문제 1. A 고객사 시스템의 채널 확대 및 사용자 증가에 따라 발생하는 인증 및 세션 관리 문제 해결을 위한 인증 방식 개선 방안

인증 방식 개선 방안

분류	사용되는 경우	설명	장점	단점
멀티 팩터 인증(MFA)	높은 보안이 요구되는 경우	단순한 ID/PW 인증 대신, 추가적인 인증 요소(예: SMS 코드, 이메일 인증, 생체인식)를 도입	보안 수준이 크게 향상되어 비인가 접근을 방지할 수 있음	사용자 경험이 다소 불편해질 수 있음
SSO	여러 채널과 시스템에 접근할 때	하나의 인증으로 여러 시스템에 접근할 수 있도록 하는 방식	사용자 경험을 개선하고, 비밀번호 관리의 복잡성을 줄임	초기 구현 및 설정에 많은 시간과 비용이 소요될 수 있음
OAuth 2.0 및 OpenID Connect	외부 인증 서버와의 연동이 필요한 경우	표준화된 프로토콜을 사용하여 사용자 인증 및 권한 부여를 관리	보안이 강화되고, 타 시스템과의 연동이 용이함	구현 복잡도가 높고, 외부 서비스 의존성이 발생할 수 있음
분산 세션 스토리지	서버 간 세션 동기화가 필요한 경우	Redis, Memcached 와 같은 분산 세션 스토리지를 활용하여 세션 데이터를 관리	세션 데이터를 중앙에서 관리하여 서버 간 세션 동기화 문제를 해결하고, 확장성이 높아짐	분산 환경에서의 설정 및 관리가 복잡할 수 있음
JWT	서버 부하를 줄이고 확장성을 높여야 하는 경우	클라이언트가 JWT 토큰을 통해 인증 상태를 유지	서버 부하를 줄이고, 확장성이 향상됨	토큰 탈취 시 보안 위협이 될 수 있으며, 토큰 만료 및 갱신 관리가 필요함

‘

A 고객사 비즈니스 유연성과 성능 관점에서 상품을 관리하기 위한 데이터 모델을 새롭게 설계하고 설계 사유 제시

A 고객사의 비즈니스 유연성과 성능 관점에서 상품을 관리하기 위한 데이터 모델을 새롭게 설계하기 위해서는 데이터 모델링 방법론 중에서 정규화(Normalization), 비정규화(Denormalization), 스타 스키마(Star Schema), 스노우플레이크 스키마(Snowflake Schema), EAV(Entity-Attribute-Value) 모델, 도메인 주도 설계(DDD) 중 가장 적절한 것을 선택해야 합니다.

분류	사용되는 경우	설명	장점	단점
정규화(Normalization)	데이터 중복을 최소화하고 무결성을 유지할 때	데이터의 중복을 제거하고 각 데이터 요소를 논리적으로 분리하는 방법론	데이터 무결성 유지, 데이터 중복 최소화, 데이터베이스의 저장 공간 효율성 향상	JOIN 연산이 많아져 쿼리 성능이 저하될 수 있음
비정규화(Denormalization)	쿼리 성능을 최적화하고 읽기 속도를 높일 때	정규화된 데이터를 다시 통합하여 중복을 허용하는 방법론	쿼리 성능 향상, 읽기 속도 증가, 복잡한 JOIN 연산 최소화	데이터 중복으로 인해 저장 공간 증가, 데이터 무결성 유지 어려움
스타 스키마(Star Schema)	데이터 웨어하우징과 분석을 위해 설계할 때	중심에 사실 테이블을 두고 주변에 차원 테이블을 배치하는 모델	쿼리 성능 최적화, 분석 및 보고서 작성에 용이, 단순하고 직관적인 구조	데이터 중복 가능성, 사실 테이블에 많은 데이터가 집중될 경우 성능 저하 가능

