MongoDB 기반 이커머스 시스템의 MySQL 전환 및 성능 개선 보고서

1. 프로젝트 개요

기존 프로젝트는 MongoDB 기반의 단순 CRUD 수준의 이커머스 API로 처음에는 트랜잭션을 적용해보고 싶다는 기술적 호기심에 시작하였다. 실제로 구현해보며 다음과 같은 문제가 명확하게 드러나 MySQL 기반 RDBMS로 전환할 필요가 생겼다.

- 트랜잭션이 필요한 도메인인데 MongoDB는 일관성 유지가 어려웠다.
- 주문 스키마 내부에 Delivery, Customer가 객체로, Products는 ObjectId의 배열 형태 등 중첩된 문서로 존재하였다.
- 재고 차감과 같은 핵심 로직을 구현하기 위해서는 별도의 Collection을 업데이트해야 하는데 MongoDB의 multi-document transaction 비용이 커 부하 상황에서는 안정적이지 않았다 이에 따라 재고 차감 로직을 구현하지 못하였고 이는 정합성을 요구하는 전형적인 이커머스 로직이 불가능한 상태임을 보여준다.
- 주문 상품이 ID 배열만으로 구성되어 상품이 삭제되어도 주문의 상품 내역은 남아 있게 된다. 이는 데이터의 일관성이 깨질 가능성을 시사한다. 또한 주문 상세 조회 시 상품을 매번 N번의 별도 조회가 필요하여 부하 시 병목 현상을 증가시킨다.
- Delivery와 Customer가 주문 안에 중첩 객체로 존재한다. 이로 인해 특정 Customer의 주문 검색, 배송지 패턴 분석, Delivery와 Customer 변경 이력 관리가 불가능하고 한 사람이 여러 주문을 할 때 중복된 데이터가 폭증하는 문제점이 있다.

2-1. 마이그레이션 범위: 스키마 설계

MongoDB에서는 하나의 문서에 여러 데이터들이 중첩된 형태로 저장되어 있어 정합성 유지와 쿼리 확장성이 떨어지는 문제가 있었다. 이를 해결하기 위해 RDBMS 특성에 맞게 제 3 정규화를 통해 다음과 같이 분리하였다.

- User

  기존 MongoDB에서는 Address가 User 개게 안에 중첩되어 있어 주소 변경 시 이력 관리 불가, 여러 배송지 관리 불가, 조회 및 필터링의 어려움 문제가 있어 사용자 인증과 계정 정보를 가진 User 엔티티와 주소 데이터를 가진 Address 엔티티로 분리하였다

- Product

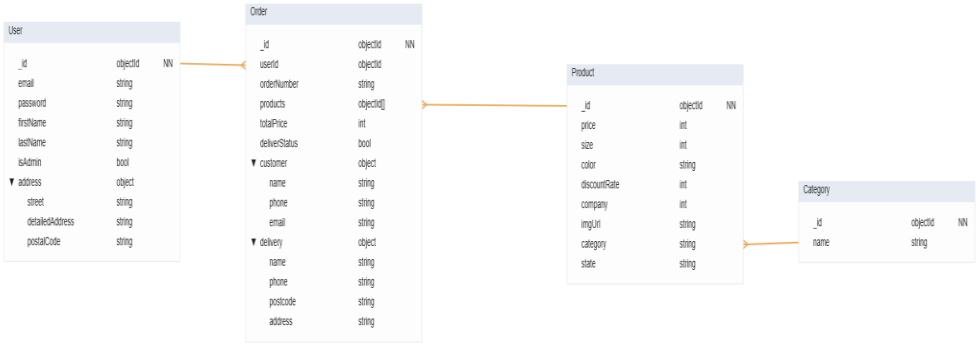
  MongoDB에서는 옵션을 Product 문서 내부 배열로 관리했기 때문에 옵션별 재고 관리가 불안정하고 주문과 재고 차감 로직에서 트랜잭션 적용이 어려웠다. 이를 상품 기본 정보를 가진 Product 엔티티와 size, color 조합별 옵션과 재고를 각각 독립된 필드를 가진 ProductOption 엔티티로 분리하였다.

