> orderDetails = new ArrayList<>();

    // Constructors
    public UserDetails() {
    }

    //getters and setters
}
```

```
@Table(name = "order_details")
@Entity
public class OrderDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String productName;

    //getters and setters
}
```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM USER_DETAILS
```

SELECT * FROM USER_DETAILS;

 (no rows, 1 ms)

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM ORDER_DETAILS
```

SELECT * FROM ORDER_DETAILS;

 (no rows, 4 ms)

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM USER_DETAILS_ORDER_DETAILS |
```

SELECT * FROM USER_DETAILS_ORDER_DETAILS;

 (no rows, 1 ms)

What if, we don't want to create new table:

- We can use **@JoinColumn**, this also tells JPA that we want to store the FK in Child table instead of creating a new table.

```

@Table(name = "user_details")
@Entity
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;
    private String name;
    private String phone;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id_fk", referencedColumnName = "userId")
    private List<OrderDetails> orderDetails = new ArrayList<>();

    // Constructors
    public UserDetails() {
    }

    //getters and setters
}

```

```

@Table(name = "order_details")
@Entity
public class OrderDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String productName;

    //getters and setters
}

```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM USER_DETAILS
```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM ORDER_DETAILS
```

SELECT * FROM USER_DETAILS;

USER_ID	NAME	PHONE
---------	------	-------

(no rows, 1 ms)

SELECT * FROM ORDER_DETAILS;

ID	USER_ID_FK	PRODUCT_NAME
----	------------	--------------

(no rows, 1 ms)

1. INSERT CALL

```

POST /api/user
localhost:8080/api/user

Params: Authorization, Headers (8), Body (raw, JSON)
Body (raw):
{
    "name": "JohnXYZ",
    "phone": "1234567890",
    "orderDetails": [
        {
            "productName": "IceCream"
        },
        {
            "productName": "ColdDrinks"
        }
    ]
}

```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM USER_DETAILS
```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM ORDER_DETAILS
```

```
SELECT * FROM USER_DETAILS;
```

USER_ID	NAME	PHONE
1	JohnXYZ	1234567890

(1 row, 1 ms)

```
SELECT * FROM ORDER_DETAILS;
```

ID	USER_ID_FK	PRODUCT_NAME
1	1	IceCream
2	1	ColdDrinks

(2 rows, 1 ms)

2. GET CALL (LAZY)

```
@RestController
@RequestMapping(value = "/api/")
public class UserContrller {

    @Autowired
    UserDetailsService userDetailsService;

    @PostMapping(path = "/user")
    public UserDetails insertUser(@RequestBody UserDetails userDetails) {
        return userDetailsService.saveUser(userDetails);
    }

    @GetMapping("/user/{id}")
    public UserDetailsDTO fetchUser(@PathVariable Long id) {
        UserDetails output = userDetailsService.findById(id);
        System.out.println("going to map UserDetails to UserDTO");
        UserDetailsDTO userDTO = output.mapUserDetailsToUserDTO();
        return userDTO;
    }
}
```

```
@Table(name = "user_details")
@Entity
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;
    private String name;
    private String phone;

    @OneToOne(mappedBy = CascadeType.ALL)
    @JoinColumn(name = "user_id_fk", referencedColumnName = "userId")
    private List<OrderDetails> orderDetails = new ArrayList<>();

    public UserDetailsDTO mapUserDetailsToUserDTO() {
        return new UserDetailsDTO(this);
    }

    //getters and setters
}
```

```
public class UserDetailsDTO {

    private Long id;
    private String name;
    private String phone;
    private List<OrderDetails> orders;

    // Constructor to populate from UserDetails entity
    public UserDetailsDTO(UserDetails userDetails) {
        this.id = userDetails.getUserId();
        this.name = userDetails.getName();
        this.phone = userDetails.getPhone();
        System.out.println("going to query order table here now");
        this.orders = userDetails.getOrderDetails();
    }

    //getters and setters
}
```

Output:

