API 개발 고급 - 지연 로딩과 조회 성능 최적화

1. 간단한 주문 조회 V1: 엔티티를 직접 노출
2. 간단한 주문 조회 V2: 엔티티를 DTO로 변환
3. 간단한 주문 조회 V3: 엔티티를 DTO로 변환
   - 페치 조인 최적화
4. 간단한 주문 조회 V4: JPA에서 DTO로 바로 조회

간단한 주문 조회 V1: 엔티티를 직접 노출

order
order -> Member ManyToOne
Order -> Delivery OneToOne

=> xToOne

```
@GetMapping("/api/v1/simple-orders")
public List<Order> ordersV1() {
    List<Order> all = orderRepository.findAllByString(new OrderSearch());


    return all;
}
```

간단한 주문 조회 V2: 엔티티를 DTO로 변환

쿼리가 총 1 + N + N번 실행된다.
order 조회 1번(order 조회 결과 수가 N이 된다.)
order -> member 지연 로딩 조회 N 번
order -> delivery 지연 로딩 조회 N 번

```java
/**
 * V2. 엔티티를 조회해서 DTO로 변환(fetch join 사용X)
 * - 단점: 지연로딩으로 쿼리 N번 호출
 */
@GetMapping("/api/v2/simple-orders")
public List<SimpleOrderDto> ordersV2() {
    List<Order> orders = orderRepository.findAll();
    List<SimpleOrderDto> result = orders.stream()
            .map(o -> new SimpleOrderDto(o))
            .collect(toList());

    return result;
}
```

```java
@Data
static class SimpleOrderDto {

    private Long orderId;
    private String name;
    private LocalDateTime orderDate; //주문시간
    private OrderStatus orderStatus;
    private Address address;

    public SimpleOrderDto(Order order) {
        orderId = order.getId();
        name = order.getMember().getName();
        orderDate = order.getOrderDate();
        orderStatus = order.getStatus();
        address = order.getDelivery().getAddress();
    }
}
```

간단한 주문 조회 V3: 엔티티를 DTO로 변환
- 페치 조인 최적화

엔티티를 페치 조인(fetch join)을 사용해서
쿼리 1번에 조회
페치 조인으로 order -> member , order -> delivery 는
이미 조회 된 상태 이므로 지연로딩X

```
/**
 * V3. 엔티티를 조회해서 DTO로 변환(fetch join 사용O)
 * - fetch join으로 쿼리 1번 호출
 * 참고: fetch join에 대한 자세한 내용은 JPA 기본편 참고(정말 중요함)
 */
@GetMapping("/api/v3/simple-orders")
public List<SimpleOrderDto> ordersV3() {
    List<Order> orders = orderRepository.findAllWithMemberDelivery();
    List<SimpleOrderDto> result = orders.stream()
            .map(o -> new SimpleOrderDto(o))
            .collect(toList());
    return result;
}
```

간단한 주문 조회 V4: JPA에서 DTO로 바로 조회

```
select
    order0_.order_id as col_0_0_,
    member1_.name as col_1_0_,
    order0_.order_date as col_2_0_,
    order0_.status as col_3_0_,
    delivery2_.city as col_4_0_,
    delivery2_.street as col_4_1_,
    delivery2_.zipcode as col_4_2_
from
    orders order0_
```

```
private final OrderSimpleQueryRepository orderSimpleQuery
주입


/**
 * V4. JPA에서 DTO로 바로 조회
 * - 쿼리 1번 호출
 * - select 절에서 원하는 데이터만 선택해서 조회
 */
@GetMapping("/api/v4/simple-orders")
public List<OrderSimpleQueryDto> ordersV4() {
    return orderSimpleQueryRepository.findOrderDtos();
}
```

```
@Repository
@RequiredArgsConstructor
public class OrderSimpleQueryRepository {

    private final EntityManager em;

    public List<OrderSimpleQueryDto> findOrderDtos() {
        return em.createQuery(
                "select new
jpabook.jpashop.repository.order.simplequery.OrderSimpleQueryDto(o.id, m.name,
o.orderDate, o.status, d.address)" +
                        " from Order o" +
                        " join o.member m" +
                        " join o.delivery d", OrderSimpleQueryDto.class)
                .getResultList();
```

```
@Data
public class OrderSimpleQueryDto {

    private Long orderId;
    private String name;
    private LocalDateTime orderDate; //주문시간
    private OrderStatus orderStatus;
    private Address address;

    public OrderSimpleQueryDto(Long orderId, String name, LocalDateTime
orderDate, OrderStatus orderStatus, Address address) {
        this.orderId = orderId;
        this.name = name;
        this.orderDate = orderDate;
        this.orderStatus = orderStatus;
        this.address = address;
    }
```

SELECT 절에서 원하는 데이터를 직접 선택하므로
DB 애플리케이션 네트워크 용량 최적화
리포지토리 재사용성 떨어짐, API 스펙에 맞춘 코드가
리포지토리에 들어가는 단점

쿼리 방식 선택 권장 순서

1. 우선 엔티티를 DTO로 변환하는 방법을 선택한다.
2. 필요하면 페치 조인으로 성능을 최적화 한다. 대부분의 성능 이슈가 해결된다.
3. 그래도 안되면 DTO로 직접 조회하는 방법을 사용한다.
4. 최후의 방법은 JPA가 제공하는 네이티브 SQL이나 스프링 JDBC Template을 사용해서 SQL을 직접 사용한다.