- Order

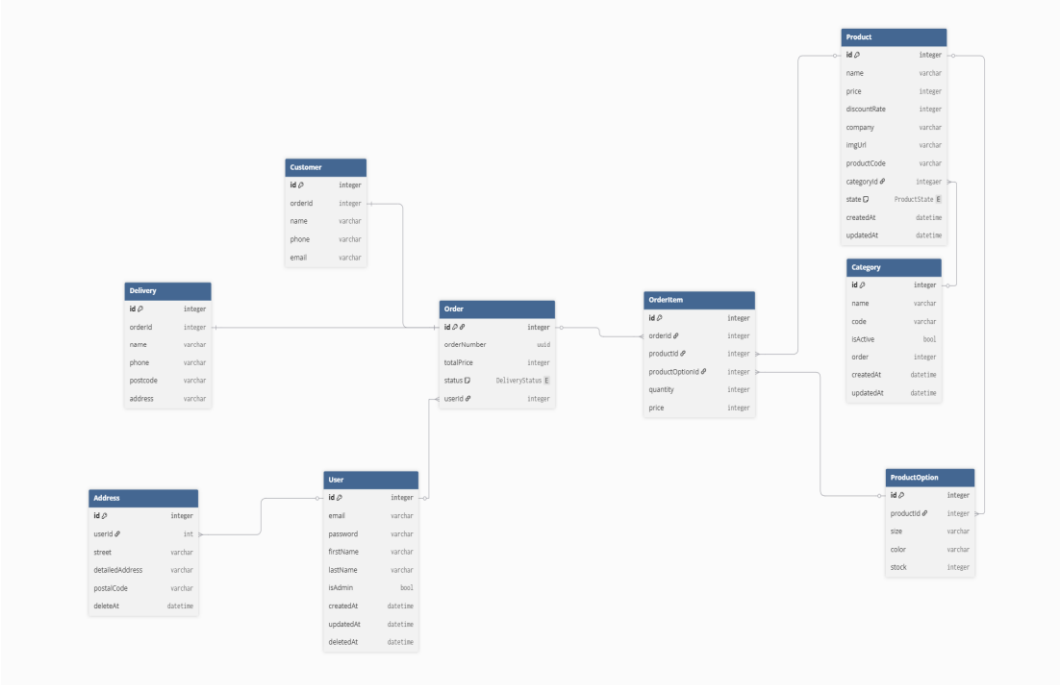
  MongoDB에서는 하나의 주문 문서에 Customer, Delivery, Products 배열 등이 모두 중첩된 형태로 저장되어 있어 정합성 유지와 쿼리 확장성이 떨어지는 문제가 있었다. 이를 해결하기 위해 주문 메타 정보를 가진 Order 엔티티, 주문 내 상품 목록 정보를 가진

OrderItem 엔티티, 주문 시점의 고객 정보를 가진 Customer 엔티티, 배송지 정보를 관리하는 Delivery 엔티티로 분리하였다.

[MongoDB ERD Diagram]



[SQL ERD Diagram]



2-2. 마이그레이션 범위: 트랜잭션 설계

기존 MongoDB 기반 구조에서는 주문, 상품, 회원 정보가 중첩된 문서 형태로 저장되어 있어 재고 차감, 주문 변경, 카테고리 병합, 상품 옵션 수정 등 중요한 비즈니스 로직을 원자적으로 처리할 수 없었다.

Prisma와 MySQL 기반으로 전환한 후에는 다음 원칙에 따라 트랜잭션을 설계했다

- 하나의 사용자 액션을 하나의 트랜잭션으로 처리한다.
- 도메인 데이터의 정합성을 가장 우선으로 한다.
- 재고 차감처럼 동시성 이슈가 생길 수 있는 로직은 마지막 UPDATE 조건까지 DB에게 위임한다.
- 도메인간 의존성은 반드시 동일한 트랜잭션에서 처리한다.
- 롤백 정책을 명확히 정의하여 부분 실패를 허용하지 않는다.

