

2024 역량진단 2

문제 1]

A고객사 시스템의 채널 확대 및 사용자 증가에 따라 발생하는 인증 및 세션 관리 문제 해결을 위한 인증 방식 개선 방안을 만들어야 함

전제조건은 Session timeout 30분, 중복 로그인은 허용하지 않음

1. 사용자 증가에 따른 인증 관리 체계의 부담을 줄이기 위한 방안을 고안하고 설계하시오
단, 각 서비스는 고객 증가 또는 인증 서비스의 장애 요인에 영향을 받지 않아야 함

1. 인증 시스템 아키텍처

- 분산 인증 시스템 (Distributed Authentication System)
 - 사용자가 증가하면서 인증 요청이 폭주할 수 있기 때문에, 인증 서버를 단일화하지 않고 여러 서버로 분산합니다. 이를 통해 특정 서버에 과부하가 걸리지 않도록 하며, 전체 시스템의 성능을 향상시킵니다.
 - 각 인증 서버는 로드 밸런서를 통해 균등하게 분산되며, 특정 서버의 장애 시 자동으로 다른 서버로 요청이 전달됩니다.

2. 세션 관리

- 세션 스토리지 개선
 - Redis 또는 Memcached와 같은 인메모리 데이터 스토어를 활용하여 세션 데이터를 저장하고 관리합니다. 이를 통해 세션 조회 및 갱신 속도를 크게 향상시킬 수 있습니다.
 - 세션 데이터를 공유 스토리지에 저장하여 서버 간 세션 데이터 동기화를 실시간으로 유지합니다. 이를 통해 사용자가 어떤 서버에 연결되더라도 동일한 세션을 사용할 수 있게 합니다.
- Session Timeout
 - 세션 타임아웃을 30분으로 설정하되, 사용자가 일정 기간(예: 25분) 동안 활동이 없을 경우, 자동으로 세션 연장을 위한 요청을 보내도록 구현합니다. 이를 통해 세션 만료 전에 사용자의 작업이 끊기지 않도록 방지할 수 있습니다.

3. 중복 로그인 방지

- JWT 기반의 토큰 인증
 - 사용자가 로그인을 시도할 때마다 고유한 JWT(Json Web Token)를 발급하여 클라이언트에 전달합니다. 이 토큰은 서버에서 검증할 때만 사용되고, 클라이언트는 이 토큰을 통해 모든 API 요청을 인증합니다.
 - 중복 로그인을 방지하기 위해 사용자가 새로 로그인할 때 기존에 발급된 JWT를 무효화하고, 새로운 JWT를 발급합니다. 이를 통해 동일 사용자의 중복 로그인 시 기존 세션이 자동으로 종료되게 합니다.
- 디바이스 및 위치 기반 로그인 관리
 - 동일 사용자가 다른 디바이스에서 로그인할 경우, 이전 세션을 종료하고 새로운 디바이스에서만 로그인이 가능하도록 설정합니다. 또한, 사용자가 다른 위치에서 로그인 시도 시 추가적인 2차 인증을 요구함으로써 보안을 강화합니다.

4. 확장성 및 장애 대응

- 마이크로서비스 아키텍처 도입

- 인증 서비스는 다른 서비스들과 독립적으로 작동하도록 마이크로서비스 아키텍처로 설계합니다. 이를 통해 인증 서비스의 장애가 다른 서비스에 영향을 주지 않도록 하고, 필요에 따라 인증 서비스를 독립적으로 확장할 수 있습니다.

- **캐시 전략 활용**

- 자주 사용되는 인증 관련 데이터는 캐시를 통해 관리하여 데이터베이스 접근을 최소화 하고, 시스템 부하를 줄입니다. 캐시 서버 장애 시, 자동으로 데이터를 다른 캐시 서버로 복제하여 고가용성을 유지합니다.

- **Failover 및 DR(Disaster Recovery)**

- 인증 시스템에 장애가 발생할 경우, 자동으로 백업 시스템으로 전환되는 Failover 설정을 도입합니다. 또한, DR(Disaster Recovery) 시나리오를 마련하여 대규모 장애 발생 시에도 신속하게 복구할 수 있도록 대비합니다.

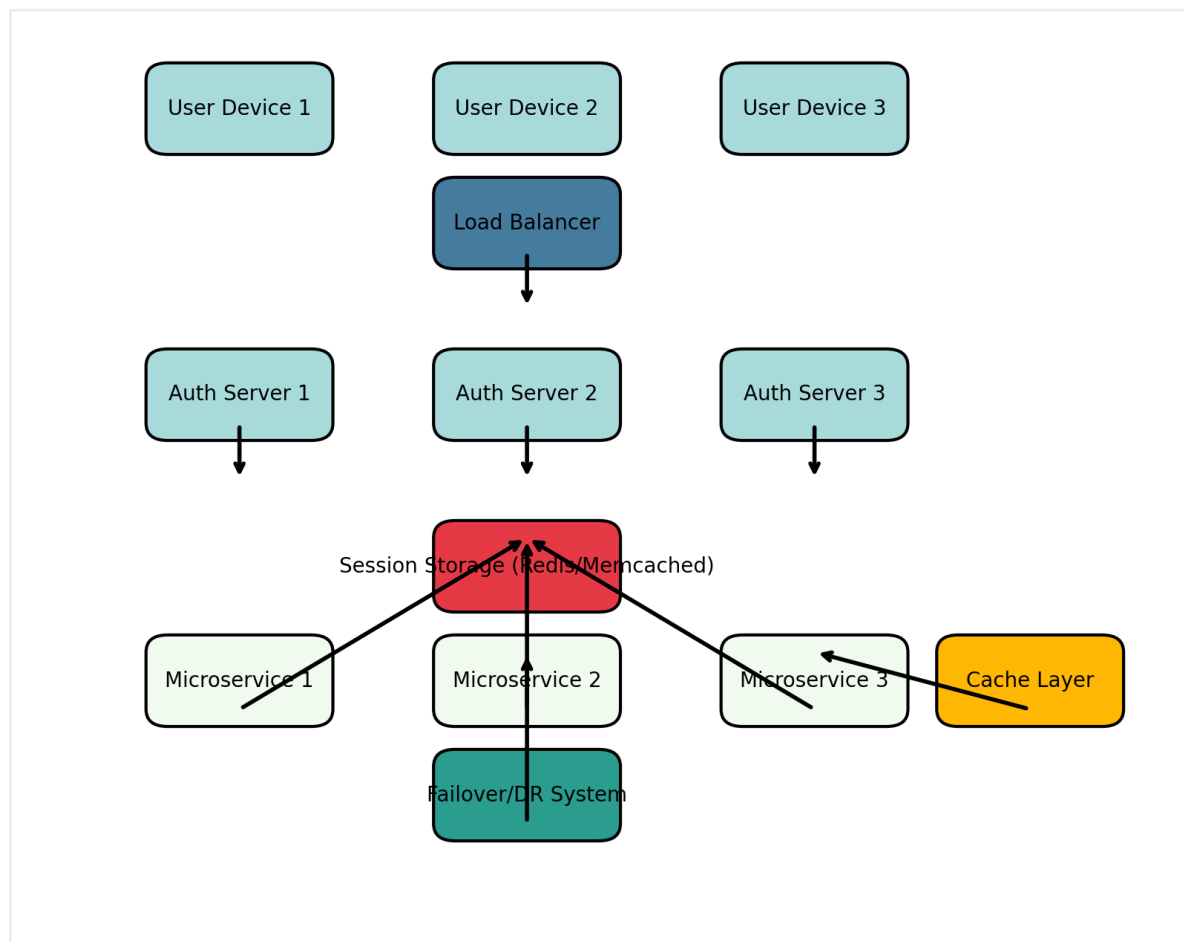
5. 보안 강화

- **다중 인증 (MFA)**

- 사용자가 중요한 작업을 수행하거나 새로운 디바이스에서 로그인할 때, 2차 인증(예: SMS, 이메일, OTP)을 요구하여 보안을 강화합니다.

- **로그 분석 및 이상 탐지**

- 실시간 로그 분석 및 머신러닝 기반의 이상 탐지 시스템을 도입하여 비정상적인 로그인 시도를 즉시 차단하고, 보안 위협에 대한 대응 시간을 단축합니다.



문2) B2B 고객을 위한 제로 트러스트 인증 방식에 대한 프로세스를 설계하시오

제로 트러스트(Zero Trust) 인증 방식은 "절대 신뢰하지 말고 항상 검증하라"는 원칙에 기반하여, 네트워크 내외부를 불문하고 모든 접근 요청에 대해 엄격한 인증과 검증을 수행하는 보안 모델입니다. B2B 고객을 위한 제로 트러스트 인증 방식 프로세스를 설계하는 데 필요한 단계는 다음과 같습니다.

1. 사용자 및 기기 식별

- **등록 및 식별:** 모든 사용자는 네트워크에 접근하기 전에 등록되어야 하며, 고유한 사용자 ID 및 기기 ID가 부여됩니다.
- **기기 인증:** 사용자가 사용하는 기기는 보안 정책에 따라 인증을 받아야 하며, 인증된 기기만 네트워크에 접근할 수 있습니다.
- **사용자 및 기기 프로파일링:** 사용자와 기기의 행동 패턴을 분석하여 정상적인 패턴과 이상 패턴을 식별합니다.

2. 다단계 인증 (Multi-Factor Authentication, MFA)

- **1차 인증:** 사용자 ID와 비밀번호를 사용한 초기 로그인.
- **2차 인증:** 추가적인 인증 요소(예: SMS, 이메일, 모바일 앱 기반의 OTP, 또는 하드웨어 토큰)로 사용자의 신원을 검증합니다.
- **지속적인 인증:** 사용자가 네트워크에 접근한 이후에도 주기적으로 또는 특정 이벤트(예: 민감 데이터 접근) 발생 시 추가 인증을 요구합니다.

3. 최소 권한 원칙 (Principle of Least Privilege)

- **역할 기반 접근 제어 (RBAC):** 사용자의 역할에 따라 접근 가능한 리소스를 제한합니다. 사용자는 업무에 필요한 최소한의 권한만 부여받습니다.
- **동적 권한 할당:** 사용자의 행동 및 기기 상태에 따라 실시간으로 권한이 조정될 수 있습니다. 예를 들어, 사용자가 이상한 시간대나 위치에서 로그인하면 권한이 축소됩니다.

4. 지속적인 모니터링 및 평가

- **실시간 모니터링:** 모든 사용자 활동과 접근 시도를 실시간으로 모니터링하고, 의심스러운 행동이 감지되면 즉각 경고하거나 접근을 차단합니다.
- **행동 분석:** 머신러닝을 사용하여 정상적인 사용자 행동 패턴을 학습하고, 이를 바탕으로 이상 행동을 자동으로 탐지합니다.
- **위협 인텔리전스 통합:** 최신 위협 정보를 바탕으로 접근 시도를 평가하고, 알려진 위협에 대한 접근을 사전에 차단합니다.

5. 네트워크 분할 (Micro-Segmentation)

- **네트워크 격리:** 네트워크를 여러 개의 작은 구역으로 분할하고, 각 구역 간 접근은 엄격하게 통제합니다. 사용자가 특정 구역에 접근하려면 추가적인 인증이 필요할 수 있습니다.
- **세분화된 접근 통제:** 각 구역에 대한 접근 권한을 최소화하여, 권한을 가진 사용자가 아니면 해당 구역에 접근할 수 없도록 설정합니다.

6. 정책 기반 접근 관리

- **정책 엔진:** 접근 요청을 평가하는 중앙 정책 엔진을 구축하여, 사용자의 위치, 시간, 기기 상태, 네트워크 상태 등 다양한 요소를 기반으로 접근 여부를 결정합니다.
- **적응형 정책:** 보안 정책은 실시간으로 변경될 수 있으며, 새로운 위협에 대응하여 자동으로 업데이트됩니다.

7. 데이터 보호 및 암호화

- **데이터 암호화:** 데이터는 전송 중이든 저장 중이든 항상 암호화되어야 합니다. 이를 통해 접근이 허용된 사용자만 데이터를 읽을 수 있습니다.
- **데이터 손실 방지 (DLP):** 민감 데이터의 유출을 방지하기 위해 데이터를 실시간으로 모니

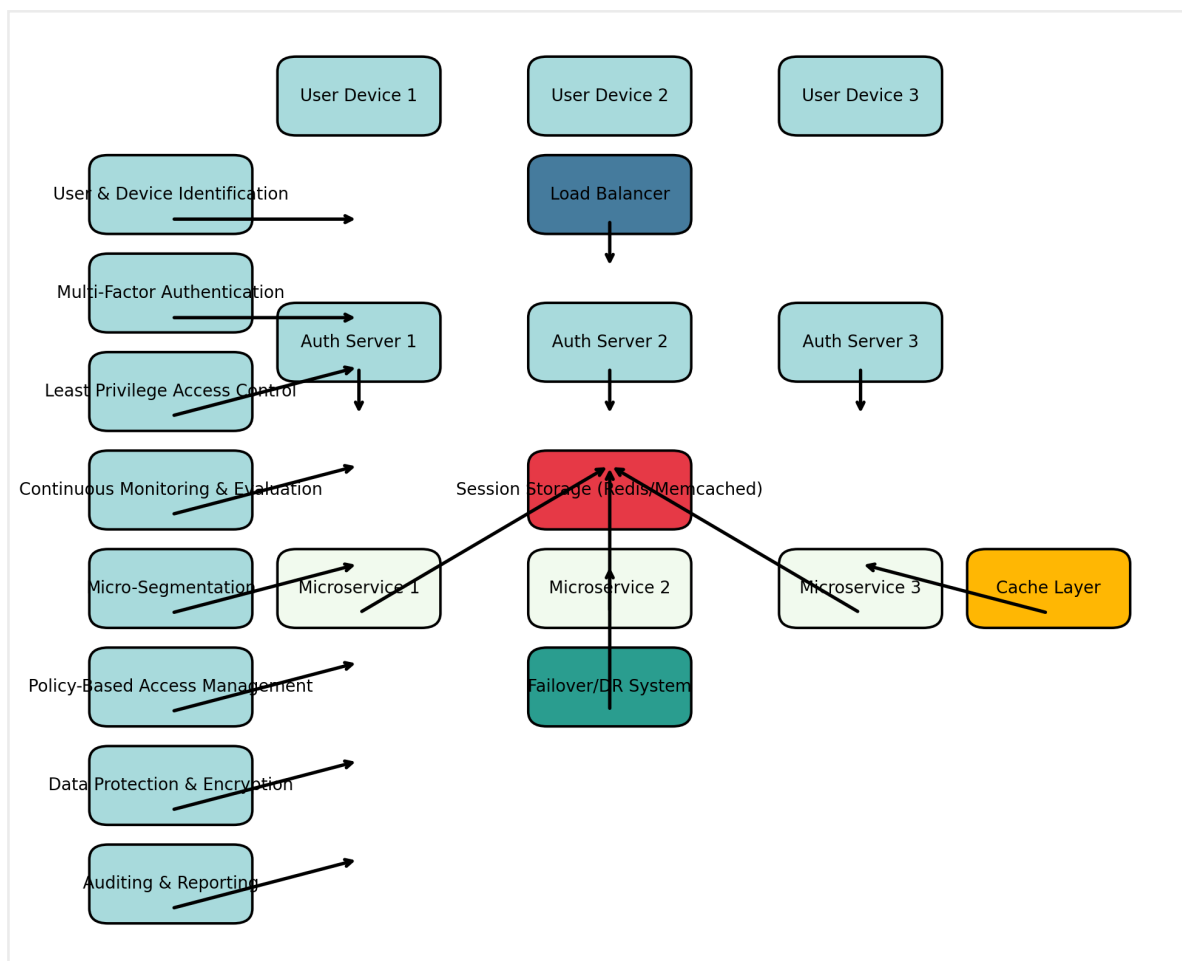
터링하고, 비정상적인 데이터 이동을 차단합니다.

8. 감사 및 보고

- **접근 로그 기록:** 모든 인증 시도, 성공 및 실패 로그를 저장하여 후속 감사에 사용할 수 있도록 합니다.
- **보고서 생성:** 보안 상태 및 접근 활동에 대한 정기적인 보고서를 생성하여 관리자가 시스템의 보안 상태를 항상 파악할 수 있도록 합니다.

9. 자동화 및 대응

- **자동화된 대응:** 의심스러운 활동이 감지되면 자동으로 계정을 잠그거나 권한을 축소하는 등 즉각적인 대응이 가능해야 합니다.
- **사후 대응 계획:** 보안 침해가 발생했을 때 이를 신속하게 대응할 수 있는 사후 대응 계획을 마련해 둡니다.



위 그림은 기존 시스템 구성도에 제로 트러스트 인증 방식을 추가한 것입니다. 이 구성도는 사용자 및 기기 식별, 다단계 인증, 최소 권한 접근 제어, 지속적인 모니터링, 마이크로 세그먼테이션, 정책 기반 접근 관리, 데이터 보호 및 암호화, 그리고 감사 및 보고와 같은 제로 트러스트 보안 요소들을 포함하고 있습니다. 각 요소는 해당 기능을 수행하는 위치에 배치되어, 전체적으로 제로 트러스트 모델을 기반으로 한 보안 강화 프로세스를 시각화합니다.

그림 구성 요소 및 화살표 설명:

1. User & Device Identification:

- 위치: 좌측 상단(1, 14)
- 화살표: **Load Balancer**로 연결됩니다. 이 화살표는 사용자 및 기기 식별이 인증 프

로세스의 첫 단계임을 나타냅니다.

2. Multi-Factor Authentication:

- 위치: 좌측 중간(1, 12)
- 화살표: **Auth Server 1**로 연결됩니다. 사용자가 로그인하면 다단계 인증이 수행됩니다.

3. Least Privilege Access Control:

- 위치: 좌측 중간(1, 10)
- 화살표: **Auth Server 2**로 연결됩니다. 사용자의 권한이 최소한으로 제한됩니다.

4. Continuous Monitoring & Evaluation:

- 위치: 좌측 중간(1, 8)
- 화살표: **Auth Server 3**로 연결됩니다. 지속적으로 사용자의 행동을 모니터링합니다.