GET localhost:8080/api/user/1

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
{"id": 1,
 "name": "JohnXYZ",
 "phone": "1234567890",
 "orders": [
     {
         "id": 1,
         "productName": "IceCream"
     },
     {
         "id": 2,
         "productName": "ColdDrinks"
     }
 ]}
```

```
Hibernate:
select
    ud1_0.user_id,
    ud1_0.name,
    ud1_0.phone
from
    user_details ud1_0
where
    ud1_0.user_id=?
going to map UserDetails to UserDTO
going to query order table here now
Hibernate:
select
    od1_0.user_id_fk,
    od1_0.id,
    od1_0.product_name
from
    order_details od1_0
where
    od1_0.user_id_fk=?
```

EAGER fetch:

```
@RestController
@RequestMapping(value = "/api/")
public class UserController {

    @Autowired
    UserDetailsService userDetailsService;

    @PostMapping(path = "/user")
    public UserDetails insertUser(@RequestBody UserDetails userDetails) {
        return userDetailsService.saveUser(userDetails);
    }

    @GetMapping("/user/{id}")
    public UserDetails fetchUser(@PathVariable Long id) {
        return userDetailsService.findById(id);
    }
}
```

```
@Table(name = "user_details")
@Entity
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;
    private String name;
    private String phone;

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @JoinColumn(name = "user_id_fk", referencedColumnName = "userId")
    private List<OrderDetails> orderDetails = new ArrayList<>();

    // Constructors
    public UserDetails() {
    }

    //getters and setters
}
```

Output:

```
Hibernate:
    select
        ud1_0.user_id,
        ud1_0.name,
        ud1_0.phone,
        od1_0.user_id_fk,
        od1_0.id,
        od1_0.product_name
    from
        user_details ud1_0
    left join
        order_details od1_0
        on ud1_0.user_id=od1_0.user_id_fk
    where
        ud1_0.user_id=?
```

Different Cascade Types:

Cascade Type	Impact
CascadeType.PERSIST	Saving User(Parent) also saves related Orders (Child).
CascadeType.MERGE	Updating User also updates Orders.
CascadeType.REMOVE	Deleting User also deletes Orders.
CascadeType.ALL	Includes all cascade operations.

Orphan Removal:

Automatically removes child entry when child removed from Parent collection.

```
@RestController
@RequestMapping(value = "/api/")
public class UserController {

    @Autowired
    UserDetailsService userDetailsService;

    @PostMapping(path = "/user")
    public UserDetails insertUser(@RequestBody UserDetails userDetails) {
        return userDetailsService.saveUser(userDetails);
    }

    @GetMapping("/user/{id}")
    public UserDetails testOrphan(@PathVariable Long id) {
        UserDetails output = userDetailsService.findById(id);
        output.getOrderDetails().remove(0);
        userDetailsService.saveUser(output);
        return output;
    }
}
```

```
@Table(name = "user_details")
@Entity
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;
    private String name;
    private String phone;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = false)
    @JoinColumn(name = "user_id_fk", referencedColumnName = "userId")
    private List<OrderDetails> orderDetails = new ArrayList<>();

    // Constructors
    public UserDetails() {
    }

    //getters and setters
}
```

1st: INSERT

The screenshot shows a POST request to `localhost:8080/api/user`. The JSON body contains:

```
1 {
2     "name": "JohnXYZ",
3     "phone": "1234567890",
4     "orderDetails": [
5         {
6             "productName": "IceCream"
7         },
8         {
9             "productName": "ColdDrinks"
10        }
11    ]
12 }
```

The response shows the inserted user details:

	USER_ID	NAME	PHONE
1	1	JohnXYZ	1234567890

(1 row, 1 ms)

Below the user details, there are two tables:

- SELECT * FROM USER_DETAILS;**
- SELECT * FROM ORDER_DETAILS;**

The `USER_DETAILS` table has one row:

ID	USER_ID_FK	PRODUCT_NAME
1	1	IceCream

(2 rows, 4 ms)

The `ORDER_DETAILS` table has two rows:

ID	USER_ID_FK	PRODUCT_NAME
1	1	IceCream
2	1	ColdDrinks

(2 rows, 4 ms)

2nd: Testing Orphan Removal (With OrphanRemoval = false)

The screenshot shows a GET request to `localhost:8080/api/user/1`. The response shows the user details:

	USER_ID	NAME	PHONE
1	1	JohnXYZ	1234567890

(1 row, 1 ms)

Below the user details, there are two tables:

- SELECT * FROM USER_DETAILS;**
- SELECT * FROM ORDER_DETAILS;**

The `USER_DETAILS` table has one row:

ID	USER_ID_FK	PRODUCT_NAME
1	1	IceCream

(2 rows, 4 ms)

The `ORDER_DETAILS` table has two rows:

ID	USER_ID_FK	PRODUCT_NAME
1	1	IceCream
2	1	ColdDrinks

(2 rows, 4 ms)

Pretty Raw Preview Visualize JSON ↻

```

1 {
2   "userId": 1,
3   "name": "JohnXYZ",
4   "phone": "1234567890",
5   "orderDetails": [
6     {
7       "id": 2,
8       "productName": "ColdDrinks"
9     }
10  ]
11 }
```

SELECT * FROM USER_DETAILS;

USER_ID	NAME	PHONE
1	JohnXYZ	1234567890

(1 row, 1 ms)

SELECT * FROM ORDER_DETAILS;

ID	USER_ID_FK	PRODUCT_NAME
1	null	IceCream
2	1	ColdDrinks

(2 rows, 0 ms)

Console Output: Persistence Context knows all the info, which is present in memory

```

Hibernate:
    select
        ud1_0.user_id,
        ud1_0.name,
        ud1_0.phone
    from
        user_details ud1_0
    where
        ud1_0.user_id=?
Hibernate:
    select
        od1_0.user_id_fk,
        od1_0.id,
        od1_0.product_name
    from
        order_details od1_0
    where
        od1_0.user_id_fk=?
Hibernate:
    update
        order_details
    set
        user_id_fk=null
    where
        user_id_fk=?
        and id=?
```

```

@Table(name = "user_details")
@Entity
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;
    private String name;
    private String phone;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "user_id_fk", referencedColumnName = "userId")
    private List<OrderDetails> orderDetails = new ArrayList<>();

    // Constructors
    public UserDetails() {
    }

    // getters and setters
}
```

GET localhost:8080/api/user/1

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key
Key

Body Cookies Headers (5) Test Results ⏺

Pretty Raw Preview Visualize JSON

```
1 {  
2   "userId": 1,  
3   "name": "JohnXYZ",  
4   "phone": "1234567890",  
5   "orderDetails": [  
6     {  
7       "id": 2,  
8       "productName": "ColdDrinks"  
9     }  
10  ]  
11 }
```

Run Run Selected Auto complete Clear SQL statement: SELECT * FROM USER_DETAILS

SELECT * FROM USER_DETAILS;

USER_ID	NAME	PHONE
1	JohnXYZ	1234567890

(1 row, 1 ms)

Run Run Selected Auto complete Clear SQL statement: SELECT * FROM ORDER_DETAILS

SELECT * FROM ORDER_DETAILS;

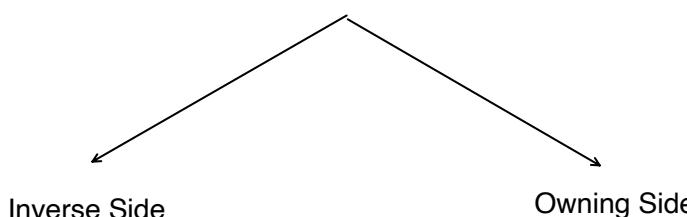
ID	USER_ID_FK	PRODUCT_NAME
2	1	ColdDrinks

(1 row, 1 ms)