이 원칙을 기반으로 User, Product, Order, Category 각각에 대한 트랜잭션 단위를 정의하고 각 시나리오별로 명확한 롤백 기준과 DAO 책임을 분리하였다.

- User 도메인: 회원가입, 수정, 삭제를 원자적으로 처리한다.
- Product 도메인: 상품 생성, 옵션 전체 교체, 옵션 및 재고 변경, 상품 삭제 시 옵션과 함께 삭제를 트랜잭션으로 묶어 처리한다.
- Order 도메인: 주문 생성, 주문 수정, 삭제를 원자적으로 처리한다.
- Category 도메인: 카테고리 삭제 시 상품 상태 변경, 카테고리 비활성화, 순서 재정렬, 병합 등 다중 엔티티 수정 작업을 트랜잭션으로 보장한다.
- Admin 도메인: 배송 상태 변경, 관리자 권한 수정, 삭제 작업을 트랜잭션으로 보장한다.

3. 성능 최적화 및 인덱스 튜닝

조회 쿼리에서 Full Scan이 발생하여 성능 저하가 확인되었고 이를 해결하기 위해 인덱스를 적용하였다. 비교 기준으로는 상품 상세 조회 API를 사용하였으며 주문 상세 조회와 카테고리별 상품 조회 API에서 각각 인덱스를 추가하여 응답 시간 개선 효과를 측정하였다.

API	Before	After	결과	개선율
상품 상세 조회	10.0ms	11.8ms	+1.8ms 증가	-
주문 상세 조회	16.8ms	14.6ms	-2.2ms 개선	약 13%
카테고리 상품 조회	13.6ms	9.6ms	-4.0ms 개선	약 29%

인덱스 적용 결과 조회 쿼리 구조에 따라 성능 개선 폭이 다르게 나타났다. 단순 PK 조회인 상품 상세 API는 큰 변화가 없었지만 다량의 주문 항목을 조회하거나 카테고리 조건으로 필터링하는 API에서는 각각 약 13%, 29%의 응답 속도 개선이 확인되었다. 이는 인덱스가 조건 기반 검색과 조인 비용 감소에 효과적임을 보여준다.

4. 테스트 환경 구축 및 자동화

Jest와 Supertest 기반으로 단위테스트와 통합 테스트 자동화를 구축하였다. 실제 Pirsma Client를 가져오지 않고 테스트 전용 SQLite Prisma Client를 분리하여 실제 데이터베이스에서의 동작을 검증하였다.

구분	Statements	Branch	Functions	Lines
전체 프로젝트	91.75%	79.34%	92.9%	91.82%
핵심 계층 평균 (Controller, Service, DAO)	약 91%	약 83%	약 95%	약 91%

정상과 예외 처리를 포함하여 총 98개의 테스트 케이스를 작성하였다. 전체 코드 기준 Statements 91.75%, Functions 92.9% 수준의 테스트 커버리지를 확보하였다. 특히 Service – DAO 계층은 평균 95% 이상 높은 커버리지를 유지하며 주요 비즈니스 로직이 테스트 자동화로 안정적으로 검증됨을 확인하였다.

5. 부하 테스트 분석

MySQL 기반 서버가 동시 요청 처리 안정성을 검증하고 기존 MongoDB 기반 서버와의 성능을 비교하기 위해 부하 테스트를 수행하였다. 또한 주문 및 조회 API의 병목 구간을 분석하기 위해 Artillery를 활용하여 전체 상품 조회, 로그인 후 주문 조회, 주문 생성의 세 가지 시나리오로 테스트를 진행하였다.

- 전체 상품 조회

	평균 응답 속도	P95	P99
NoSQL	61.0ms	76ms	92.ms
SQL	5.8ms	8.9ms	10.1ms
	90.5%	88.3%	89.0%

동일 부하 환경에서 SQL 기반 구조는 NoSQL 대비 평균 약 90%, P95 / P99 기준 약 89% 빠른 응답 속도를 보여주었다. 특히 tail latency가 크게 줄어들어 고부하 상황에서도 안정적인 조회 성능을 확보하였다.