5. Micro-Segmentation:

- 위치: 좌측 하단(1, 6)
- 화살표: ****Session Storage (Redis/Memcached)****로 연결됩니다. 네트워크 내에서 세분화된 접근 통제를 구현합니다.

6. Policy-Based Access Management:

- 위치: 좌측 하단(1, 4)
- 화살표: **Microservice 1**로 연결됩니다. 정책에 따라 접근 관리가 이루어집니다.

7. Data Protection & Encryption:

- 위치: 좌측 하단(1, 2)
- 화살표: **Microservice 2**로 연결됩니다. 데이터 보호 및 암호화가 적용됩니다.

8. Auditing & Reporting:

- 위치: 좌측 최하단(1, 0)
- 화살표: **Microservice 3**로 연결됩니다. 모든 접근과 활동이 기록되고 보고됩니다.

기본 시스템 구성 요소 설명:

1. User Devices (User Device 1, 2, 3):

- 화살표: **Load Balancer**로 연결됩니다. 사용자의 요청이 로드 밸런서로 전달됩니다.

2. Load Balancer:

- 화살표: **Distributed Authentication Servers (Auth Server 1, 2, 3)**로 연결됩니다. 요청을 인증 서버로 분산합니다.

3. Distributed Authentication Servers:

- 화살표: **Session Storage**로 연결됩니다. 세션 정보를 저장합니다.

4. Session Storage (Redis/Memcached):

- 화살표: **Microservices**로 연결됩니다. 각 마이크로서비스가 세션 데이터를 사용합니다.

5. Microservices:

- 화살표: **Cache Layer** 및 **Failover/DR System**으로 연결됩니다. 자주 사용하는 데이터를 캐시하며, 장애 발생 시 대체 시스템으로 전환됩니다.

문3) 중복 로그인 배재를 위한 방안을 제안하고 프로세스를 설계하시오

중복 로그인 배제 방안 제안 및 프로세스 설계

중복 로그인 방지는 보안 강화와 계정 무결성 유지에 중요한 역할을 합니다. 이를 구현하기 위해 다음과 같은 방안을 제안하고 프로세스를 설계하겠습니다.

1. 중복 로그인 방지 방안

1.1. 단일 세션 유지 (Single Session Policy)

- **세션 관리:** 각 사용자는 한 번에 하나의 세션만 가질 수 있습니다. 새로운 로그인 시 기존 세션은 자동으로 종료됩니다.
- **세션 상태 추적:** 사용자 로그인 시 세션 ID와 로그인 상태를 중앙 서버 또는 데이터베이스에 저장하여 관리합니다.

1.2. JWT (JSON Web Token) 기반 세션 관리

- **JWT 발급:** 사용자가 로그인할 때 고유한 JWT를 발급받습니다. 이 토큰은 세션과 연계되어 사용됩니다.
- **토큰 갱신 및 폐기:** 새로운 로그인 시 기존 JWT를 폐기하고 새로운 토큰을 발급합니다. 이전 토큰은 더 이상 유효하지 않습니다.

1.3. 사용자 알림 및 인증

- **중복 로그인 시도 시 알림:** 사용자가 다른 장치나 위치에서 로그인하려고 하면, 현재 활성화된 세션을 종료해야 한다는 알림을 받게 됩니다.
- **2차 인증:** 다른 장치에서 로그인을 시도할 경우, 2차 인증(예: OTP)을 통해 본인 확인 절차를 강화합니다.

2. 중복 로그인 방지 프로세스 설계

2.1. 로그인 요청 처리

1. 사용자 로그인 시도:

- 사용자가 로그인 요청을 서버에 보냅니다.

2. 사용자 인증:

- 사용자의 자격 증명을 확인하고, 성공 시 세션을 생성하거나 기존 세션을 확인합니다.

3. 활성 세션 체크:

- 데이터베이스에서 해당 사용자의 기존 세션 상태를 조회합니다.
- 활성 세션이 존재하면, 사용자에게 중복 로그인이 감지되었음을 알리고 다음 단계를 진행합니다.

2.2. 중복 로그인 처리

4. 중복 세션 확인:

- 사용자가 기존 세션을 유지할지 또는 새 로그인을 선택할지 결정하게 합니다.
- 새 로그인 선택 시, 기존 세션을 종료하고 새로운 세션을 활성화합니다.

5. 세션 종료:

- 사용자가 새 로그인을 선택하면, 기존의 활성 세션을 종료하고 이를 데이터베이스에 기록합니다.
- 기존 세션의 JWT를 무효화하여 보안 위험을 최소화합니다.

6. 새 세션 생성:

- 새로운 세션 ID 및 JWT를 발급하여 사용자에게 전달합니다.
- 데이터베이스에 새로운 세션 정보를 업데이트하고, 기존 세션 정보를 삭제 또는 무효화합니다.

2.3. 추가 보안 조치

7. 2차 인증 (MFA) 요청:

- 다른 장치나 IP 주소에서 로그인이 시도될 경우 2차 인증을 요청하여 추가 보안을 제공합니다.

8. 알림 발송:

- 사용자가 중복 로그인을 시도하거나 기존 세션이 종료될 경우 알림을 통해 사용자에게 통보합니다.

9. 세션 모니터링 및 로그 기록:

- 모든 로그인, 로그아웃 및 세션 종료 이벤트를 로깅하고, 비정상적인 세션 활동을 모니터링합니다.
- 비정상적인 활동 감지 시, 즉각적인 대응 조치를 취합니다.

3. 프로세스 흐름도 (시각적 이해)

1. 로그인 시도 → 기존 세션 확인 → 중복 로그인 알림

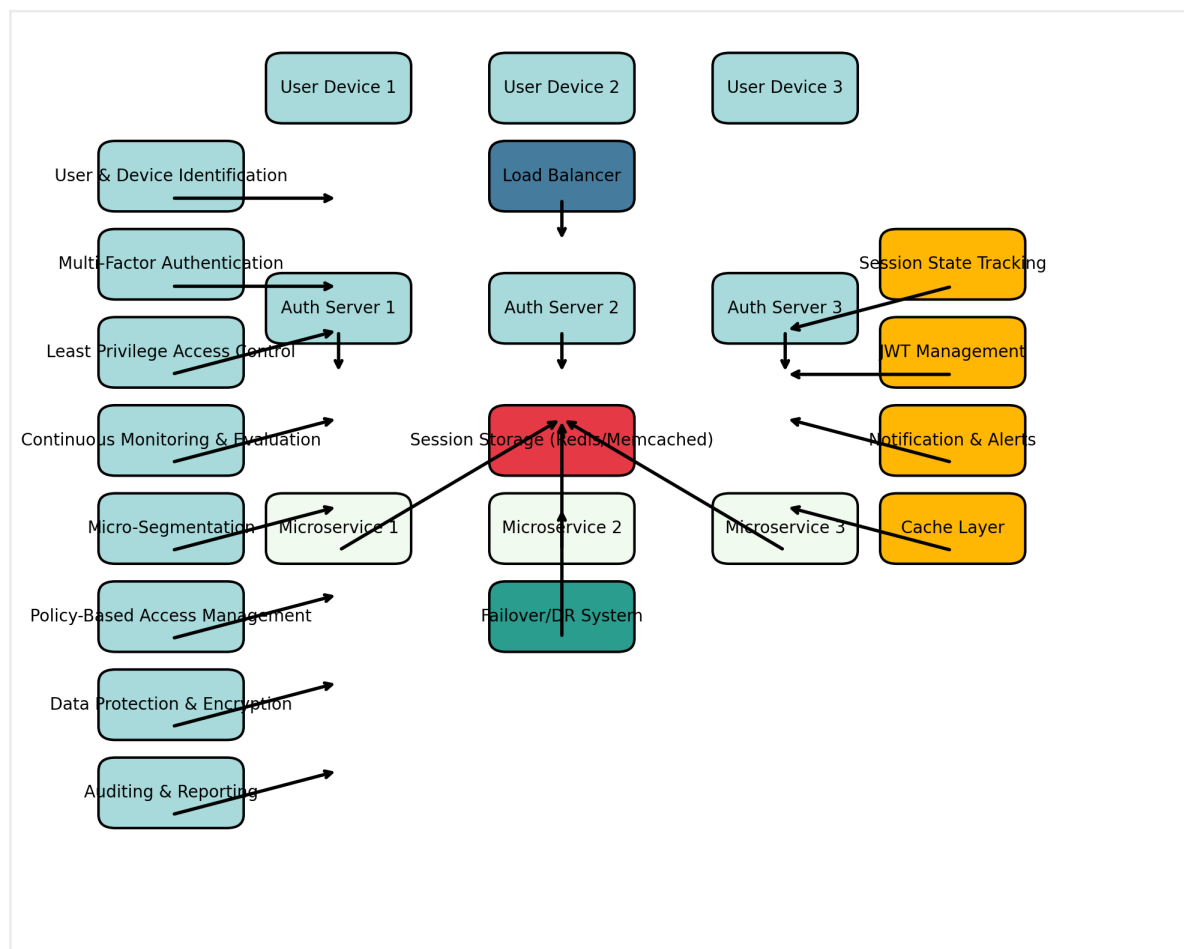
↓

2. 새 세션 선택 시 → 기존 세션 종료 → 새 세션 생성

↓

3. 2차 인증 (필요 시) → 알림 발송 → 로그인 완료

이 과정에서 사용자는 중복 로그인을 방지할 수 있으며, 세션 보안을 강화하여 계정 도용이나 불법 접근을 방지할 수 있습니다. 또한, 세션 활동을 실시간으로 모니터링하여 비정상적인 행위를 즉각 차단할 수 있습니다.



위 그림은 기존 시스템 구성도에 중복 로그인 방지 요소를 추가한 것입니다. 새로운 추가 요소는 다음과 같습니다:

1. Session State Tracking:

- 기존 인증 서버와 연결되어 있으며, 현재 활성화된 세션 상태를 추적하여 중복 로그인을 방지합니다.

2. JWT Management:

- JSON Web Token(JWT)을 관리하며, 새로운 로그인 시 기존 토큰을 무효화하고 새로운 토큰을 발급합니다.

3. Notification & Alerts:

- 중복 로그인 시도가 있을 때 사용자에게 알림을 보내는 시스템으로, 보안 경고 및 세션 종료와 같은 조치를 포함합니다.

새롭게 추가된 요소 및 화살표 설명:

1. Session State Tracking:

- 위치: 우측 상단 (15, 14)
- 화살표: **Auth Server 3**(12, 13)을 가리킵니다.
- 설명: 이 화살표는 Session State Tracking이 인증 서버와 연결되어 사용자의 활성 세션 상태를 추적하고 관리하는 것을 의미합니다. 이를 통해 중복 로그인이 감지되면 적절한 조치가 취해집니다.

2. JWT Management:

- 위치: 우측 중간 (15, 12)
- 화살표: **Auth Server 3**(12, 13)을 가리킵니다.
- 설명: 이 화살표는 JWT 관리 시스템이 인증 서버와 연결되어 JWT의 발급 및 무효화를 처리함을 나타냅니다. 새로운 로그인 시 기존 JWT를 무효화하고 새로운 JWT를 발급하는 역할을 합니다.

3. Notification & Alerts:

- 위치: 우측 하단 (15, 10)
- 화살표: **Microservice 3**(12, 11)을 가리킵니다.
- 설명: 이 화살표는 Notification & Alerts 시스템이 마이크로서비스와 연결되어, 중복 로그인 시도나 세션 종료 시 사용자에게 알림을 보내는 역할을 한다는 것을 나타냅니다.

기존 시스템과의 연결

각 추가 요소는 기존 시스템의 **Auth Server** 및 **Microservices**와 연결되어 있습니다. 이를 통해 중복 로그인 방지 메커니즘이 제대로 작동할 수 있도록 지원합니다.

- **Session State Tracking**과 **JWT Management**는 **Auth Server**와 연결되어 세션 상태를 관리하고 토큰 발급을 처리합니다.
- **Notification & Alerts**는 **Microservices**와 연결되어 사용자에게 실시간으로 알림을 보냅니다.

문4) 인증방식의 전환에 따른 방안을 제시하고 프로세스를 설계하시오

인증 방식 전환 방안 및 프로세스 설계

기존의 인증 방식을 전환하는 것은 보안 강화, 사용자 경험 개선, 그리고 최신 기술 적용을 위해 중요한 단계입니다. 새로운 인증 방식으로의 전환은 주의 깊은 계획과 단계적 접근이 필요합니다. 여기서는 기존의 전통적인 비밀번호 기반 인증 방식을 **다단계 인증(Multi-Factor Authentication, MFA)** 및 **비밀번호 없는 인증>Passwordless Authentication** 방식으로 전환하는 방안을 제시하고, 이를 위한 프로세스를 설계하겠습니다.

1. 전환 방안 제시

1.1. 다단계 인증(Multi-Factor Authentication, MFA) 도입

- **MFA 개념:** 사용자 인증 시, 두 가지 이상의 인증 요소(예: 비밀번호 + OTP, 비밀번호 + 생체인증)를 요구하여 보안을 강화합니다.
- **MFA 적용 범위:** 모든 사용자 계정에 대해 MFA를 단계적으로 도입합니다. 초기에는 고위

험 계정 및 민감 데이터 접근 계정부터 시작하여 전사적으로 확대합니다.

1.2. 비밀번호 없는 인증(Passwordless Authentication) 도입

- **Passwordless 개념:** 비밀번호 대신, 일회성 링크, 생체인식, 푸시 알림 등을 통해 인증을 진행합니다.
- **Passwordless 적용 범위:** 사용자의 편의성을 위해 선택적으로 도입하며, 특히 모바일 앱 사용자의 경험을 개선하는 데 중점을 둡니다.

1.3. 레거시 시스템과의 호환성 유지

- **이중 인증 지원:** 기존 비밀번호 기반 인증과 새로운 인증 방식을 병행하여 사용자의 순조로운 전환을 지원합니다.
- **데이터 마이그레이션 계획:** 기존 사용자 인증 데이터를 새로운 시스템으로 안전하게 이전합니다.

2. 프로세스 설계

2.1. 준비 단계

1. **현재 시스템 분석:**
 - 기존 인증 방식을 분석하고, 현재의 인증 절차와 보안 상태를 평가합니다.
 - 사용자 인증 데이터를 분석하여, 전환이 필요한 사용자 그룹을 식별합니다.
2. **솔루션 선택 및 테스트:**
 - MFA 및 Passwordless 인증 솔루션을 선정하고, 파일럿 테스트를 통해 시스템 호환성 및 성능을 평가합니다.
 - 테스트 결과를 바탕으로 최적의 솔루션을 결정합니다.
3. **교육 및 커뮤니케이션:**
 - 사용자에게 새로운 인증 방식에 대해 교육하고, 전환 일정 및 방법을 공지합니다.
 - 내부 직원 및 IT 지원 팀을 대상으로 전환 절차에 대한 교육을 실시합니다.

2.2. 전환 단계

4. **MFA 도입:**
 - **1단계:** 고위험 계정(예: 관리자, 금융 데이터 접근 계정)에 대해 MFA를 도입합니다.
 - **2단계:** 전사적으로 MFA를 확대 적용하며, 사용자에게 적용 일정 및 절차를 안내합니다.
 - **3단계:** 모든 사용자가 MFA를 설정하고 사용하도록 유도합니다.
5. **Passwordless 도입:**
 - **1단계:** 모바일 사용자 및 특정 서비스에 대해 Passwordless 인증을 시범적으로 도입합니다.
 - **2단계:** 사용자의 피드백을 수집하고, 시스템 안정성을 확인합니다.
 - **3단계:** 단계적으로 Passwordless 인증을 확대 적용합니다.
6. **레거시 시스템 지원:**
 - 기존 인증 방식과 새로운 인증 방식을 병행 지원하여, 사용자가 점진적으로 전환할 수 있도록 유도합니다.
 - 사용자 데이터를 안전하게 마이그레이션하고, 인증 데이터의 무결성을 유지합니다.

2.3. 완료 및 유지 단계

7. **전환 완료 평가:**
 - 전환 과정에서 발생한 문제를 분석하고, 필요한 수정 작업을 수행합니다.
 - 사용자 경험을 평가하고, 추가적인 개선점을 파악합니다.
8. **모니터링 및 지원:**
 - 전환 이후, 인증 시스템을 지속적으로 모니터링하고, 사용자 지원을 강화합니다.
 - 비정상적인 활동이나 보안 위협을 실시간으로 탐지하고 대응합니다.

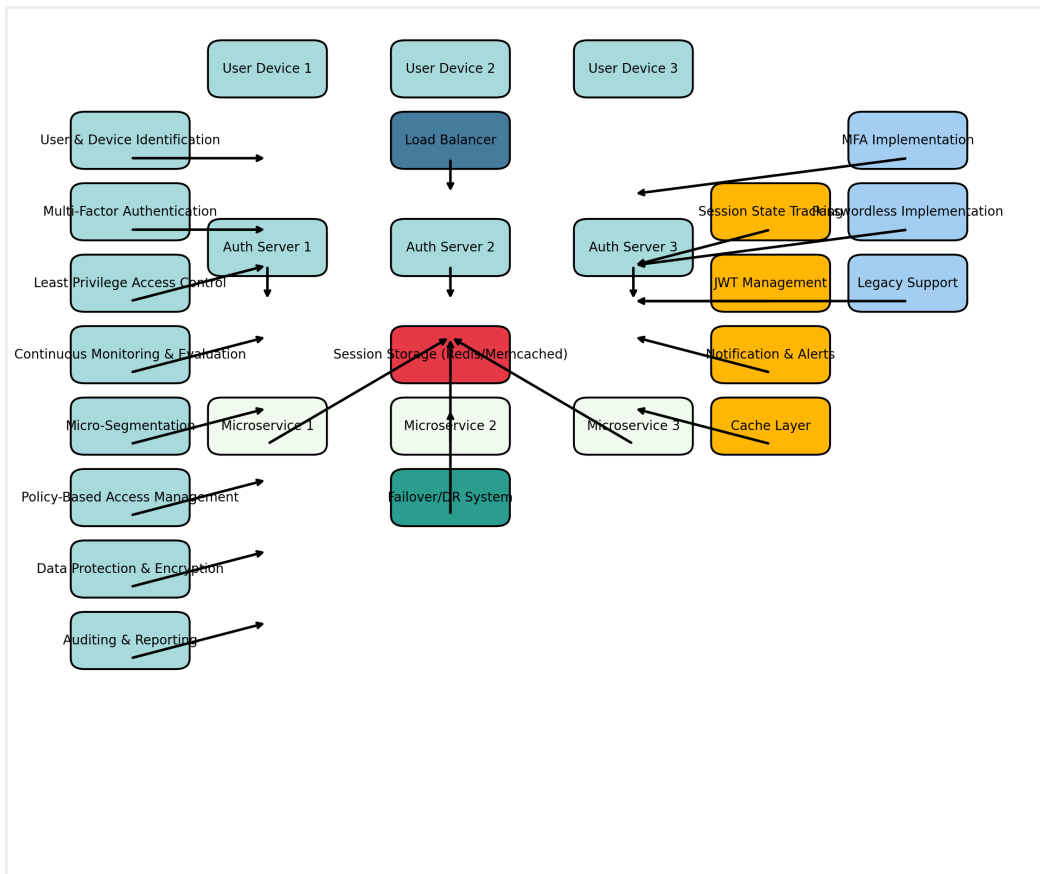
9. 지속적인 개선:

- 새로운 보안 위협에 대응하기 위해 인증 방식을 지속적으로 평가하고, 필요 시 개선 작업을 수행합니다.
- 사용자 피드백을 바탕으로 인증 절차를 최적화하여 사용자 경험을 지속적으로 향상시킵니다.