```
Hibernate:  
    select  
        ud1_0.user_id,  
        ud1_0.name,  
        ud1_0.phone  
    from  
        user_details ud1_0  
    where  
        ud1_0.user_id=?  
Hibernate:  
    select  
        od1_0.user_id_fk,  
        od1_0.id,  
        od1_0.product_name  
    from  
        order_details od1_0  
    where  
        od1_0.user_id_fk=?  
Hibernate:  
    update  
        order_details  
    set  
        user_id_fk=null  
    where  
        user_id_fk=?  
        and id=?  
Hibernate:  
    delete  
        from  
        order_details  
    where  
        id=?
```

One-to-Many Bidirectional

- Parent reference to child.
- Each Child reference to Parent.



- No Foreign key is created in table.
- Only holds **Object** reference of owing entity.
- Holds the Foreign Key relationship in a table.

```

@Table(name = "user_details")
@Entity
@JsonIdentityInfo(
    generator = ObjectIdGenerators.PropertyGenerator.class,
    property = "userId")
public class UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;
    private String name;
    private String phone;
    @OneToMany(mappedBy = "userDetails", cascade = CascadeType.ALL)
    private List<OrderDetails> orderDetails = new ArrayList<>();

    public Long getUserId() {
        return userId;
    }

    public void setUserId(Long userId) {
        this.userId = userId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public List<OrderDetails> getOrderDetails() {
        return orderDetails;
    }

    public void setOrderDetails(List<OrderDetails> orderDetails) {
        this.orderDetails = orderDetails;
        for(OrderDetails order: orderDetails) {
            order.setUserDetails(this);
        }
    }
}

```

```

@Table(name = "order_details")
@Entity
@JsonIdentityInfo(
    generator = ObjectIdGenerators.PropertyGenerator.class,
    property = "id")
public class OrderDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String productName;
    @ManyToOne
    @JoinColumn(name = "user_id_owing_fk", referencedColumnName = "userId")
    private UserDetails userDetails;

    public Long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getProductName() {
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public UserDetails getUserDetails() {
        return userDetails;
    }

    public void setUserDetails(UserDetails userDetails) {
        this.userDetails = userDetails;
    }
}

```

localhost:8080/api/user

POST localhost:8080/api/user

Params • Authorization Headers (8) Body • Scripts Settings

none form-data x-www-form-urlencoded raw binary

```

1 {
2     "name": "JohnXYZ",
3     "phone": "1234567890",
4     "orderDetails": [
5         {
6             "productName": "IceCream"
7         },
8         {
9             "productName": "ColdDrinks"
10        }
11    ]
12 }

```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2     "userId": 1,
3     "name": "JohnXYZ",
4     "phone": "1234567890",
5     "orderDetails": [
6         {
7             "id": 1,
8             "productName": "IceCream",
9             "userDetails": 1
10        },
11        {
12             "id": 2,
13             "productName": "ColdDrinks",
14             "userDetails": 1
15        }
16    ]
17 }

```

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM USER_DETAILS;

USER_ID	NAME	PHONE
1	JohnXYZ	1234567890

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM ORDER_DETAILS;

ID	USER_ID_OWING_FK	PRODUCT_NAME
1	1	IceCream
2	1	ColdDrinks

(2 rows, 1 ms)

localhost:8080/api/user/1

GET localhost:8080/api/user/1

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key
Key

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```

1 {
2     "userId": 1,

```

```

3   "name": "JohnXYZ",
4   "phone": "1234567890",
5   "orderDetails": [
6     {
7       "id": 1,
8       "productName": "IceCream",
9       "userDetails": 1
10    },
11    {
12      "id": 2,
13      "productName": "ColdDrinks",
14      "userDetails": 1
15    }
16  ]
17 }

```

Many-to-One:Unidirectional

- In this, we generally talk from the perspective of Child like : Many Orders can be placed by 1 User.
- So still, User is considered as Parent and Orders as Child.
- Parent don't have reference to child.
- Each child has reference to parent.