스노우플레이크 스키마(Snowflake Schema)	복잡한 데이터 분석과 다차원적 데이터를 처리할 때	스타 스키마를 확장하여 차원 테이블을 더 세분화한 모델	데이터 중복 최소화, 정규화로 인한 저장 공간 효율성 향상	복잡한 구조로 인해 쿼리 성능 저하 가능, 데이터베이스 설계 및 유지보수 복잡성 증가
EAV(Entity-Attribute-Value) 모델	유연한 속성 관리를 필요로 할 때	속성의 종류와 개수가 동적으로 변하는 경우 사용하는 모델	매우 유연한 데이터 구조, 속성 추가 및 변경이 용이	쿼리가 복잡하고 비효율적일 수 있음, 데이터 무결성 및 성능 문제 발생 가능
도메인 주도 설계(DDD, Domain-Driven Design)	복잡한 비즈니스 로직을 캡슐화할 때	도메인 모델을 중심으로 비즈니스 로직을 설계하는 방법론	복잡한 비즈니스 요구사항을 잘 반영, 유지보수 용이, 높은 재사용성	초기 설계가 복잡하고 비용이 많이 들 수 있음, 도메인 전문가와의 긴밀한 협업 필요

-

  설계 사유

A 고객사의 비즈니스 유연성과 성능을 고려한 데이터 모델을 설계할 때 가장 적절한 방법론은 **정규화(Normalization)**와 **비정규화(Denormalization)**의 혼합 사용입니다.

1. **정규화(Normalization)**를 통해 데이터 중복을 최소화하고 데이터 무결성을 유지합니다. 이는 비즈니스 유연성을 확보하는 데 중요한 요소입니다. 데이터 무결성은 비즈니스 규칙을 강제하고 데이터의 신뢰성을 보장합니다.
2. **비정규화(Denormalization)**를 통해 특정 쿼리 성능을 최적화합니다. 자주 사용하는 쿼리의 성능을 높이기 위해 일부 데이터를 중복 저장하고 JOIN 연산을 최소화함으로써 성능을 향상시킵니다. 이는 특히 실시간 성능이 중요한 경우에 유리합니다.

설계 예시

상품(Product) 테이블 (정규화)

- ProductID
- ProductName
- ProductDescription
- Price
- StockQuantity
- CategoryID

카테고리(Category) 테이블 (정규화)

- CategoryID
- CategoryName
- ParentCategoryID

가격변동(PriceHistory) 테이블 (정규화)

- PriceChangeID
- ProductID
- ChangeDate
- PreviousPrice
- NewPrice

재고변동(StockHistory) 테이블 (정규화)

- StockChangeID
- ProductID
- ChangeDate
- PreviousStock
- NewStock

고객리뷰(CustomerReview) 테이블 (정규화)

- ReviewID
- ProductID
- CustomerID
- Rating
- ReviewContent
- ReviewDate

상품 요약(ProductSummary) 테이블 (비정규화)

- ProductID
- ProductName
- CurrentPrice
- CurrentStock
- AverageRating

이러한 데이터 모델은 데이터 무결성과 비즈니스 유연성을 유지하면서도 성능을 최적화할 수 있습니다. 필요에 따라 일부 데이터를 비정규화하여 쿼리 성능을 향상시킴

다음은 각 테이블의 설계 사유를 정규화/비정규화를 포함하여 표로 정리한 것입니다:

테이블명	설계 사유	정규화/비정규화
상품(Product)	- 상품에 대한 기본 정보를 중앙에서 관리하기 위해 설계 - 정규화를 통해 데이터 중복을 최소화하고 무결성을 유지 - 효율적인 검색과 업데이트를 위해 필수적인 속성들로 구성	정규화
카테고리(Category)	- 상품을 다양한 카테고리로 분류하여 유연한 분류 구조를 지원하기 위해 설계 - 정규화를 통해 카테고리 변경 시 상품 데이터와의 연관성을 유지 - 부모카테고리 ID를 포함하여 계층적 구조를 반영	정규화
가격변동(PriceHistory)	- 각 상품의 가격 변동 이력을 기록하여 추적하기 위해 설계 - 정규화를 통해 과거 가격 데이터를 별도로 저장하여 가격 분석 및 변동 내역 확인 용이 - 가격 변동에 따른 비즈니스 의사 결정을 지원하고 데이터 일관성 유지	정규화
재고변동(StockHistory)	- 각 상품의 재고 변동 이력을 기록하여 추적하기 위해 설계 - 정규화를 통해 과거 재고 데이터를 별도로 저장하여 재고 관리 효율성 증대 - 재고 변동에 따른 비즈니스 의사 결정을 지원하고 데이터 일관성 유지	정규화
고객리뷰(CustomerReview)	- 각 상품에 대한 고객 리뷰와 평점을 기록하여 추적하기 위해 설계 - 정규화를 통해 고객의 피드백을 수집하고 분석하여 상품 개선 방향 도출 - 리뷰 데이터를 별도로 관리하여 주요 상품 정보와의 분리 유지 및 성능 최적화	정규화
상품 요약(ProductSummary)	- 자주 조회되는 상품의 요약 정보를 저장하여 쿼리 성능을 최적화하기 위해 설계 - 비정규화를 통해 현재 가격, 현재 재고, 평균 평점과 같은 정보를 포함	비정규화

	- 데이터 중복을 허용하여 주요 데이터를 신속하게 접근 가능	

- 예약 처리 프로세스와 데이터 모델에서 발생하고 있는 동시성 이슈의 해결 방안

다음은 예약 처리 프로세스와 데이터 모델에서 발생하는 동시성 이슈의 해결 방안을 분류, 사용되는 경우, 설명, 장점, 단점으로 정리한 표입니다.

분류	사용되는 경우	설명	장점	단점
낙관적 잠금(Optimistic Locking)	동시 업데이트가 드물 때	데이터 읽을 때 버전을 기록하고 업데이트 시 버전을 비교하여 충돌을 감지	충돌이 거의 없을 때 성능이 우수, 잠금으로 인한 병목 현상 없음	충돌이 발생하면 롤백 필요, 높은 충돌 발생 시 성능 저하
비관적 잠금(Pessimistic Locking)	동시 업데이트가 빈번할 때	데이터 읽기 시 즉시 잠금 설정, 다른 트랜잭션이 접근하지 못하게 함	데이터 충돌 완전 방지, 일관성 유지에 유리	잠금으로 인한 병목 현상 발생 가능, 동시성 감소
타임스탬프 기반 접근(Timestamp-based Concurrency Control)	다수의 트랜잭션이 있을 때	각 트랜잭션에 타임스탬프를 부여하여 순서를 관리, 트랜잭션 간의 순서 충돌을 방지	높은 일관성 유지, 충돌 시 빠른 검출 가능	타임스탬프 관리의 복잡성 증가, 순서 관리 오버헤드
MVCC(Multi-Version Concurrency Control)	읽기 및 쓰기 작업이 많을 때	데이터의 여러 버전을 유지하여 읽기 작업이 쓰기 작업을 방해하지 않도록 함	동시성 매우 우수, 읽기 작업의 지연 없음	저장 공간의 증가, 버전 관리의 복잡성 증가
분산 잠금(Distributed Locking)	분산 시스템에서 동시성 제어할 때	여러 노드 간의 일관성을 유지하기 위해 분산 락 매니저를 통해 잠금 관리	분산 환경에서 데이터 일관성 유지, 확장성 우수	복잡한 설정과 관리 필요, 네트워크 지연에 따른 성능 저하 가능
애플리케이션 수준 잠금(Application-level Locking)	특정 비즈니스 로직에 잠금 필요할 때	애플리케이션 코드 내에서 동시성을 제어, 특정 리소스에 대한 잠금을 직접 구현	구현의 유연성, 특정 요구사항에 맞춘 잠금 제어 가능	구현 복잡성 증가, 개발자의 실수로 인한 잠금 문제 발생 가능