3. 인증 전환 프로세스 흐름도

1. 시스템 분석 → 솔루션 선택 및 테스트 → 사용자 교육 및 공지 ↓
2. MFA 도입 (고위험 계정 → 전사적 확대) ↓
3. Passwordless 도입 (모바일 → 단계적 확대) ↓
4. 레거시 시스템 지원 → 데이터 마이그레이션 ↓
5. 전환 완료 평가 → 모니터링 및 지속적 개선

이러한 전환 방안을 통해 기존의 비밀번호 기반 인증에서 보다 안전하고 사용자 친화적인 MFA 및 Passwordless 인증 방식으로의 전환이 가능하며, 보안 강화와 사용자 경험 개선을 동시에 달성할 수 있습니다.



새롭게 추가된 요소 및 화살표 설명:

1. MFA Implementation:

- 위치: 우측 상단 (18, 18)
- 화살표: **Auth Server 3** (13, 15)을 가리킵니다.
- 설명: 이 화살표는 다단계 인증(MFA) 구현이 인증 서버와 연결되어 기존 인증 프로세스에 MFA를 추가로 적용함을 나타냅니다. 이로 인해 사용자 로그인 시 추가적인 인증 단계가 요구됩니다.

2. Passwordless Implementation:

- 위치: 우측 중간 (18, 16)
- 화살표: **Auth Server 2** (13, 15)을 가리킵니다.

- **설명:** 이 화살표는 비밀번호 없는 인증 방식이 인증 서버와 연결되어 기존 비밀번호 기반 인증을 대체하거나 보완하는 역할을 한다는 것을 나타냅니다. 이는 사용자 편의성과 보안성을 높이기 위한 방안입니다.

3. Legacy Support:

- **위치:** 우측 중간 (18, 14)
- **화살표:** **Auth Server 1** (13, 14)을 가리킵니다.
- **설명:** 이 화살표는 기존 인증 방식을 지원하기 위한 레거시 시스템 지원이 인증 서버와 연결됨을 나타냅니다. 이는 새로운 인증 방식으로 전환하는 동안 기존 시스템이 계속 작동할 수 있도록 보장합니다.

기존 시스템과의 연결

- **Session State Tracking, JWT Management, 및 Notification & Alerts**는 각각 **Auth Server** 및 **Microservices**와 연결되어 중복 로그인 방지 및 사용자 알림 시스템을 관리합니다.
- **Zero Trust Components**는 다양한 보안 기능과 연결되어 사용자 및 기기 식별, 접근 제어, 모니터링 등을 처리합니다.

화살표의 역할 요약:

- **MFA Implementation**과 **Passwordless Implementation**은 새로운 인증 방식이 기존 시스템에 통합되는 과정에서 중요한 역할을 합니다.
- **Legacy Support**는 기존 사용자들이 새로운 시스템으로 전환하는 동안에도 원활하게 서비스를 이용할 수 있도록 보장합니다.

문5) 인증 불일치 또는 Session 만료에 따른 처리에 대한 UI 처리 방안을 설계하시오

인증 불일치 또는 세션 만료 시 사용자 경험을 고려한 UI 처리 방안을 설계하는 것은 매우 중요합니다. 이를 통해 사용자가 혼란을 겪지 않도록 하고, 보안을 유지하면서도 사용자 친화적인 방법으로 문제를 해결할 수 있습니다. 아래에 UI 처리 방안 및 그에 따른 프로세스를 제안합니다.

1. 인증 불일치 또는 세션 만료 시의 UI 처리 방안

1.1. 사용자 통지 및 설명

- **명확한 메시지 표시:** 인증 불일치 또는 세션 만료 시 사용자에게 명확한 메시지를 표시하여 상황을 설명합니다.
 - 예: "Your session has expired. Please log in again to continue."
 - 예: "Authentication failed. Please check your credentials and try again."
- **시각적 강조:** 메시지를 눈에 띄는 위치에 배치하고, 적절한 색상(예: 빨간색 경고)으로 강조하여 사용자가 즉시 인지할 수 있도록 합니다.

1.2. 자동 리디렉션 또는 팝업

- **리디렉션:** 세션이 만료된 경우, 사용자에게 자동으로 로그인 페이지로 리디렉션합니다.
 - 예: "Redirecting to the login page in 5 seconds..."와 같은 메시지와 함께 리디렉션이 이루어지도록 합니다.
- **팝업 경고:** 인증 불일치 시에는 현재 페이지에서 팝업을 통해 경고 메시지를 띄우고, 사용자에게 로그인 정보를 다시 입력하도록 유도합니다.

1.3. 세션 복구 옵션 제공

- **"Remember Me" 기능:** 세션이 만료되었지만 사용자가 "Remember Me" 기능을 활성화한 경우, 빠르게 다시 로그인할 수 있는 옵션을 제공합니다.
 - 예: "Session expired. Click here to log back in with your saved"

credentials."

- **자동 로그아웃 타이머:** 사용자가 일정 시간(예: 2분) 동안 활동이 없을 경우, 세션이 곧 만료된다는 경고를 보내고 자동 로그아웃까지 남은 시간을 표시합니다.

1.4. 재인증 프로세스 간소화

- **이전 페이지 복구:** 사용자가 로그인을 완료하면, 인증 불일치나 세션 만료가 발생했던 페이지로 다시 돌아갈 수 있도록 합니다.
 - 예: "You have been re-authenticated. Returning to your previous page..."
- **간편 로그인 제공:** 추가 인증 수단(예: OTP 또는 생체인식)이 필요할 경우, 간단한 인터페이스를 통해 빠르게 진행할 수 있도록 합니다.

1.5. 오류 로그 및 사용자 지원

- **오류 상세 정보 제공:** 사용자가 문제를 해결할 수 있도록 간단한 오류 코드를 제공하고, 자세한 도움말 링크를 제공합니다.
 - 예: "Error Code: 401. Click here for help."
- **고객 지원 연결:** 문제 해결이 어려울 경우, 즉시 고객 지원에 문의할 수 있는 버튼을 제공하여 도움을 받을 수 있도록 합니다.

2. UI 처리 방안에 따른 프로세스 설계

2.1. 세션 만료 시의 처리

1. **세션 만료 감지:**
 - 사용자 활동이 없거나 세션 타임아웃 발생 시 세션 만료를 감지합니다.
2. **자동 리디렉션:**
 - 만료된 세션을 감지한 후, 로그인 페이지로 자동 리디렉션합니다.
 - 사용자에게 만료 메시지와 함께 "Remember Me" 기능을 사용할 수 있는 옵션을 제공합니다.
3. **재인증 후 페이지 복구:**
 - 사용자가 로그인하면, 이전에 있던 페이지로 복구되도록 처리합니다.

2.2. 인증 불일치 시의 처리

1. **인증 불일치 감지:**
 - 사용자가 잘못된 자격 증명을 입력하거나 다른 원인으로 인해 인증이 실패하면 이를 감지합니다.
2. **팝업 경고 및 재입력 유도:**
 - 현재 페이지에서 팝업을 통해 경고 메시지를 띄우고, 사용자가 다시 로그인할 수 있도록 입력 폼을 제공합니다.
3. **오류 로그 및 지원:**
 - 오류가 반복되면, 사용자에게 상세한 도움말을 제공하고 고객 지원에 문의할 수 있는 옵션을 제공합니다.

2.3. 타이머 기반 자동 로그아웃

1. **타이머 시작:**
 - 사용자 활동이 감지되지 않으면 일정 시간 후 자동 로그아웃 타이머가 시작됩니다.
2. **경고 메시지:**
 - 타이머가 만료되기 전에 "세션이 곧 만료됩니다"라는 경고 메시지를 표시합니다.
3. **자동 로그아웃 및 리디렉션:**
 - 사용자가 활동을 재개하지 않으면 자동으로 로그아웃되고, 로그인 페이지로 리디렉션됩니다.

3. UI 시나리오

1. 세션 만료 발생 → 세션 만료 메시지 표시 → 로그인 페이지로 리디렉션 → 재로그인 후 이전 페이지로 복구
2. 인증 불일치 발생 → 인증 실패 팝업 표시 → 사용자가 자격 증명 재입력 → 성공 시 정상 진행, 실패 시 추가 지원 옵션 제공
3. 사용자 비활동 → 자동 로그아웃 타이머 시작 → 경고 메시지 표시 → 타이머 만료 시 자동 로그아웃 및 리디렉션

4. 디자인 및 사용자 경험 고려사항

- **명확성:** 메시지는 간결하고 명확해야 하며, 사용자가 현재 상황을 쉽게 이해할 수 있도록 해야 합니다.
- **즉각적인 피드백:** 사용자가 잘못된 정보를 입력할 경우 즉시 피드백을 제공하여 빠르게 수정할 수 있도록 합니다.
- **유연성:** 사용자가 다양한 디바이스와 환경에서 동일한 경험을 할 수 있도록 반응형 디자인을 고려합니다.
- **보안 강화:** 인증 불일치나 세션 만료 시 비밀번호를 재입력하거나 2차 인증을 통해 보안을 강화합니다.

1. 세션 만료 처리 프로세스

- **Session Activity:** 세션 활동을 모니터링합니다.
- **Session Expired?:** 세션이 만료되었는지 확인합니다. 만료된 경우, 다음 단계로 진행됩니다.
- **Display Expiration Message:** 사용자에게 세션 만료 메시지를 표시합니다.
- **Redirect to Login Page:** 사용자를 로그인 페이지로 리디렉션합니다.
- **Re-authenticate User:** 사용자가 다시 로그인합니다.
- **Restore Previous Page:** 사용자가 로그인한 후, 이전 페이지로 복원됩니다.

2. 인증 불일치 처리 프로세스

- **Authentication Failure?:** 인증 실패 여부를 확인합니다. 실패한 경우, 다음 단계로 진행됩니다.
- **Display Failure Popup:** 사용자에게 인증 실패 팝업을 표시합니다.
- **Prompt for Credentials:** 사용자가 자격 증명을 다시 입력하도록 요청합니다.
- **Retry Authentication:** 사용자가 다시 인증을 시도합니다.
- **Provide Support Options:** 반복적인 실패가 발생하면, 고객 지원 옵션을 제공합니다.

3. 자동 로그아웃 처리 프로세스

- **Inactivity Detected?:** 사용자의 비활동을 감지합니다.
- **Start Logout Timer:** 자동 로그아웃 타이머를 시작합니다.
- **Show Warning Message:** 세션이 곧 만료된다는 경고 메시지를 표시합니다.
- **Auto Logout & Redirect:** 타이머가 만료되면 자동으로 로그아웃하고, 로그인 페이지로 리디렉션합니다.

[주제2]

A고객사 비즈니스 유연성과 성능 관점에서 상품을 관리하기 위한 데이터 모델을 새롭게 설계하고

설계 사유제시

<전제 조건>

• 상황에 따른 ERD제공 됨

문1) A고객사 비즈니스 유연성과 성능 관점에서 상품을 관리하기 위한 데이터 모델을 새롭게 설계하고 설계 사유제시하려고 함. 전제조건으로 상황에 따른 ERD제공 됨

예약 위주의 상품 관련 데이터 모델에 실시간 판매 가능한 상품 유형을 추가 하기 위한 방안을 제안하고 설계 하시오

데이터 모델 설계 방안

A 고객사의 비즈니스 유연성과 성능을 고려하여, 기존 예약 위주의 상품 관련 데이터 모델에 실시간 판매 가능한 상품 유형을 추가하기 위해 다음과 같은 데이터 모델을 설계하고자 합니다.

1. 기존 데이터 모델 분석

기존 데이터 모델은 주로 예약 중심으로 설계되어 있으며, 예약 시점에 따라 상품의 가용성을 관리합니다. 주요 엔티티는 상품(Product), 예약(Reservation), 고객(Customer) 등으로 구성되어 있습니다.

2. 실시간 판매 가능한 상품 유형 추가 방안

실시간 판매 가능한 상품을 추가하기 위해서는 다음과 같은 새로운 엔티티와 관계를 추가해야 합니다:

- **실시간 판매 가능한 상품 유형(Product Type)**

- 실시간으로 판매 가능한 상품 유형을 관리하는 엔티티를 추가합니다. 예를 들어, ProductType 테이블에 실시간(RealTime)과 예약(Reservation) 등의 상품 유형을 정의합니다.

- **재고 관리(Inventory)**

- 실시간 판매 가능한 상품은 실시간으로 재고를 확인할 수 있어야 합니다. 이를 위해 Inventory 엔티티를 추가하고, 실시간 상품의 재고 수량을 관리합니다. 이 엔티티는 Product와 일대일 또는 일대다 관계를 가질 수 있습니다.

- **실시간 주문(RealTimeOrder)**

- 실시간 판매 시, 주문을 즉시 처리해야 하므로, RealTimeOrder 엔티티를 추가하여 실시간 주문을 기록합니다. 이 엔티티는 Customer 및 Product와 연관됩니다.

3. 새로운 데이터 모델 설계 (ERD)

1. Product (상품)

- 기존 Product 엔티티에 ProductTypeID를 추가하여, 상품이 실시간 판매 가능한지 또는 예약이 필요한지 구분할 수 있도록 합니다.

2. ProductType (상품 유형)

- ProductTypeID (PK)
- ProductTypeName (예: 실시간, 예약)

3. Inventory (재고)

- InventoryID (PK)
- ProductID (FK)
- QuantityAvailable (현재 재고 수량)
- LastUpdated (마지막 재고 업데이트 시간)

4. RealTimeOrder (실시간 주문)

- OrderID (PK)
- ProductID (FK)
- CustomerID (FK)

- OrderDate (주문 일자)
- Quantity (주문 수량)
- OrderStatus (주문 상태)

4. 데이터 모델 설계 사유

4.1. 비즈니스 유연성 강화

- **유형별 관리:** ProductType 엔티티를 통해 예약과 실시간 판매를 구분하여 관리할 수 있으며, 새로운 상품 유형을 추가할 때도 유연하게 대응할 수 있습니다.
- **재고 관리:** 실시간 판매가 가능한 상품의 경우 재고 관리를 통해 비즈니스 로직을 효율적으로 처리할 수 있습니다. 재고 수량을 실시간으로 관리함으로써, 상품의 가용성을 정확하게 파악하고 판매할 수 있습니다.

4.2. 성능 최적화

- **효율적인 주문 처리:** 실시간 주문을 별도로 관리함으로써, 예약과 실시간 판매 간의 성능 문제를 분리하고, 각각의 프로세스를 최적화할 수 있습니다.
- **빠른 재고 확인:** 재고 확인이 빠르게 이루어져야 실시간 판매가 원활하게 진행될 수 있습니다. 이를 위해 Inventory 엔티티를 별도로 두어 실시간 조회 성능을 최적화합니다.