```

@RestController
@RequestMapping(value = "/api/")
public class OrderController {

    @Autowired
    OrderService orderService;

    @GetMapping("/order/{id}")
    public OrderDetails fetchUser(@PathVariable Long id) {
        return orderService.findById(id);
    }

    @PostMapping(path = "/order")
    public OrderDetails insertOrder(@RequestBody OrderDetails orderDetails) {
        return orderService.saveOrder(orderDetails);
    }
}

```

```

@Service
public class OrderService {

    @Autowired
    OrderDetailsRepository orderDetailsRepository;

    public OrderDetails findById(Long primaryKey) {
        return orderDetailsRepository.findById(primaryKey).get();
    }

    public OrderDetails saveOrder(OrderDetails order) {
        return orderDetailsRepository.save(order);
    }
}

```

```

@Table(name = "user_details")
@Entity
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;
    private String name;
    private String phone;

    //getters and setters
}

```

```

@Table(name = "order_details")
@Entity
public class OrderDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String productName;
    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id_owing_fk", referencedColumnName = "userID")
    private UserDetails userDetails;

    //getter and setter
}

```

POST localhost:8080/api/order

Params Authorization Headers (8) Body Scripts Settings

None form-data x-www-form-urlencoded Raw binary GraphQL JSON

```

1 {
2   "productName": "IceCream",
3   "userDetails": [
4     {
5       "name": "SJ",
6       "phone": "111212121221"
7     }
8 ]

```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM USER_DETAILS;

USER_ID	NAME	PHONE
1	SJ	111212121221

(1 row, 1 ms)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM ORDER_DETAILS;

id	productName	userDetails
1	IceCream	{...}

```

5     "userId": 1,
6     "name": "SJ",
7     "phone": "111212121221"
8 }

```

SELECT * FROM ORDER_DETAILS;

ID	USER_ID_OWING_FK	PRODUCT_NAME
1	1	IceCream

(1 row, 1 ms)

Try it out:

Many-to-One: Bidirectional would be same as OneToMany Bidirectional

Many-to-Many: Unidirectional

- Reference from One way only

```

@Table(name = "order_details")
@Entity
public class OrderDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long orderNo;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(
        name = "order_product", // new Join table name
        joinColumns = @JoinColumn(name = "order_id"), // Foreign key for Order
        inverseJoinColumns = @JoinColumn(name = "product_id") // Foreign key for Product
    )
    private List<ProductDetails> productDetails = new ArrayList<>();

    public Long getOrderNo() {
        return orderNo;
    }

    public void setOrderNo(Long orderNo) {
        this.orderNo = orderNo;
    }

    public List<ProductDetails> getProductDetails() {
        return productDetails;
    }

    public void setProductDetails(List<ProductDetails> productDetails) {
        this.productDetails = productDetails;
    }
}

```

```

@Table(name = "product_details")
@Entity
public class ProductDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long productId;

    private String name;
    private double price;

    //getters and setters
}

```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM ORDER_DETAILS;
```

```
SELECT * FROM ORDER_DETAILS;
ORDER_NO
(no rows, 1 ms)
```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM PRODUCT_DETAILS;
```

```
SELECT * FROM PRODUCT_DETAILS;
PRICE PRODUCT_ID NAME
(no rows, 0 ms)
```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM ORDER_PRODUCT
```

```
SELECT * FROM ORDER_PRODUCT;
ORDER_ID PRODUCT_ID
(no rows, 1 ms)
```

```

@RestController
@RequestMapping(value = "/api/")
public class ProductController {

    @Autowired
    ProductService productService;

    @PostMapping(path = "/product")
    public ProductDetails insertUser(@RequestBody ProductDetails product) {
        return productService.saveProduct(product);
    }
}

```

```

@RestController
@RequestMapping(value = "/api/")
public class OrderController {

    @Autowired
    OrderService orderService;

    @Autowired
    ProductService productService;

    @GetMapping("/order/{id}")
    public OrderDetails fetchUser(@PathVariable Long id) {
        return orderService.findById(id);
    }

    @PostMapping(path = "/order")
    public OrderDetails insertOrder(@RequestBody OrderDetails orderDetail) {

        List<ProductDetails> managedProducts = orderDetail.getProductDetails().stream()
            .map(product -> productService.findById(product.getProductId()))
            .collect(Collectors.toList());

        orderDetail.setProductDetails(managedProducts);
        return orderService.saveOrder(orderDetail);
    }
}

```

1st : CREATE PRODUCTS

The screenshot shows two separate Postman requests for creating products.