- 로그인 및 사용자 주문 조회

	평균 응답 속도	P95	P99	성공률
NoSQL 과부하	2552ms	8520ms	8520ms	8%
NoSQL 부하조정	2737.6ms	8692.8ms	9607.1ms	98%
SQL	39ms	80.6ms	100.5ms	100%

고부하 조건에서는 NoSQL 구조가 9%의 성공률만 기록하여 정상적인 비교가 불가능했다. 부하를 1초당 3명 수준으로 낮춘 경우에도 평균 응답 2.7초, P95 8.7초로 여전히 높은 지연이 발생했다. SQL 기반 서버는 동일한 고부화 환경에서도 평균 39ms, 성공률 100%를 기록하여 확장성과 안정성이 크게 향상됨을 확인하였다.

- 주문 생성

	평균 응답 속도	중앙값	P95	P99	성공률
NoSQL 과부하	2911ms	58.6ms	-	-	24%
NoSQL 부하조정	2888ms	7ms	8868.4ms	9801.2ms	86.6%
SQL	49ms		96.6ms	108.9ms	

NoSQL 기반 서버는 주문 생성 API에서 부하가 누적될 경우 응답 시간이 급격하게 증가하였다. 실제로 서버가 한가할 때는 median 기준 7~50ms 수준으로 정상 처리되었지만 요청이 누적되는 순간 평균 응답이 3초 이상으로 증가하며 전체 요청 처리 지연이 심화되었다. P95/P99 구간에서는 8~9초까지 tail latency가 폭발하며 일부 요청이 timeout 직전까지 대기하는 현상이 나타났다. SQL 기반 서버는 동일 부하에서 평균 49ms, P95/P99 기준 96~108ms로 일관된 응답 시간을 유지하여 고부하 상황에서도 안정적인 처리 성능과 높은 성공률(100%)을 보였다.

6. 트랜잭션 검증

재고에 따른 주문 생성 부하 테스트

	요청	평균 응답 속도	P95	P99
재고 부족(초기 50개)	201 상태: 150개 401 상태: 50개	49ms	92.8ms	127.8ms
재고 1000개	201상태: 200개	49ms	96.9ms	108.9ms

재고 기반 트랜잭션의 정합성을 검증하기 위해 재고가 부족한 상황(50개)과 충분한 상황(1000개)에 대해 주문 생성 부하 테스트를 수행하였다. 재고가 부족한 경우 요청 200개 중 50건이 401(재고 부족)으로 처리되었고 재고가 충분한 경우에는 모든 요청이 201(정상 처리)로 응답하였다. 처리된 주문 수만큼 재고가 정확히 감소했으며 생성된 주문과 주문 상품을 Left Join 하여 확인한 결과 고아 데이터가 존재하지 않아 트랜잭션 정합성이 보장됨을 확인하였다. 모든 시나리오에서 평균 응답속도는 49ms 내외, P95 / P99도 안정적으로 유지되며 동시성 환경에서도 일관된 성능을 보였다.

7. 종합 결론 및 기술적 성과

MongoDB 기반의 이커머스 시스템을 MySQL 기반으로 성공적으로 마이그레이션 하였다. 이 과정에서 데이터베이스 정규화, 트랜잭션, 제약 조건 도입을 통해 데이터 정합성과 품질을 개선하였다. 또한 인덱스 튜닝과 조인 최적화, 쿼리 리팩토링으로 조회 성능 및 응답의 안정성도 대폭 향상시켰으며 동시성 제어로 재고 감소 로직에서 무결성을 확보하였다.

기존 시스템에 없던 재고 처리 로직, 카테고리 재정렬 및 병합 기능, 참조 무결성을 고려한 안

전한 삭제 로직 등 핵심 비즈니스 도메인을 새롭게 설계하여 기능 범위와 데이터 일관성을 확장하였다. 단위 테스트와 통합 테스트 기반의 자동화 환경을 구축하여 유지 보수성과 회귀 안정성을 확보하였고 부하 테스트로 실제 운영 수준의 확장성과 안정성을 검증하였다.