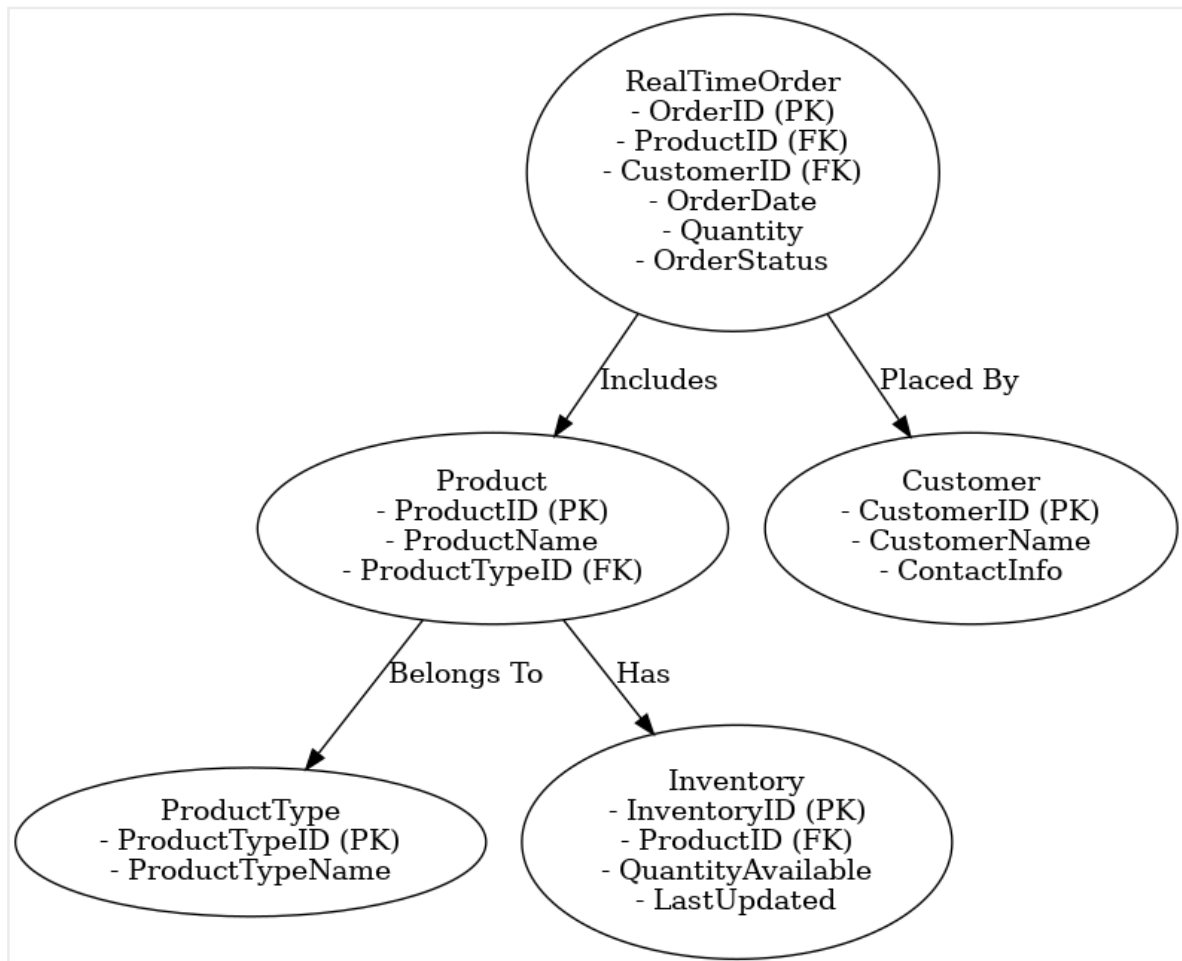
5. 실시간 상품 유형 추가에 따른 ERD 예시

ERD 주요 요소:

- **Product:** 기존 제품 정보를 관리하며, ProductTypeID를 추가하여 제품 유형을 구분.
- **ProductType:** 제품의 유형을 정의 (예: 실시간, 예약).
- **Inventory:** 실시간 상품의 재고를 관리하는 엔티티.
- **RealTimeOrder:** 실시간 주문을 관리하는 엔티티.

6. 결론

이 설계는 A 고객사의 비즈니스 유연성과 성능을 동시에 고려한 접근법으로, 실시간 상품 판매의 효율성과 성능 최적화를 목표로 합니다. 새로운 데이터 모델은 기존 예약 중심의 구조를 확장하여, 실시간 판매의 요구사항을 효과적으로 처리할 수 있도록 설계되었습니다. 이를 통해 실시간 판매와 예약 상품의 통합 관리가 가능해지며, 비즈니스 확장성 또한 확보할 수 있습니다.



문2) 자사 직접 판매 상품뿐만 아닌 입점 업체에 대한 상품 주문 관련 서비스 모델을 설계 하시오
 입점 업체의 상품 주문 관련 서비스 모델을 설계하는 것은 복잡한 비즈니스 로직을 포함하며, 자사 상품과 입점 업체 상품 간의 차이를 고려해야 합니다. 아래에 제안된 데이터 모델과 함께, 서비스 모델 설계 방안을 제시하겠습니다.

1. 서비스 모델 설계 방안

1.1. 기존 모델 확장

- **기존 상품 엔티티 확장**: Product 엔티티를 확장하여 자사 상품과 입점 업체 상품을 구분할 수 있도록 합니다. 이를 위해 SellerType 필드(예: 'Direct', 'Vendor')를 추가합니다.
- **입점 업체 엔티티 추가**: Vendor 엔티티를 추가하여 입점 업체 정보를 관리하고, 이와 연관된 상품을 관리합니다.

1.2. 주문 관리 확장

- **Order 엔티티 확장**: 주문 테이블에서 자사 상품과 입점 업체 상품을 함께 관리할 수 있도록, Order 엔티티를 확장합니다. 각 주문이 어떤 판매자(Seller)에 의해 처리되는지 구분할 수 있는 필드를 추가합니다.
- **OrderItem 엔티티 추가**: 주문 아이템을 관리하는 OrderItem 엔티티를 추가하여, 주문 내에 여러 상품을 관리할 수 있도록 합니다.

1.3. 배송 및 결제 관리

- **배송 관리**: Shipping 엔티티를 추가하여 자사와 입점 업체 상품의 배송을 각각 관리할 수 있도록 합니다.
- **결제 관리**: Payment 엔티티를 통해 자사와 입점 업체의 결제 흐름을 각각 관리합니다.

2. 새로운 데이터 모델 설계 (ERD)

2.1. 주요 엔티티

1. Product (상품)

- ProductID (PK)
- ProductName
- SellerType (자사 또는 입점 업체 구분, Direct/Vendor)
- VendorID (FK, 입점 업체 상품인 경우)

2. Vendor (입점 업체)

- VendorID (PK)
- VendorName
- ContactInfo

3. Order (주문)

- OrderID (PK)
- CustomerID (FK)
- OrderDate
- TotalAmount
- OrderStatus
- ShippingID (FK)

4. OrderItem (주문 아이템)

- OrderItemID (PK)
- OrderID (FK)
- ProductID (FK)
- Quantity
- Price
- VendorID (FK, 입점 업체 상품인 경우)

5. Shipping (배송)

- ShippingID (PK)
- ShippingMethod
- ShippingDate
- TrackingNumber
- VendorID (FK, 입점 업체 상품인 경우)

6. Payment (결제)

- PaymentID (PK)
- OrderID (FK)
- PaymentDate
- PaymentMethod
- PaymentStatus

3. 데이터 모델 설계 사유

3.1. 비즈니스 확장성

- **입점 업체 상품 관리:** Vendor 엔티티를 통해 입점 업체의 상품과 자사 상품을 유연하게 관리할 수 있습니다. 이를 통해 다양한 판매 채널을 통해 상품을 판매할 수 있는 비즈니스 확장성을 확보할 수 있습니다.
- **복합 주문 처리:** OrderItem 엔티티를 추가하여, 한 주문 내에서 자사 상품과 입점 업체 상품을 함께 처리할 수 있습니다.

3.2. 운영 효율성

- **독립적 배송 관리:** 자사와 입점 업체의 배송을 별도로 관리함으로써, 각기 다른 배송 정책과 절차를 적용할 수 있습니다.
- **결제 관리 통합:** 결제 흐름을 통합 관리하되, 자사와 입점 업체 간의 수익 분배를 명확히 할 수 있습니다.

4. 입점 업체 상품 주문 서비스 모델에 따른 ERD 예시

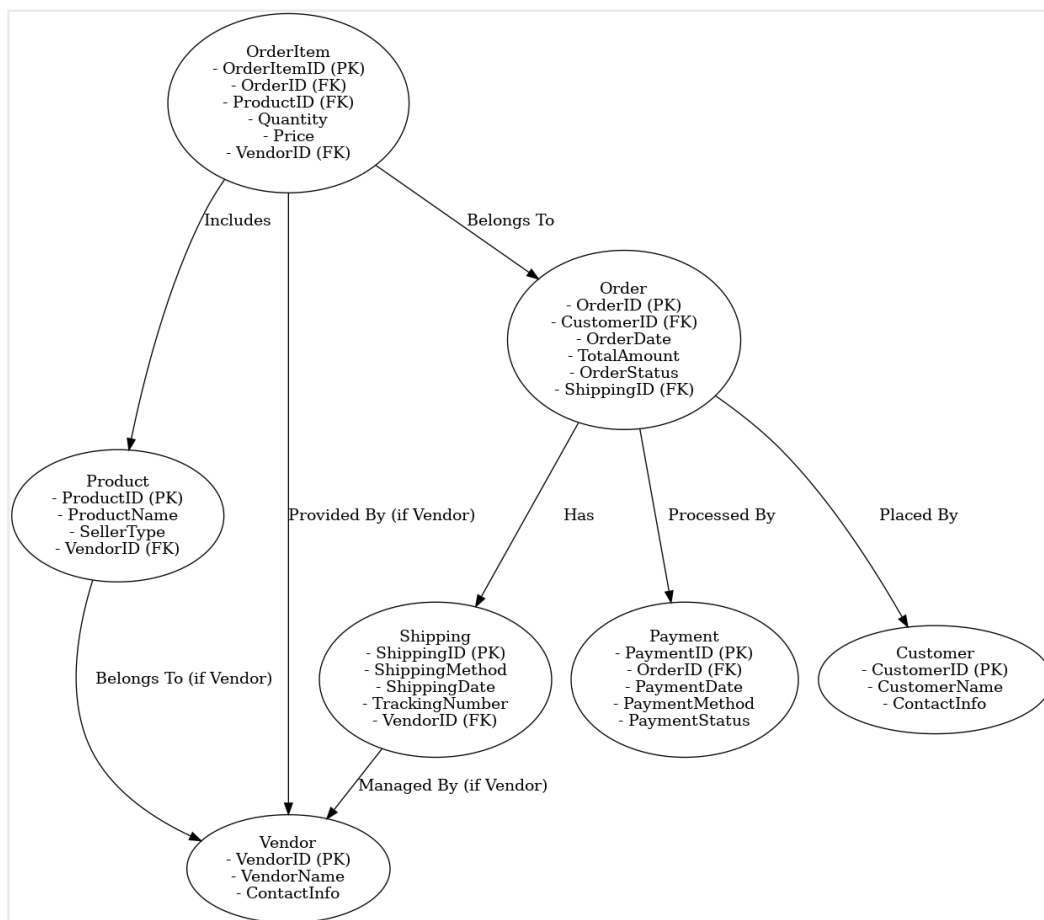
ERD 주요 요소:

- **Product:** 상품 정보와 판매자 유형을 관리.
- **Vendor:** 입점 업체 정보 관리.
- **Order:** 주문 관리.
- **OrderItem:** 주문 내 각 상품을 관리.
- **Shipping:** 배송 관리.
- **Payment:** 결제 정보 관리.

5. 결론

이 모델은 A 고객사가 자사 상품뿐만 아니라 입점 업체의 상품을 효과적으로 관리할 수 있도록 지원합니다. 입점 업체의 상품을 자사 시스템에 통합함으로써 다양한 상품을 한 주문 내에서 처리할 수 있으며, 비즈니스 확장성과 운영 효율성을 동시에 달성할 수 있습니다.

이제, 제안된 데이터 모델에 따른 ERD를 시각적으로 표현하겠습니다.



문3) 상품 주문 절차에 대한 프로세스를 설계 하시오

입점 업체 상품을 포함한 전체 상품 주문 절차를 설계하는 것은 여러 이해관계자의 요구를 충족시키고, 사용자 경험을 최적화하며, 효율적인 주문 처리를 보장하기 위한 핵심적인 작업입니다. 아

래에 제안된 상품 주문 절차에 대한 프로세스 설계를 소개합니다.

상품 주문 절차 프로세스 설계

1. 상품 선택 및 장바구니 추가

1. 상품 검색 및 선택

- 사용자는 자사 상품과 입점 업체 상품을 검색하고, 상세 정보를 확인한 후, 구매할 상품을 선택합니다.
- 상품의 가용성을 실시간으로 확인하여, 사용자가 선택한 상품이 주문 가능한지 여부를 안내합니다.

2. 장바구니 담기

- 사용자는 선택한 상품을 장바구니에 추가합니다.
- 자사 상품과 입점 업체 상품이 함께 장바구니에 추가될 수 있으며, 각 상품의 판매자 정보와 예상 배송비가 표시됩니다.

2. 주문서 작성 및 확인

3. 장바구니 확인 및 수정

- 사용자는 장바구니에 담긴 상품을 확인하고, 수량 변경이나 삭제와 같은 수정 작업을 할 수 있습니다.
- 입점 업체 상품의 경우, 각 업체별로 배송비가 다를 수 있음을 사용자에게 명확히 표시합니다.

4. 주문서 작성

- 사용자는 주문서를 작성하며, 배송지 정보, 결제 방법, 배송 방법 등을 입력합니다.
- 자사와 입점 업체 상품의 경우, 각각의 배송 방법과 예상 배송 기간이 별도로 표시됩니다.

5. 주문서 확인 및 최종 검토

- 사용자는 주문서를 제출하기 전에 최종 검토 페이지에서 전체 주문 내역, 결제 금액, 배송비 등을 확인합니다.
- 사용자에게 주문 확정 전에 자사 상품과 입점 업체 상품의 별도 배송에 대한 정보를 다시 한번 명확히 안내합니다.

3. 결제 및 주문 완료

6. 결제 처리

- 사용자가 선택한 결제 방법에 따라 결제가 처리됩니다. (예: 신용카드, 페이팔, 은행 송금 등)
- 결제 성공 여부가 실시간으로 확인되며, 결제가 완료된 후 주문이 확정됩니다.
- 결제가 성공적으로 완료되면, 각 입점 업체에 주문이 자동으로 전달됩니다.

7. 주문 확인 및 알림

- 주문이 확정되면, 사용자는 주문 확인 페이지로 이동하며, 주문 번호와 함께 주문 내역이 표시됩니다.
- 이메일 또는 SMS를 통해 주문 확인서와 결제 영수증을 사용자에게 발송합니다.

8. 주문 상태 추적

- 사용자는 주문 내역 페이지에서 주문 상태를 실시간으로 추적할 수 있습니다.
- 자사와 입점 업체의 상품 각각에 대한 배송 추적 정보가 개별적으로 제공됩니다.

4. 배송 및 배송 완료

9. 상품 준비 및 배송

- 자사 상품은 자체 물류 창고에서, 입점 업체 상품은 각 업체에서 개별적으로 상품 준비 및 배송이 진행됩니다.
- 각 상품의 준비 상태가 주문 내역에 업데이트되며, 준비 완료 시 배송이 시작됩니다.

10. 배송 완료 및 피드백 요청

- 모든 상품이 배송 완료되면, 사용자는 주문 내역에서 배송 완료를 확인할 수 있습니다.
- 배송이 완료된 후, 사용자에게 피드백을 요청하여 서비스 개선에 반영합니다.

5. 프로세스 흐름도

1. 상품 검색 및 선택 → 장바구니 담기
2. 장바구니 확인 및 수정 → 주문서 작성 → 주문서 확인 및 최종 검토
3. 결제 처리 → 주문 확인 및 알림 → 주문 상태 추적
4. 상품 준비 및 배송 → 배송 완료 및 피드백 요청

6. 설계 사유

6.1. 사용자 경험 최적화

- **명확한 정보 제공:** 주문 과정에서 자사 상품과 입점 업체 상품의 차이점을 명확히 안내함으로써 사용자 혼란을 줄입니다.
- **단일 장바구니:** 자사 상품과 입점 업체 상품을 하나의 장바구니에서 관리할 수 있도록 하여, 사용자의 편의성을 높입니다.

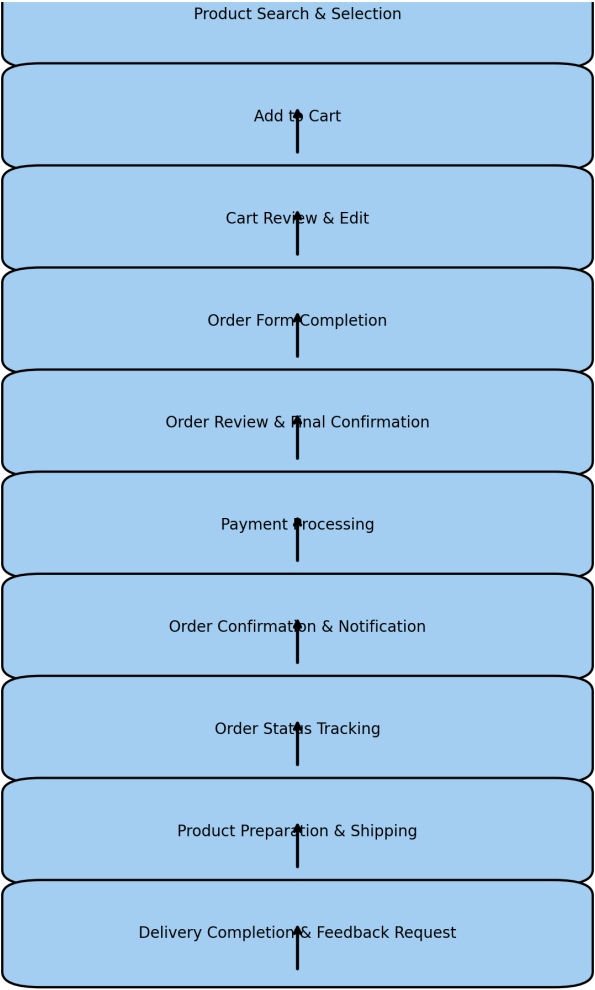
6.2. 효율적인 운영

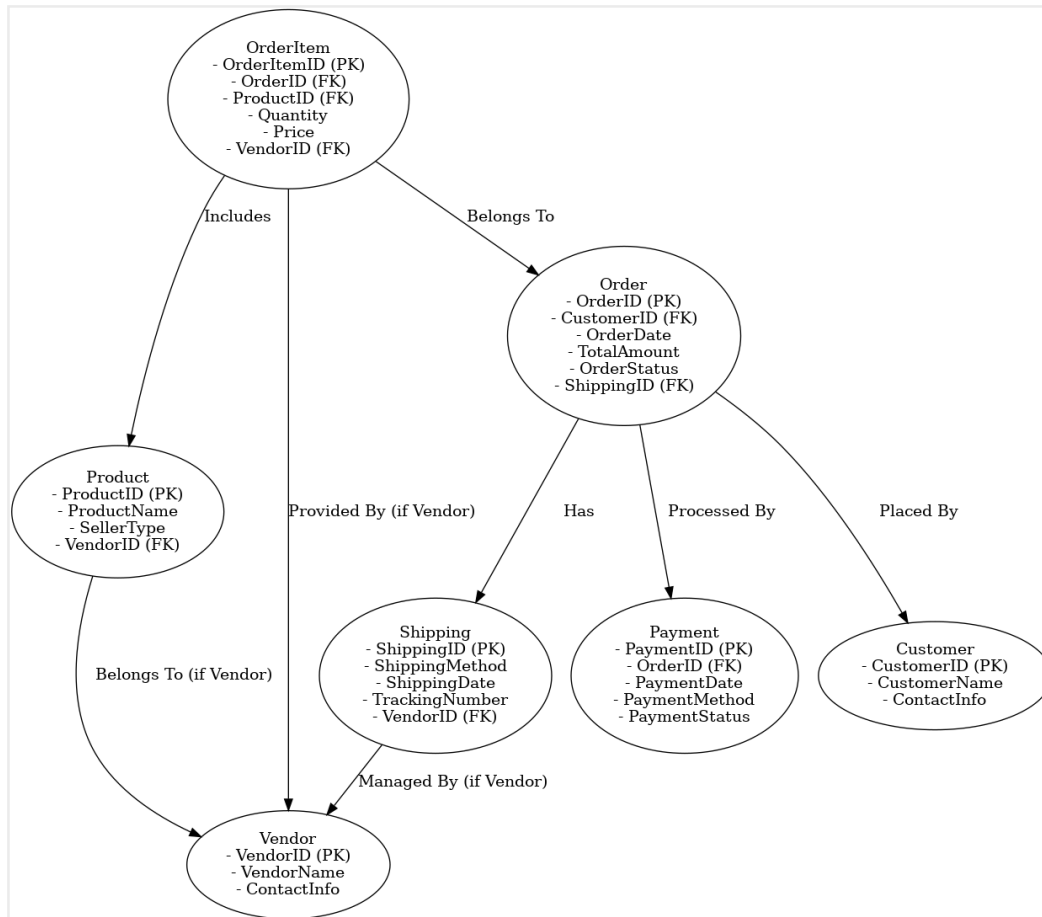
- **별도 배송 관리:** 자사와 입점 업체의 상품을 각각 관리하되, 전체 주문은 통합하여 처리함으로써 운영의 효율성을 확보합니다.
- **실시간 추적:** 주문 상태와 배송 추적 정보를 실시간으로 제공하여 사용자 만족도를 높입니다.