Request 1:

- Method: POST
- URL: localhost:8080/api/product
- Body (raw JSON):

```

1 {
2   "name": "ice-cream",
3   "price": "100"
4 }

```

Request 2:

- Method: POST
- URL: localhost:8080/api/product
- Body (raw JSON):

```

1 {
2   "name": "cold-drink",
3   "price": "150"
4 }

```

The screenshot shows a SQL query interface with two queries run.

Query 1:

```
SELECT * FROM PRODUCT_DETAILS;
```

Result 1:

PRICE	PRODUCT_ID	NAME
100.0	1	ice-cream
150.0	2	cold-drink

(2 rows, 2 ms)

Query 2:

```
SELECT * FROM ORDER_DETAILS;
```

Result 2:

ORDER_NO
(no rows, 1 ms)

2nd : CREATE Multiple Orders

The screenshot shows a Postman request for creating multiple orders.

Request:

- Method: POST
- URL: localhost:8080/api/order

```

Params • Authorization Headers (8) Body • Scripts Se
 none  form-data  x-www-form-urlencoded  raw  b

1 {
2   "productDetails": [
3     {
4       "productId": 1
5     },
6     {
7       "productId": 2
8     }
9   ]
10 }
11 }
12 }

Body Cookies Headers (5) Test Results ⓘ
```

Pretty Raw Preview Visualize JSON

```

1 {
2   "orderNo": 1,
3   "productDetails": [
4     {
5       "productId": 1,
6       "name": "ice-cream",
7       "price": 100.0
8     },
9     {
10       "productId": 2,
11       "name": "cold-drink",
12       "price": 150.0
13     }
14   ]
15 }
```

```

Params • Authorization Headers (8) Body • Scripts Se
 none  form-data  x-www-form-urlencoded  raw  b

1 {
2   "productDetails": [
3     {
4       "productId": 2
5     }
6   ]
7 }
```

Pretty Raw Preview Visualize JSON

```

1 {
2   "orderNo": 2,
3   "productDetails": [
4     {
5       "productId": 2,
6       "name": "cold-drink",
7       "price": 150.0
8     }
9   ]
10 }
```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM ORDER_DETAILS
```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM PRODUCT_DETAILS
```

Run Run Selected Auto complete Clear SQL statement:

```
SELECT * FROM ORDER_PRODUCT
```

ORDER_ID	PRODUCT_ID
1	1
1	2
2	2

(3 rows, 1 ms)

Many-to-Many: Bidirectional

. Since its Many to Many, anyone can be Owning and inverse side.

```

@Table(name = "order_details")
@Entity
public class OrderDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long orderNo;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(
        name = "order_product", // new Join table name
        joinColumns = @JoinColumn(name = "order_id"), // Foreign key for Order
        inverseJoinColumns = @JoinColumn(name = "product_id") // Foreign key for Product
    )
    private List<ProductDetails> productDetails = new ArrayList<>();

    public Long getOrderNo() {
        return orderNo;
    }

    public void setOrderNo(Long orderNo) {
        this.orderNo = orderNo;
    }

    public List<ProductDetails> getProductDetails() {
        return productDetails;
    }

    public void setProductDetails(List<ProductDetails> productDetails) {
        this.productDetails = productDetails;
    }
}
```

```

@Table(name = "product_details")
@Entity
public class ProductDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long productId;

    private String name;
    private double price;

    @ManyToMany(mappedBy = "productDetails")
    @JsonIgnore
    List<OrderDetails> orders = new ArrayList<>();

    //GETTERS AND SETTERS
}
```