데드락 방지(Deadlock Prevention)	데드락 발생 가능성이 있을 때	데드락 발생 가능성을 사전에 방지하는 알고리즘을 사용, 예: 타임아웃 설정, 자원 요청 순서 정의	데드락 방지로 시스템 안정성 증가	성능 저하 가능, 특정 상황에서 과도한 자원 낭비 가능
잠금 시간 제한(Lock Timeout)	잠금으로 인한 무한 대기를 방지할 때	잠금 시간 제한을 설정하여 일정 시간 이후 잠금이 해제되도록 함	무한 대기 방지, 시스템의 유연성 증가	잠금 해제로 인한 데이터 일관성 문제 발생 가능

SAGA 와 2 단계 커밋(2PC)은 분산 시스템에서 트랜잭션 관리 및 동시성 제어를 위한 두 가지 주요 방법입니다. 이 둘을 설명하고, 이를 표로 정리해 보겠습니다.

SAGA (Saga)

SAGA 는 분산 트랜잭션 관리 기법으로, 긴 시간에 걸쳐 실행되는 분산 트랜잭션을 여러 개의 작은 트랜잭션으로 나누고, 각 트랜잭션이 성공할 때마다 다음 트랜잭션을 수행하며, 실패 시 보상 트랜잭션을 통해 이전 상태로 되돌리는 방식입니다.

2 단계 커밋 (Two-Phase Commit, 2PC)

2 단계 커밋(2PC)는 분산 트랜잭션 관리 기법으로, 트랜잭션을 참여자들 간에 일관되게 적용하기 위해 두 단계로 커밋을 수행합니다. 첫 번째 단계에서는 모든 참여자들이 준비 상태를 확인하고, 두 번째 단계에서는 모든 참여자가 준비 상태일 경우 커밋을 확정하는 방식입니다.

SAGA 와 2PC 의 비교

분류	사용되는 경우	설명	장점	단점
SAGA	장시간 트랜잭션이나 높은 분산 환경	여러 개의 작은 트랜잭션으로 나누어 수행, 실패 시 보상 트랜잭션을 통해 이전 상태로 되돌림	비동기적 처리 가능, 높은 확장성, 롱 런닝 트랜잭션에 적합	보상 트랜잭션 필요, 복잡한 롤백 로직 구현 필요
2 단계 커밋(2PC)	강한 일관성이 요구되는 경우	트랜잭션을 두 단계로 나누어 준비 상태 확인 후 커밋을 확정	강한 일관성 보장, 모든 참여자가 일관된 상태 유지	높은 지연 시간, 장애 발생 시

				복구 복잡성 증가

SAGA 설명

- **사용되는 경우**: 장시간 트랜잭션이나 높은 분산 환경에서 유용.
- **설명**: 분산 트랜잭션을 여러 개의 작은 트랜잭션으로 나누어 수행하고, 각 트랜잭션이 실패할 경우 보상 트랜잭션을 통해 이전 상태로 되돌리는 방식.
- **장점**: 비동기적 처리 가능, 높은 확장성, 롱 런닝 트랜잭션에 적합.
- **단점**: 보상 트랜잭션이 필요하며, 복잡한 롤백 로직 구현이 필요.

2 단계 커밋(2PC) 설명

- **사용되는 경우**: 강한 일관성이 요구되는 경우에 유용.
- **설명**: 분산 트랜잭션을 두 단계로 나누어 수행. 첫 단계에서 모든 참여자들의 준비 상태를 확인하고, 두 번째 단계에서 준비가 완료되면 커밋을 확정.
- **장점**: 강한 일관성 보장, 모든 참여자가 일관된 상태를 유지.
- **단점**: 높은 지연 시간 발생 가능, 장애 발생 시 복구 복잡성 증가.

이 두 가지 방법론은 분산 시스템에서 트랜잭션 일관성을 유지하고 동시성 이슈를 해결하기 위한 중요한 도구입니다. SAGA 는 비동기 처리와 높은 확장성을 요구하는 환경에 적합하며, 2PC 는 강한 일관성이 필요한 경우에 유용합