6.3. 비즈니스 확장성

- **입점 업체 지원:** 입점 업체의 상품 주문 처리와 배송 관리를 기존 시스템에 통합함으로써 비즈니스 확장성을 높이고, 다양한 상품을 제공할 수 있습니다.

이 프로세스 설계는 A 고객사가 자사와 입점 업체의 상품을 효과적으로 관리하고, 사용자에게 최적의 구매 경험을 제공할 수 있도록 지원합니다.





문4) 상품 출시에 대한 라이프 사이클을 관리 하는 모델을 제시하고 설계 사유를 제시하시오

상품 출시 라이프 사이클 관리 모델 제안

상품 출시 라이프 사이클을 관리하는 데이터 모델은 상품의 초기 기획부터 출시, 판매 종료까지의 모든 단계를 체계적으로 관리할 수 있도록 설계되어야 합니다. 이 모델은 각 단계에서 발생하는 주요 이벤트와 상태 변화를 효과적으로 추적하고 관리할 수 있는 구조를 제공해야 합니다.

1. 상품 출시 라이프 사이클 관리 모델 설계

1.1. 주요 엔티티

1. Product (상품)

- ProductID (PK)
- ProductName
- ProductDescription
- ProductCategory
- LaunchDate (출시일)
- EndDate (판매 종료일)
- CurrentStatusID (FK, 현재 상태)

2. ProductStatus (상품 상태)

- StatusID (PK)
- StatusName (예: 기획 중, 개발 중, 출시 대기, 출시 중, 판매 종료)
- StatusDescription

3. ProductLifecycleEvent (상품 라이프 사이클 이벤트)

- EventID (PK)
- ProductID (FK)

- StatusID (FK, 상태 변경을 나타냄)
- EventDate (이벤트 발생일)
- Notes (이벤트 관련 추가 정보)

4. ProductVersion (상품 버전)

- VersionID (PK)
- ProductID (FK)
- VersionNumber (예: 1.0, 2.0)
- ReleaseDate (버전 출시일)
- VersionDescription

5. TeamMember (팀 멤버)

- TeamMemberID (PK)
- Name
- Role
- ContactInfo

6. ProductTeamAssignment (상품 팀 할당)

- AssignmentID (PK)
- ProductID (FK)
- TeamMemberID (FK)
- Role (예: 제품 매니저, 개발자, 디자이너)
- AssignedDate (할당일)

2. 모델 설계 사유

2.1. 유연한 상태 관리

- **ProductStatus 엔티티**: 상품의 상태를 유연하게 정의하고 관리할 수 있습니다. 예를 들어, 상품이 기획 단계에서 개발 단계로, 그리고 출시 단계로 이동하는 과정이 명확하게 정의됩니다.
- **ProductLifecycleEvent 엔티티**: 각 상태 변경과 관련된 이벤트를 기록하여, 상품이 어느 시점에 어떤 상태였는지 추적할 수 있습니다. 이를 통해 출시 지연, 개발 이슈 등의 원인을 분석할 수 있습니다.

2.2. 상세한 버전 관리

- **ProductVersion 엔티티**: 상품의 버전별 관리가 가능하여, 새로운 버전 출시 시의 변경 사항과 주요 특징을 기록할 수 있습니다. 이를 통해 버전 간의 차이점을 명확히 하고, 특정 버전에 문제가 발생했을 때 과거 버전과 비교할 수 있습니다.

2.3. 팀 협업 관리

- **ProductTeamAssignment 엔티티**: 상품 개발과 출시 과정에서 참여하는 팀 멤버들을 관리할 수 있습니다. 각 팀원이 언제, 어떤 역할로 참여했는지를 기록함으로써 협업 과정을 투명하게 관리할 수 있습니다.

3. 상품 출시 라이프 사이클 관리 프로세스

1. 상품 기획:

- 상품이 기획 단계에 들어가면 Product 엔티티가 생성되고, 초기 상태로 "기획 중"이 설정됩니다.
- 관련 팀원들이 ProductTeamAssignment 엔티티를 통해 할당됩니다.

2. 개발 및 테스트:

- 상품이 개발 단계로 이동하면 ProductLifecycleEvent가 기록되며, 상태가 "개발 중"으로 변경됩니다.
- 개발이 완료되면, 테스트 단계로 이동하고 "테스트 중" 상태로 변경됩니다.

3. 출시 준비:

- 테스트가 완료되면 상태가 "출시 대기"로 변경됩니다. 이 시점에 ProductVersion 엔티티가 업데이트되며, 출시 버전이 기록됩니다.

4. 출시:

- 상품이 실제로 출시되면, 상태가 "출시 중"으로 변경되고, 출시 이벤트가 기록됩니다. LaunchDate가 설정됩니다.

5. 판매 종료 및 종료 후 관리:

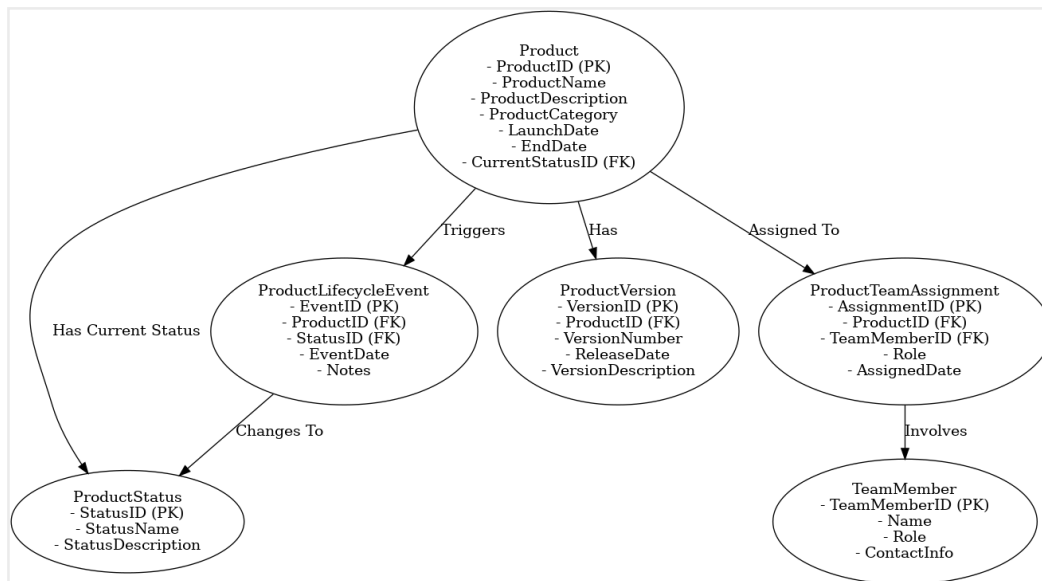
- 상품의 수명이 다했거나 시장에서 철수하게 되면, 상태가 "판매 종료"로 변경되고 EndDate가 설정됩니다.
- 이후에도 필요한 경우 ProductLifecycleEvent를 통해 상태 변경이나 기타 이벤트를 기록할 수 있습니다.

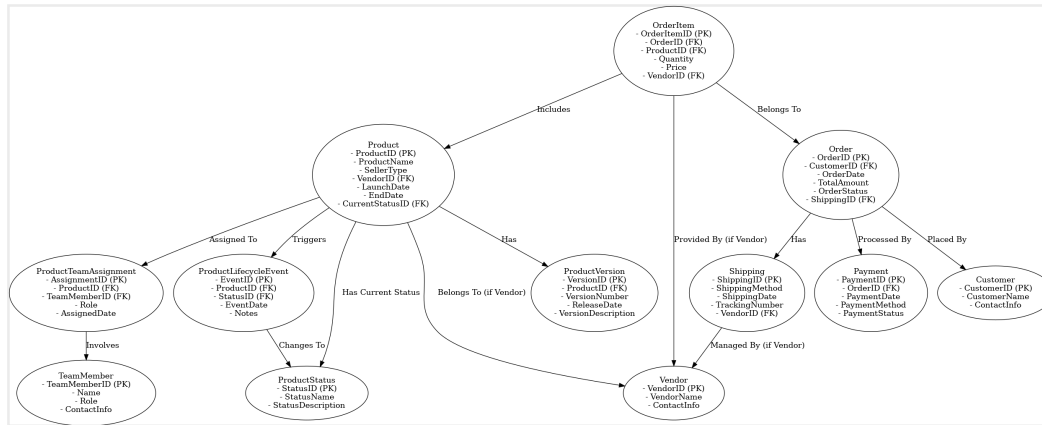
4. 설계된 모델의 활용

이 데이터 모델은 상품의 생애주기 전반에 걸쳐 상세한 관리가 가능하게 설계되었습니다. 이를 통해 다음과 같은 이점을 얻을 수 있습니다:

- **투명한 상태 추적:** 상품의 각 상태 변화를 기록함으로써 전체 라이프 사이클을 명확하게 추적할 수 있습니다.
- **효과적인 협업:** 팀 멤버 할당과 역할 관리를 통해 협업 효율성을 극대화할 수 있습니다.
- **신속한 문제 해결:** 특정 상태에서 문제가 발생했을 때, 과거 이벤트와 상태를 분석하여 원인을 신속히 파악할 수 있습니다.

이제 이 모델을 시각적으로 표현해 드리겠습니다.





문5) 특정 상품에 대한 옵션별 관리 방안에 대한 모델을 제시하시오

상품 옵션별 관리 방안은 상품의 다양한 변형(예: 색상, 사이즈, 재질 등)을 체계적으로 관리할 수 있는 데이터 모델을 설계하는 것이 핵심입니다. 이를 통해 고객이 원하는 옵션을 선택할 수 있도록 하고, 재고 및 가격 관리 등을 효율적으로 처리할 수 있습니다.

1. 상품 옵션 관리 모델 설계

1.1. 주요 엔티티

1. Product (상품)

- ProductID (PK)
- ProductName
- ProductDescription
- ProductCategory
- BasePrice (기본 가격)

2. OptionCategory (옵션 카테고리)

- OptionCategoryID (PK)
- OptionCategoryName (예: 색상, 사이즈, 재질)
- ProductID (FK)

3. OptionValue (옵션 값)

- OptionValueID (PK)
- OptionCategoryID (FK)
- OptionValueName (예: 빨강, 파랑, S, M, L)

4. ProductOption (상품 옵션)

- ProductOptionID (PK)
- ProductID (FK)
- OptionValueID (FK)
- AdditionalPrice (옵션에 따른 추가 가격)
- SKU (재고 관리용 고유 식별자)
- QuantityAvailable (재고 수량)

5. ProductInventory (상품 재고)

- InventoryID (PK)
- ProductOptionID (FK)
- WarehouseLocation
- QuantityOnHand
- QuantityReserved

- QuantityAvailable

1.2. 모델 설명

- **OptionCategory**와 **OptionValue**: 상품 옵션을 구조화하여 관리할 수 있습니다. 예를 들어, OptionCategory는 "색상"과 "사이즈"를 정의하고, 각각의 OptionValue는 "빨강", "파랑", "S", "M" 등으로 정의됩니다.
- **ProductOption**: 특정 상품의 구체적인 옵션을 정의하며, 이 옵션이 적용된 경우의 추가 가격, SKU, 그리고 재고 정보를 관리합니다.
- **ProductInventory**: 상품 옵션별로 재고를 관리하기 위해 ProductInventory 엔티티를 도입합니다. 이는 옵션별로 재고 수량을 효율적으로 관리하고, 재고 부족 등을 미리 예측할 수 있도록 합니다.

2. 설계 사유

2.1. 유연한 옵션 관리

- **옵션 카테고리 및 값의 분리**: OptionCategory와 OptionValue를 분리하여 상품 옵션을 유연하게 관리할 수 있습니다. 이를 통해 다양한 조합의 옵션을 체계적으로 관리할 수 있습니다.
- **다양한 옵션 조합**: ProductOption 엔티티는 상품별로 다양한 옵션 조합을 허용하여 고객이 원하는 옵션을 쉽게 선택할 수 있도록 합니다.

2.2. 재고 관리 최적화

- **옵션별 재고 관리**: ProductInventory 엔티티를 통해 옵션별 재고를 세밀하게 관리할 수 있습니다. 이를 통해 특정 옵션이 재고가 부족한 경우 빠르게 대응할 수 있습니다.
- **SKU 관리**: SKU(Stock Keeping Unit)를 통해 옵션별 상품을 고유하게 식별하고, 효율적인 재고 관리를 할 수 있습니다.

2.3. 가격 정책 관리

- **추가 가격 관리**: ProductOption 엔티티에서 옵션별로 추가 가격을 설정할 수 있어, 고객이 선택한 옵션에 따라 최종 가격이 자동으로 계산됩니다.
- **기본 가격과의 통합**: 상품의 기본 가격에 옵션에 따른 추가 가격을 합산하여 최종 판매 가격을 계산합니다.

3. 옵션 관리 프로세스

1. 옵션 정의:

- 상품에 적용할 옵션 카테고리 및 옵션 값을 정의합니다. 예: 색상(빨강, 파랑), 사이즈(S, M, L).

2. 옵션 조합 생성:

- 각 상품에 대해 선택 가능한 옵션 조합을 생성합니다. 예: 색상-빨강, 사이즈-M.

3. 재고 및 가격 관리:

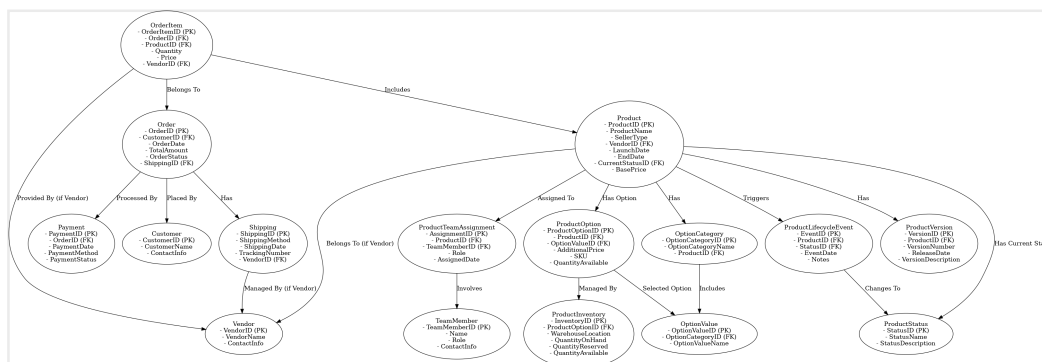
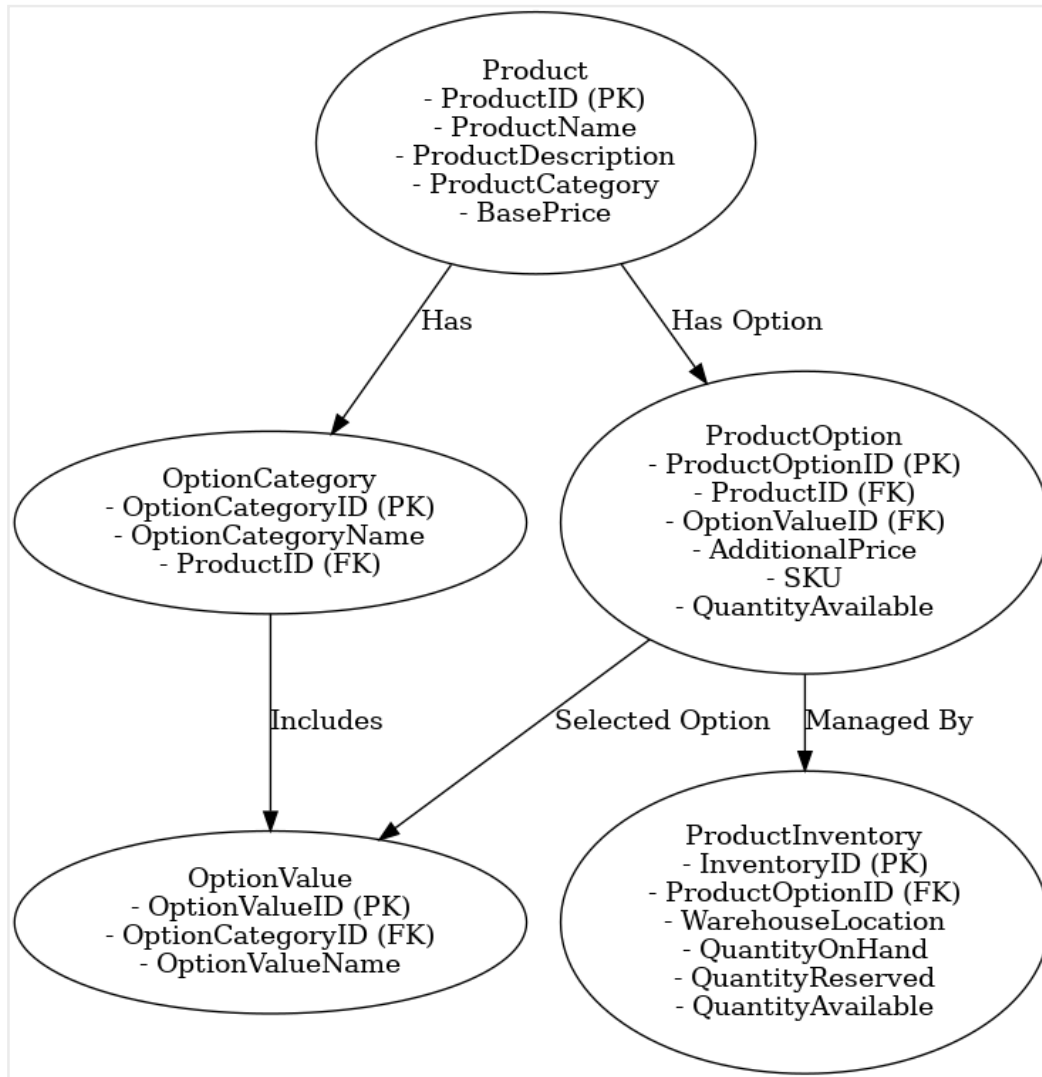
- 옵션 조합별로 재고 수량을 입력하고, 옵션에 따른 추가 가격을 설정합니다.
- SKU를 생성하여 옵션 조합을 고유하게 식별할 수 있도록 합니다.

4. 판매 및 재고 추적:

- 고객이 옵션을 선택하여 상품을 구매할 때, 선택한 옵션에 대한 재고가 감소하고, 추가 가격이 최종 가격에 반영됩니다.
- 옵션별로 재고를 지속적으로 모니터링하여 재고 부족 시 사전 대응할 수 있습니다.

4. ERD 예시

이제 위에서 제시한 상품 옵션 관리 모델을 포함한 ERD를 시각적으로 표현하겠습니다.



첫번째 질문: 예약 위주의 상품 관련 데이터 모델에 실시간 판매 가능한 상품 유형 추가 방안

설계 제안 및 사유:

기존 예약 상품 중심의 데이터 모델에 실시간 판매 가능한 상품을 추가하기 위해서는 Product 엔티티에 판매 유형(SaleType)을 정의하고, 예약과 실시간 판매 유형을 구분하는 속성을 추가하는 방안을 제안합니다. 이 속성을 통해 예약이 필요한 상품과 실시간으로 구매 가능한 상품을 명확하게 구분할 수 있으며, 이를 통해 비즈니스의 유연성을 확보할 수 있습니다.

주요 엔티티:

- **Product (상품)**
 - ProductID: 고유 식별자
 - ProductName: 상품명

- SaleType: 판매 유형 (예약, 실시간)
- Price: 가격
- Stock: 재고 (실시간 판매 상품에 해당)
- **Reservation (예약)**
 - ReservationID: 고유 식별자
 - ProductID: 연결된 상품 ID
 - ReservationDate: 예약 일자
 - CustomerID: 고객 ID
 - Quantity: 예약 수량
- **RealTimeSale (실시간 판매)**
 - SaleID: 고유 식별자
 - ProductID: 연결된 상품 ID
 - SaleDate: 판매 일자
 - CustomerID: 고객 ID
 - Quantity: 판매 수량

ERD:

- **Product**가 중심에 위치하고, SaleType 속성을 통해 예약과 실시간 판매 유형을 구분합니다.
- **Reservation**은 Product와 1 관계를 가지며, 예약 정보를 관리합니다.
- **RealTimeSale**은 Product와 1 관계를 가지며, 실시간 판매 정보를 관리합니다.

두번째 질문: 입점 업체의 상품 주문 관련 서비스 모델 설계

설계 제안 및 사유:

입점 업체에 대한 상품 주문을 포함하기 위해 새로운 Vendor 엔티티와 VendorProduct 엔티티를 추가하고, 주문 데이터 모델을 확장합니다. 이렇게 함으로써 자사 상품과 입점 업체 상품을 동일한 구조로 관리할 수 있으며, 각각의 주문 처리에 필요한 데이터 관리가 가능합니다.

주요 엔티티:

- **Vendor (입점 업체)**
 - VendorID: 고유 식별자
 - VendorName: 업체명
 - ContactInfo: 연락처 정보
- **VendorProduct (입점 업체 상품)**
 - VendorProductID: 고유 식별자
 - VendorID: 연결된 입점 업체 ID
 - ProductName: 상품명
 - Price: 가격
 - Stock: 재고
- **Order (주문)**
 - OrderID: 고유 식별자
 - CustomerID: 고객 ID
 - OrderDate: 주문 일자
 - OrderType: 주문 유형 (자사, 입점업체)
 - TotalAmount: 총 금액
- **OrderItem (주문 항목)**

- OrderItemID: 고유 식별자
- OrderID: 연결된 주문 ID
- ProductID (자사 상품) 또는 VendorProductID (입점 업체 상품)
- Quantity: 주문 수량
- Price: 가격

ERD:

- **Vendor**와 **VendorProduct**가 추가되었으며, **Order**와 **OrderItem**을 통해 자사 상품과 입점 업체 상품을 주문할 수 있습니다.

세번째 질문: 상품 주문 절차에 대한 프로세스 설계

설계 제안 및 사유:

상품 주문 절차는 고객이 상품을 선택하고, 주문을 생성하며, 결제를 진행하고, 이후 주문 상태를 관리하는 절차로 구성됩니다. 이 프로세스를 지원하기 위해 주문 상태와 관련된 엔티티를 추가합니다.

주요 엔티티:

- **OrderStatus (주문 상태)**
 - StatusID: 고유 식별자
 - StatusName: 상태명 (예: 주문 접수, 결제 완료, 배송 중, 완료)
- **Payment (결제)**
 - PaymentID: 고유 식별자
 - OrderID: 연결된 주문 ID
 - PaymentDate: 결제 일자
 - Amount: 결제 금액
 - PaymentMethod: 결제 방법

ERD:

- **Order**와 **OrderStatus** 간의 1 관계를 통해 주문 상태를 관리합니다.
- **Payment** 엔티티는 주문과 1:1 관계로 결제 정보를 관리합니다.

네번째 질문: 상품 출시에 대한 라이프 사이클 관리 모델

설계 제안 및 사유:

상품 출시에 대한 라이프 사이클을 관리하기 위해 ProductLifecycle 엔티티를 추가하여 상품의 상태 변화 (출시, 판매 중, 중지 등)를 기록하고 관리합니다. 이를 통해 각 상품의 상태를 추적하고, 적절한 조치를 취할 수 있습니다.

주요 엔티티:

- **ProductLifecycle (상품 라이프 사이클)**
 - LifecycleID: 고유 식별자
 - ProductID: 연결된 상품 ID
 - Status: 상태 (출시, 판매 중, 중지 등)
 - StartDate: 상태 시작 일자
 - EndDate: 상태 종료 일자 (중지일 경우)

ERD:

- **Product**와 **ProductLifecycle** 간의 1 관계를 통해 상품의 상태 변화를 관리합니다.

다섯번째 질문: 특정 상품에 대한 옵션 별 관리 방안 모델

설계 제안 및 사유:

특정 상품에 대한 옵션을 관리하기 위해 ProductOption 엔티티와 OptionValue 엔티티를 추

가하여 각 상품의 옵션과 그 값을 체계적으로 관리합니다.

주요 엔티티:

- **ProductOption (상품 옵션)**
 - OptionID: 고유 식별자
 - ProductID: 연결된 상품 ID
 - OptionName: 옵션명 (예: 색상, 사이즈)
- **OptionValue (옵션 값)**
 - ValueID: 고유 식별자
 - OptionID: 연결된 옵션 ID
 - ValueName: 값명 (예: 빨강, 파랑, S, M, L)

ERD:

- **Product**와 **ProductOption** 간의 1
관계, **ProductOption**과 **OptionValue** 간의 1
관계를 통해 상품 옵션을 관리합니다.

결론:

이와 같이 각 단계에서 필요한 데이터 모델을 설계하고, 이를 바탕으로 유연하고 확장 가능한 비즈니스 시스템을 구축할 수 있습니다. 각각의 ERD를 통해 데이터 관계를 시각화하고, 비즈니스 로직을 명확하게 이해할 수 있습니다.

첫 번째 질문: 실시간 판매 가능한 상품 유형 추가

ERD 설명

- **Product** 엔티티는 ProductID, ProductName, SaleType, Price, Stock 속성을 가집니다.
- **Reservation** 엔티티는 ReservationID, ProductID, ReservationDate, CustomerID, Quantity 속성을 가집니다. ProductID는 **Product** 엔티티와 FK로 연결됩니다.
- **RealTimeSale** 엔티티는 SaleID, ProductID, SaleDate, CustomerID, Quantity 속성을 가집니다. ProductID는 **Product** 엔티티와 FK로 연결됩니다.

두 번째 질문: 입점 업체의 상품 주문 관련 서비스 모델 설계

ERD 설명

- **Vendor** 엔티티는 VendorID, VendorName, ContactInfo 속성을 가집니다.
- **VendorProduct** 엔티티는 VendorProductID, VendorID, ProductName, Price, Stock 속성을 가집니다. VendorID는 **Vendor** 엔티티와 FK로 연결됩니다.
- **Order** 엔티티는 OrderID, CustomerID, OrderDate, OrderType, TotalAmount 속성을 가집니다.
- **OrderItem** 엔티티는 OrderItemID, OrderID, ProductID, VendorProductID, Quantity, Price 속성을 가집니다. OrderID는 **Order** 엔티티와 FK로 연결되고, ProductID와 VendorProductID는 각각 **Product** 및 **VendorProduct** 엔티티와 FK로 연결됩니다.

세 번째 질문: 상품 주문 절차에 대한 프로세스 설계

ERD 설명

- **Order** 엔티티는 위와 동일합니다.
- **OrderStatus** 엔티티는 StatusID, StatusName 속성을 가집니다.
- **Payment** 엔티티는 PaymentID, OrderID, PaymentDate, Amount,

PaymentMethod 속성을 가집니다. OrderID는 **Order** 엔티티와 FK로 연결됩니다.
네 번째 질문: 상품 출시에 대한 라이프 사이클 관리 모델

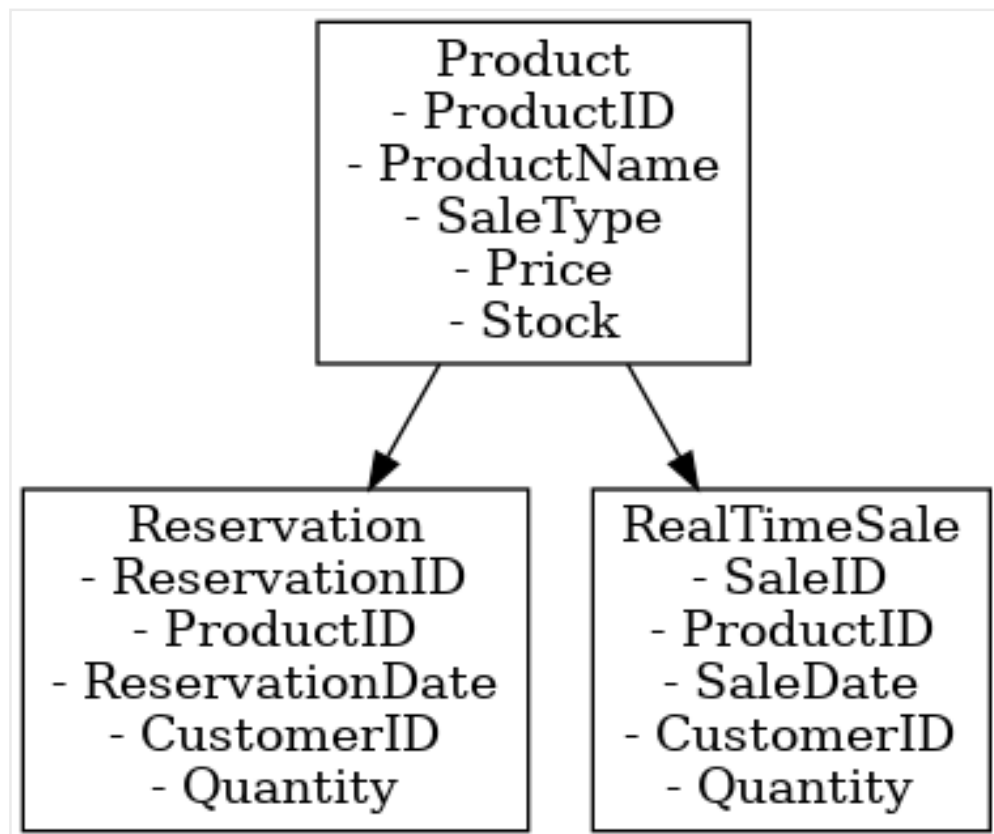
ERD 설명

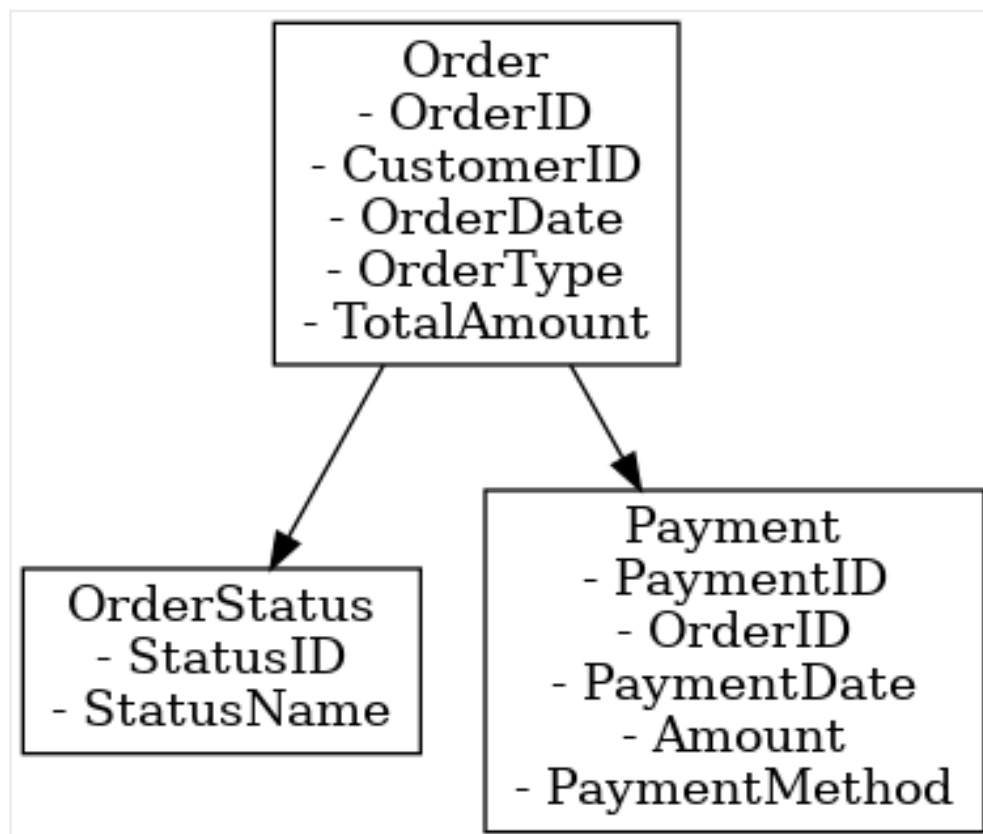
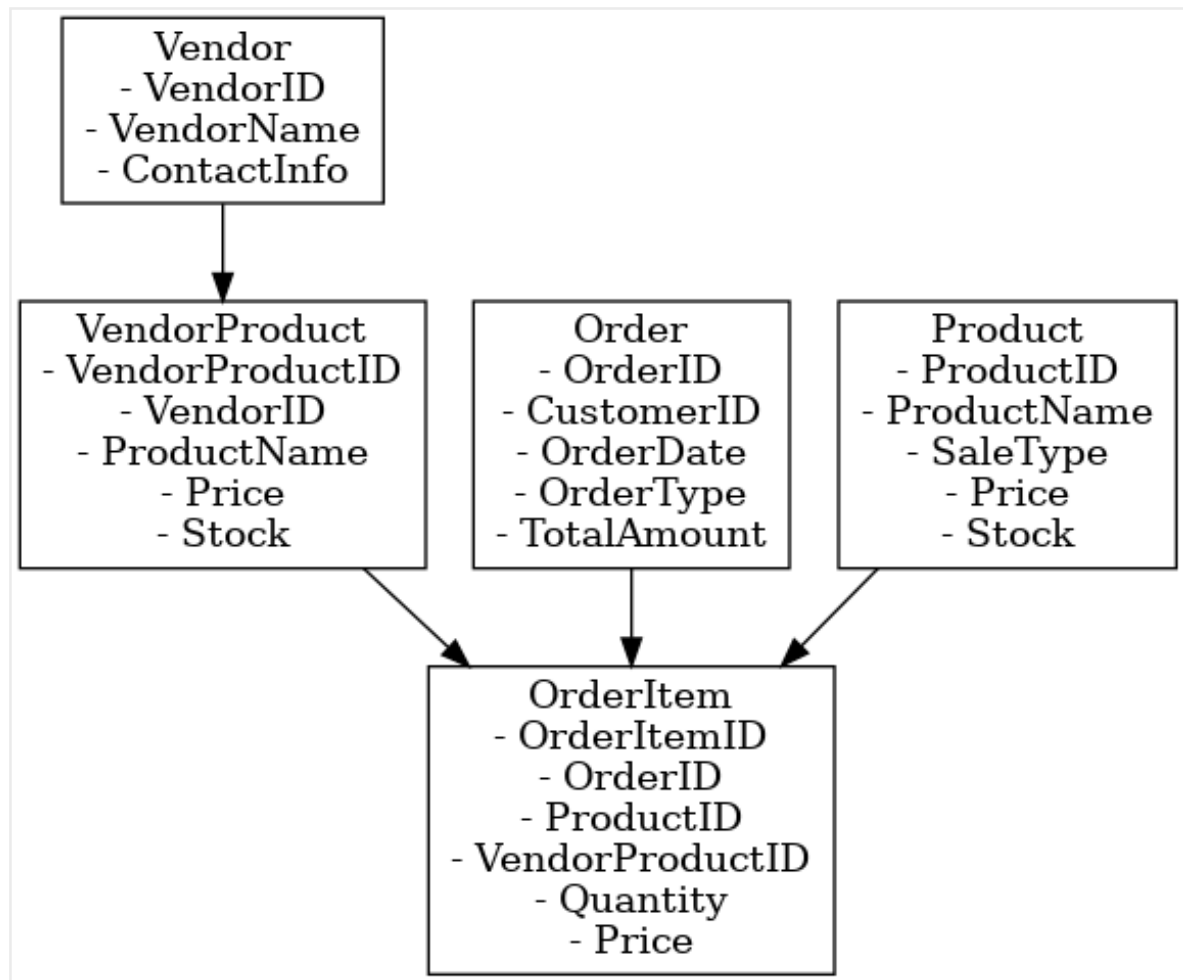
- **Product** 엔티티는 위와 동일합니다.
- **ProductLifecycle** 엔티티는 LifecycleID, ProductID, Status, StartDate, EndDate 속성을 가집니다. ProductID는 **Product** 엔티티와 FK로 연결됩니다.

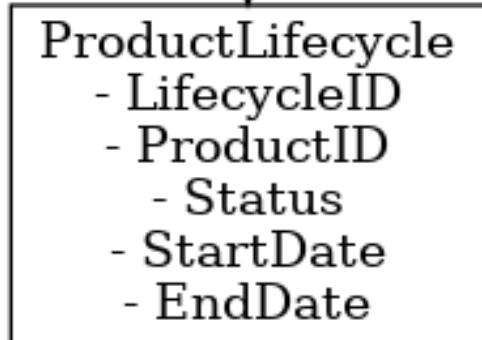
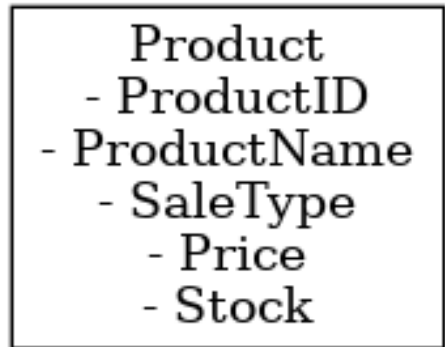
다섯 번째 질문: 특정 상품에 대한 옵션 별 관리 방안

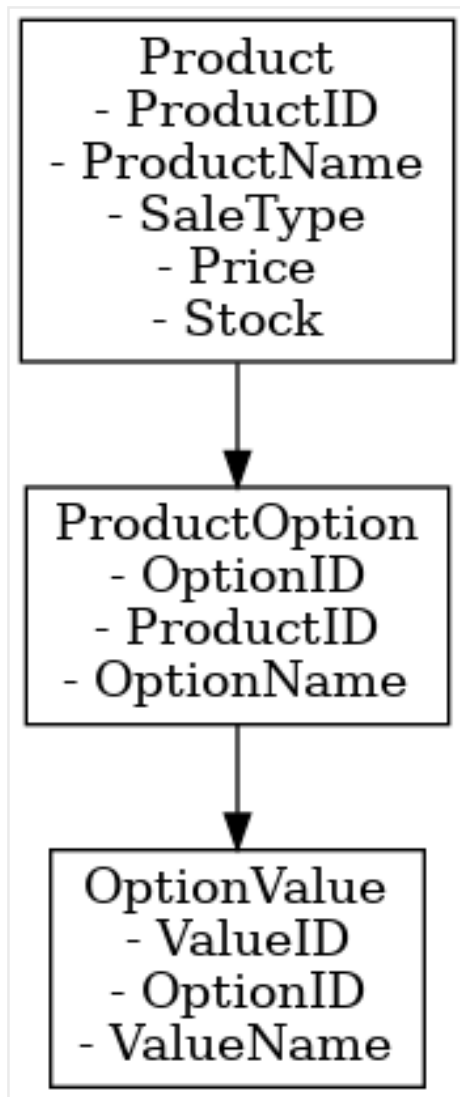
ERD 설명

- **Product** 엔티티는 위와 동일합니다.
- **ProductOption** 엔티티는 OptionID, ProductID, OptionName 속성을 가집니다. ProductID는 **Product** 엔티티와 FK로 연결됩니다.
- **OptionValue** 엔티티는 ValueID, OptionID, ValueName 속성을 가집니다. OptionID는 **ProductOption** 엔티티와 FK로 연결됩니다.









[주제3]

예약 처리 프로세스와 데이터 모델에서 발생하고 있는 동시성 이슈의 해결 방안을 찾으려고 해

문1) 다수의 사용자가 다수의 인벤토리 예약을 하기 위한 데이터 모델의 문제점을 확인하고, 해결을 위한

방안을 제시해줘

구현방법은 예약 처리 프로세스는 Async 한 방식으로 구현 되어야 해

동시성 이슈는 여러 사용자가 동시에 동일한 리소스를 접근하고 수정할 때 발생할 수 있으며, 특히 예약 처리와 같은 상황에서 인벤토리의 상태가 일관되게 유지되도록 하는 것이 중요합니다. 아래는 이러한 동시성 문제를 해결하기 위한 방안입니다.

1. 문제점 확인

1.1 잠금 메커니즘 부족

다수의 사용자가 동시에 동일한 인벤토리 항목을 예약하려고 할 때, 잠금 메커니즘이 없으면 데이터 불일치가 발생할 수 있습니다. 예를 들어, 두 사용자가 동시에 동일한 좌석을 예약하려고 할 때, 시스템이 동시에 두 사용자의 요청을 처리하면 좌석이 중복 예약될 수 있습니다.

1.2 비원자성 연산

예약 처리 프로세스가 여러 단계로 나뉘어 있을 때(예: 재고 확인 -> 예약 확정 -> 결제), 이들 연

산이 원자적(atomic)으로 처리되지 않으면, 중간 상태에서 동시성 이슈가 발생할 수 있습니다.

2. 해결 방안

2.1 잠금 메커니즘 도입

- **Optimistic Locking:** 이 방식에서는 데이터를 업데이트할 때 충돌을 감지하고, 충돌이 발생하면 트랜잭션을 롤백합니다. 이를 위해 버전 번호를 사용하여 데이터가 업데이트되기 전에 다른 트랜잭션에 의해 변경되지 않았는지 확인할 수 있습니다.
- **Pessimistic Locking:** 데이터에 대해 잠금을 획득한 후 다른 트랜잭션이 해당 데이터에 접근하지 못하도록 합니다. 이 방식은 동시성 이슈를 원천적으로 차단할 수 있지만, 성능 저하를 유발할 수 있습니다.

2.2 데이터베이스 수준에서 트랜잭션 관리

- **ACID 트랜잭션 사용:** 예약 처리 프로세스 전체를 트랜잭션으로 감싸서 원자성을 보장합니다. 즉, 재고 확인, 예약 확정, 결제 모두가 하나의 트랜잭션 내에서 처리되어야 하며, 이 트랜잭션이 완료되지 않으면 데이터베이스의 상태가 변경되지 않도록 합니다.

2.3 예약 대기열(Queue) 시스템 사용

- **메시지 큐(Message Queue):** 사용자가 예약 요청을 보낼 때, 이를 바로 처리하지 않고 메시지 큐에 넣습니다. 백엔드에서 예약 프로세스를 비동기적으로 처리하고, 한 번에 하나의 요청만 처리할 수 있도록 합니다. 이렇게 하면 동시성 문제를 피하면서 순차적으로 예약을 처리할 수 있습니다.

2.4 재고 감소 시 원자적 연산 사용

- **데이터베이스의 원자적 연산(Atomic Operation):** 예약 시 인벤토리 수량을 감소시키는 작업이 하나의 원자적 연산으로 이루어져야 합니다. 예를 들어, SQL에서는 `UPDATE inventory SET quantity = quantity - 1 WHERE id = ? AND quantity > 0` 같은 쿼리를 사용하여 동시성 문제를 해결할 수 있습니다.

2.5 캐시 사용 고려

- **캐시와 Redis 같은 In-Memory 데이터 저장소:** 재고 확인과 같은 빈번한 읽기 작업에 대해 캐시를 사용하여 데이터베이스의 부하를 줄일 수 있습니다. 또한 Redis의 INCR 같은 원자적 연산을 사용하여 동시성 이슈를 해결할 수도 있습니다.

3. Async 예약 처리 프로세스 구현

Async 방식으로 예약 프로세스를 구현하는 방법은 다음과 같습니다:

1. **비동기 예약 요청 수신:** 예약 요청을 수신할 때마다 비동기 작업을 생성합니다.
2. **잠금 메커니즘 사용:** 비동기 작업 내에서 Optimistic Locking 또는 Pessimistic Locking을 사용하여 동시성 문제를 해결합니다.
3. **메시지 큐와 함께 사용:** 예약 요청을 큐에 넣고 비동기 작업에서 큐의 작업을 하나씩 처리합니다.
4. **결과 반환:** 예약 처리 결과를 비동기적으로 사용자에게 반환합니다. 예를 들어 예약이 완료되면, 완료 상태를 웹소켓을 통해 실시간으로 전달할 수 있습니다.

4. 데이터 모델 설계

4.1 인벤토리 테이블

sql

코드 복사

```
CREATE TABLE inventory (  
  id BIGINT PRIMARY KEY,  
  item_name VARCHAR(255),  
  quantity INT,  
  version INT DEFAULT 0
```

);

- quantity: 현재 남아있는 재고 수량
- version: Optimistic Locking을 위한 버전 번호

4.2 예약 테이블

sql

코드 복사

```
CREATE TABLE reservation (  
    id BIGINT PRIMARY KEY,  
    inventory_id BIGINT,  
    user_id BIGINT,  
    reserved_at TIMESTAMP,  
    status VARCHAR(50)  
);
```

- inventory_id: 예약된 인벤토리 항목의 ID
- user_id: 예약을 시도한 사용자 ID
- status: 예약 상태 (예: 'PENDING', 'CONFIRMED', 'CANCELLED')

이러한 방법을 통해 다수의 사용자가 동시에 예약을 시도하더라도 동시성 이슈를 해결하고 시스템의 일관성을 유지할 수 있습니다.

문2) 다수의 사용자가 하나의 인벤토리 예약을 하려고 한다, 데이터 모델의 문제점을 확인하고 해결을 위한

방안을 제시하시오.

<예약 조건>

- 예약 처리 프로세스는 Async 한 방식으로 구현 되어야 한다

다수의 사용자가 하나의 인벤토리 항목을 예약하려고 할 때 발생할 수 있는 데이터 모델의 문제점과 이를 해결하기 위한 방안을 아래에 제시합니다.

1. 문제점 확인

1. 경합 조건(Race Condition) 문제:

- 다수의 사용자가 동시에 동일한 인벤토리 항목을 예약하려고 할 때, 경합 조건이 발생할 수 있습니다. 이는 동시다발적으로 처리되는 요청들이 인벤토리의 상태를 일관성 있게 유지하지 못하게 만들 수 있으며, 중복 예약이나 초과 예약이 발생할 수 있습니다.

2. 데이터 무결성 문제:

- 인벤토리 수량을 확인하고 업데이트하는 과정이 비원자적으로 이루어지면, 다른 프로세스가 동시에 인벤토리 데이터를 읽고 업데이트할 수 있어 데이터 무결성이 훼손될 수 있습니다.

3. 비동기 처리로 인한 예측 불가능성:

- Async 방식으로 구현된 예약 처리 과정에서는 예약 요청이 순차적으로 처리되지 않을 수 있습니다. 이러한 비동기 처리 방식에서 동시성 문제를 적절히 관리하지 않으면 비즈니스 로직이 의도한 대로 동작하지 않을 가능성이 높아집니다.

2. 해결 방안

2.1 데이터베이스 수준에서 잠금 메커니즘 적용

1. Optimistic Locking:

- 인벤토리 항목에 version 필드를 추가하고, 예약을 시도할 때마다 현재 version 값을 확인하여 업데이트합니다. 만약 다른 요청이 먼저 해당 인벤토리 항목을 업데이트했다면, 트랜잭션을 롤백하고 다시 시도합니다.

2. sql코드 복사

```
UPDATE inventory
```

```
3. SET quantity = quantity - 1, version = version + 1
```

```
4. WHERE id = ? AND version = ? AND quantity > 0;
```

```
5.
```

- 위의 쿼리는 인벤토리 수량을 줄이면서 버전 번호를 증가시킵니다. 만약 version이 예상과 다르다면, 다른 트랜잭션이 먼저 인벤토리를 업데이트한 것이므로 현재 트랜잭션은 실패하게 됩니다.

6. Pessimistic Locking:

- 특정 인벤토리 항목을 예약할 때, 해당 행에 대한 배타적 잠금을 획득하여 다른 트랜잭션이 접근하지 못하게 할 수 있습니다. 이렇게 하면 경합 조건을 피할 수 있지만, 성능 저하가 발생할 수 있습니다.

7. sql코드 복사

```
SELECT quantity
```

```
8. FROM inventory
```

```
9. WHERE id = ?
```

```
10. FOR UPDATE;
```

```
11.
```

- 이 쿼리는 해당 인벤토리 항목을 잠그고 다른 트랜잭션이 접근하지 못하도록 합니다.

2.2 재고 관리 및 예약 요청 처리 모델 개선

1. 큐 기반 처리 시스템 도입:

- 예약 요청을 큐에 넣고 비동기적으로 처리하는 방식을 사용합니다. 각 요청은 순차적으로 처리되며, 경합 조건을 피할 수 있습니다. 이 방식은 특정 요청이 처리되는 동안 다른 요청들이 대기하게 되어, 하나의 인벤토리에 대한 동시 접근이 발생하지 않습니다.

2. 원자적 연산 활용:

- Redis와 같은 In-Memory 데이터 저장소에서 원자적 연산을 활용하여 동시성 문제를 해결할 수 있습니다. 예를 들어, Redis의 DECR 명령어는 원자적으로 작동하여 경합 조건을 피할 수 있습니다.

3. lua코드 복사

```
local quantity = redis.call('GET', KEYS[1])
```

```
4. if tonumber(quantity) > 0 then
```

```
5.   redis.call('DECR', KEYS[1])
```

```
6.   return true
```

```
7. else
```

```
8.   return false
```

```
9. end
```

```
10.
```

- 위의 Lua 스크립트는 Redis에서 동작하며, 재고가 남아있는 경우에만 원자적으로 수량을 줄입니다.

2.3 트랜잭션 사용

- **ACID 트랜잭션을 통한 일관성 보장:**

- 재고 확인과 업데이트, 예약 확정 등의 모든 작업을 하나의 트랜잭션 내에서 처리하여 원자성을 보장합니다. 이를 통해 모든 작업이 성공해야만 최종적으로 커밋되어 데이터의 일관성을 유지할 수 있습니다.

3. 데이터 모델 수정 제안

1. Inventory 테이블:

- version 필드를 추가하여 Optimistic Locking을 구현할 수 있도록 합니다.

2. sql코드 복사

```
CREATE TABLE inventory (  
3.   id BIGINT PRIMARY KEY,  
4.   item_name VARCHAR(255),  
5.   quantity INT,  
6.   version INT DEFAULT 0  
7. );  
8.
```

9. Reservation 테이블:

- 예약된 인벤토리와 관련된 정보를 기록하는 테이블입니다.

10. sql코드 복사

```
CREATE TABLE reservation (  
11.   id BIGINT PRIMARY KEY,  
12.   inventory_id BIGINT,  
13.   user_id BIGINT,  
14.   reserved_at TIMESTAMP,  
15.   status VARCHAR(50)  
16. );  
17.
```

18. 예약 처리 프로세스:

- 예약 요청을 큐에 넣고, 비동기적으로 하나씩 처리하도록 합니다.
- 비동기 작업에서는 잠금 메커니즘을 사용하여 동일 인벤토리에 대한 동시 접근을 관리합니다.

결론

Async 방식의 예약 처리 프로세스에서 동시성 이슈를 해결하기 위해서는 데이터베이스의 잠금 메커니즘(Optimistic 또는 Pessimistic Locking), 큐 기반 처리 시스템, 원자적 연산, 그리고 트랜잭션을 적절히 활용하는 것이 중요합니다. 이러한 방법들을 통해 다수의 사용자가 동일한 인

벤토리 항목을 예약하려고 할 때 발생할 수 있는 문제들을 효과적으로 해결할 수 있습니다.

문3) 예약된 서비스에 대해 취소 프로세스를 구성하시오

예약된 서비스의 취소 프로세스를 구성할 때는 기존의 예약 데이터를 일관성 있게 관리하고, 인벤토리의 상태를 정확하게 업데이트하는 것이 중요합니다. 취소 프로세스를 잘못 설계하면 인벤토리 상태가 불일치하거나, 사용자에게 잘못된 정보가 제공될 수 있습니다. 아래는 Async 방식으로 구현된 예약 시스템에서의 취소 프로세스를 설명합니다.

1. 취소 프로세스 흐름

1. **취소 요청 수신:** 사용자가 예약 취소 요청을 보냅니다.
2. **예약 상태 확인:** 시스템은 해당 예약이 유효한지, 이미 취소된 상태는 아닌지 확인합니다.
3. **인벤토리 업데이트:** 취소된 예약에 해당하는 인벤토리 수량을 증가시킵니다.
4. **예약 상태 업데이트:** 예약 상태를 "CANCELLED"로 업데이트합니다.
5. **결과 반환:** 취소 성공 여부를 사용자에게 비동기적으로 알립니다.

2. 데이터베이스 구조 업데이트

Reservation 테이블에 취소 상태를 추가

sql

코드 복사

```
ALTER TABLE reservation ADD COLUMN cancel_reason VARCHAR(255) NULL;  
ALTER TABLE reservation ADD COLUMN canceled_at TIMESTAMP NULL;
```

- **cancel_reason:** 사용자가 예약을 취소한 이유를 저장할 수 있습니다.
- **canceled_at:** 예약이 취소된 시간을 기록합니다.

3. 취소 프로세스 상세 설계

3.1 취소 요청 처리

python

코드 복사

```
async def cancel_reservation(reservation_id: int, user_id: int, cancel_reason:  
str):
```

```
    # 예약 상태 확인
```

```
    reservation = await get_reservation_by_id(reservation_id)
```

```
    if not reservation:
```

```
        return {"error": "Reservation not found"}
```

```
    if reservation.status == "CANCELLED":
```

```
        return {"error": "Reservation already cancelled"}
```

```
    # 트랜잭션 시작
```

```
    async with db.transaction():
```

```
        # 인벤토리 수량 복구
```

```
        inventory = await get_inventory_by_id(reservation.inventory_id)
```

```
        if inventory:
```

```
            await update_inventory_quantity(inventory.id, inventory.quantity + 1)
```

```
    # 예약 상태 업데이트
```

```
    await update_reservation_status(  
        reservation_id, "CANCELLED", cancel_reason=cancel_reason
```

```
)
```

)

결과 반환

```
return {"success": True, "message": "Reservation cancelled successfully"}
```

이 함수는 비동기 방식으로 예약을 취소하며, 주요 단계는 다음과 같습니다:

1. **예약 상태 확인:** 주어진 reservation_id와 user_id를 기준으로 예약을 조회하고, 해당 예약이 이미 취소된 상태가 아닌지 확인합니다.
2. **트랜잭션 처리:** 트랜잭션 내에서 인벤토리와 예약 상태를 동시에 업데이트하여, 중간에 오류가 발생할 경우 전체 트랜잭션을 롤백합니다.
3. **인벤토리 업데이트:** 취소된 예약에 해당하는 인벤토리 수량을 증가시킵니다.
4. **예약 상태 업데이트:** 예약의 상태를 "CANCELLED"로 변경하고, 취소 이유와 취소 시간을 기록합니다.

3.2 트랜잭션 내 인벤토리와 예약 상태 업데이트

sql

코드 복사

```
UPDATE inventory
SET quantity = quantity + 1
WHERE id = ?;
```

UPDATE reservation

```
SET status = 'CANCELLED', cancel_reason = ?, canceled_at = ?
```

```
WHERE id = ? AND user_id = ?;
```

이 두 쿼리는 동일한 트랜잭션 내에서 실행되어, 인벤토리 수량이 정확히 업데이트되고 예약 상태가 변경됩니다. 이렇게 하면 취소 처리 중에 다른 트랜잭션이 발생하더라도 데이터의 일관성을 유지할 수 있습니다.

4. 사용자 알림

취소 프로세스가 완료되면 사용자에게 비동기적으로 취소 결과를 알려야 합니다. 예를 들어, 웹소켓 또는 푸시 알림을 통해 사용자가 실시간으로 취소 결과를 확인할 수 있도록 할 수 있습니다.

python

코드 복사

```
async def notify_user_of_cancellation(user_id: int, reservation_id: int):
    message = f"Your reservation {reservation_id} has been successfully cancelled."
```

```
    await send_notification(user_id, message)
```

5. 결제 취소 및 환불 처리 (선택 사항)

만약 예약에 대해 결제가 이루어졌다면, 예약 취소 시 결제 취소 및 환불 처리를 해야 합니다. 이 과정은 결제 게이트웨이와의 통신이 필요하며, 비동기적으로 처리할 수 있습니다.

python

코드 복사

```
async def process_refund(reservation_id: int, user_id: int):
```

```
    # 결제 게이트웨이와 통신하여 환불 처리
```

```
    refund_status = await payment_gateway_refund(reservation_id, user_id)
```

```
    if refund_status["success"]:
```

```
        await update_reservation_status(reservation_id, "REFUNDED")
```

```
    return refund_status
```


6. 예외 처리

예약 취소 과정에서 발생할 수 있는 오류나 예외 상황에 대한 처리가 중요합니다. 예를 들어, 인벤토리 업데이트 중 오류가 발생했을 때 트랜잭션을 롤백하고 사용자에게 적절한 오류 메시지를 반환해야 합니다.

python

코드 복사

try:

```
    result = await cancel_reservation(reservation_id, user_id, cancel_reason)
except Exception as e:
    logger.error(f"Error cancelling reservation {reservation_id}: {str(e)}")
    return {"error": "Failed to cancel reservation. Please try again later."}
```

결론

이 취소 프로세스는 다수의 사용자가 동시에 동일한 인벤토리 항목을 취소하려고 할 때 발생할 수 있는 문제를 방지하고, Async 방식으로 데이터를 안전하게 처리하도록 설계되었습니다. 인벤토리 수량을 원자적으로 복구하고, 예약 상태를 정확히 업데이트하며, 필요한 경우 결제 취소 및 환불 처리도 포함하여 사용자에게 완전한 취소 서비스를 제공합니다.

예약 처리 프로세스와 데이터 모델에서 발생하고 있는 동시성 이슈의 해결 방안을 찾으려고 해 첫번째, 다수의 사용자가 다수의 인벤토리 예약을 하기 위한 데이터 모델의 문제점을 확인하고, 해결을 위한 방안을 제시하고 ERD를 그림으로 그려줘

두번째, 다수의 사용자가 하나의 인벤토리 예약을 하려고 해, 데이터 모델의 문제점을 확인하고 해결을 위한 방안을 제시하고 ERD를 그림으로 그려줘

세번째, 예약된 서비스에 대해 취소 프로세스를 구성하고 ERD를 그림으로 그려줘

마지막으로 전체 통합된 ERD를 그림으로 그려줘

동시성 이슈를 해결하기 위한 데이터 모델링과 프로세스 설계를 단계별로 설명하고, 각 상황에 대한 ERD를 제시하겠습니다.

1. 다수의 사용자가 다수의 인벤토리 예약을 하는 경우

문제점

다수의 사용자가 동시에 여러 인벤토리를 예약하려고 할 때, 다음과 같은 문제점이 발생할 수 있습니다:

- **경합 조건:** 여러 사용자가 동시에 동일한 인벤토리를 선택하여 예약을 시도할 경우, 데이터베이스에 중복된 예약이 발생할 수 있습니다.
- **무결성 위반:** 중복된 예약이 발생하면 인벤토리의 수량이 초과되어 예약의 무결성을 위반할 수 있습니다.

해결 방안

- **락(LOCK) 사용:** 인벤토리 항목에 대한 예약이 발생할 때, 데이터베이스에서 해당 인벤토리 항목에 락을 걸어 다른 사용자가 동시에 접근하지 못하게 합니다. 락이 해제된 후에만 다음 사용자가 예약을 시도할 수 있습니다.
- **트랜잭션 처리:** 모든 예약 작업을 하나의 트랜잭션으로 처리하여, 중간에 실패하면 전체 작업이 롤백되도록 합니다.
- **Optimistic Locking:** 각 인벤토리 항목에 버전 정보를 두어, 변경 시점에서 충돌을 감지하고, 충돌 발생 시 다시 시도하도록 처리합니다.

ERD

- **Inventory:** id, name, available_quantity, version
- **Reservation:** id, user_id, inventory_id, reserved_quantity, status, created_at

2. 다수의 사용자가 하나의 인벤토리 예약을 하는 경우

문제점

여러 사용자가 동시에 하나의 인벤토리를 예약하려고 할 때, 동일한 인벤토리에 대한 예약이 중복 처리되거나, 재고가 초과 예약되는 문제가 발생할 수 있습니다.

해결 방안

- **배타적 락(Exclusive Lock) 사용:** 해당 인벤토리에 대해 락을 걸어 다른 사용자가 접근하지 못하게 합니다.
- **최신 상태 검증:** 예약 시점에서 인벤토리의 최신 상태(재고)를 검증하고, 상태가 변경된 경우 예약을 다시 시도하게 합니다.
- **큐잉 시스템:** 예약 요청을 큐에 저장하고, 하나씩 순차적으로 처리하여 동시성 문제를 방지합니다.

ERD

- **Inventory:** id, name, available_quantity, version
- **Reservation:** id, user_id, inventory_id, reserved_quantity, status, created_at

3. 예약된 서비스에 대한 취소 프로세스

구성

예약된 서비스에 대해 사용자가 취소 요청을 할 수 있어야 하며, 취소된 경우 인벤토리의 재고를 다시 업데이트해야 합니다.

해결 방안

- **예약 상태 관리:** 예약 상태를 '예약됨(Reserved)', '취소됨(Cancelled)' 등으로 관리하고, 취소 시 해당 상태를 업데이트합니다.
- **트랜잭션 처리:** 취소 시점에 인벤토리의 재고를 증가시키는 과정을 트랜잭션으로 처리하여, 취소가 실패할 경우 롤백합니다.

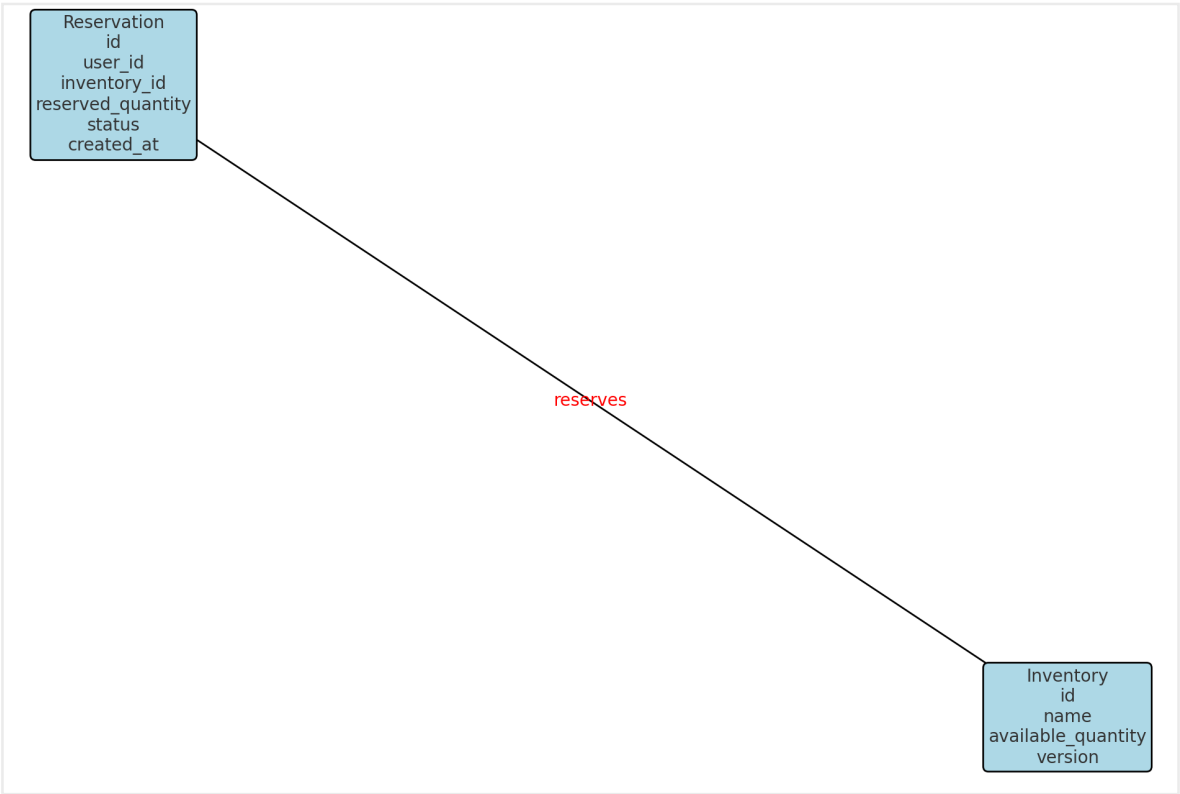
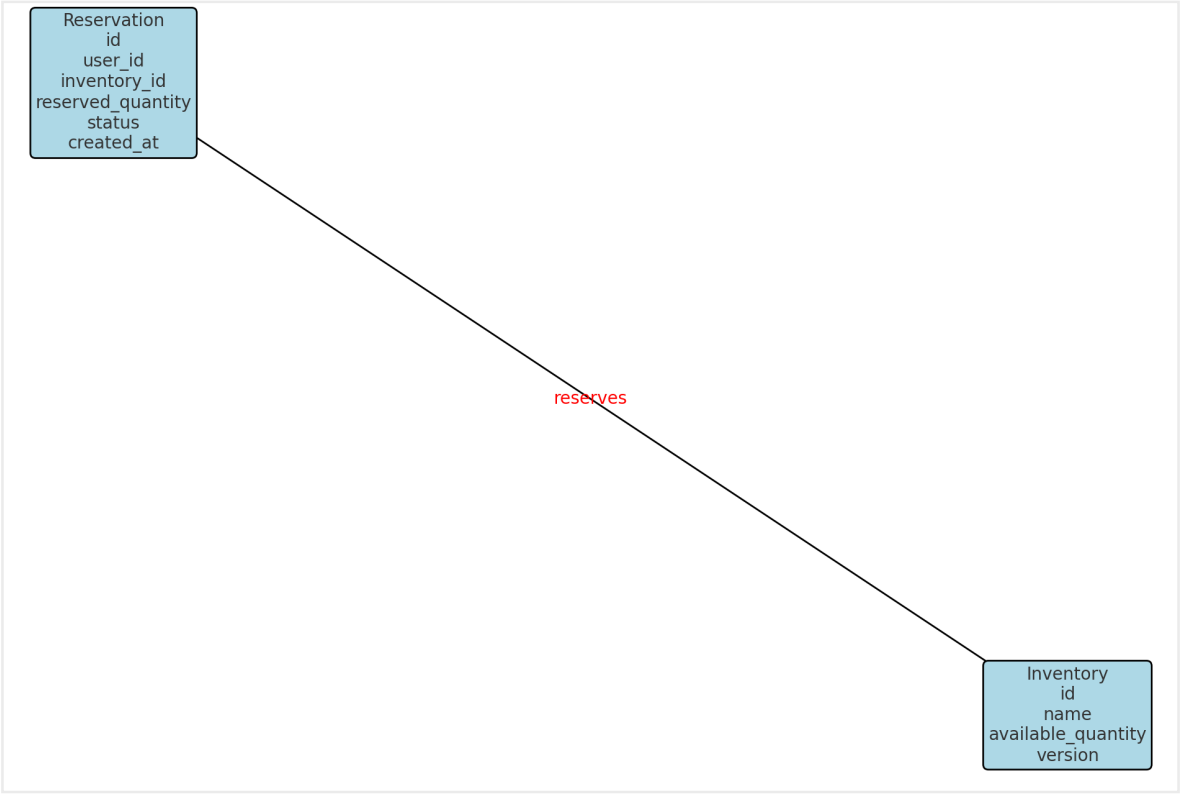
ERD

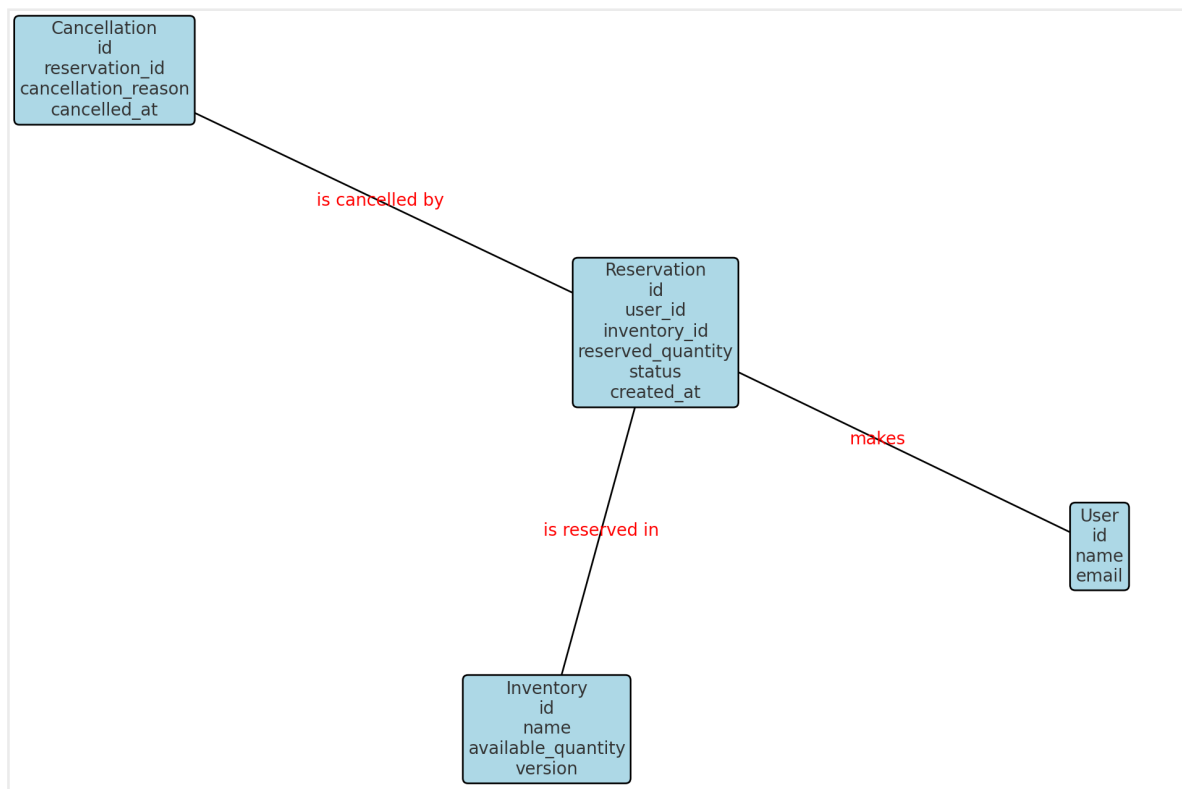
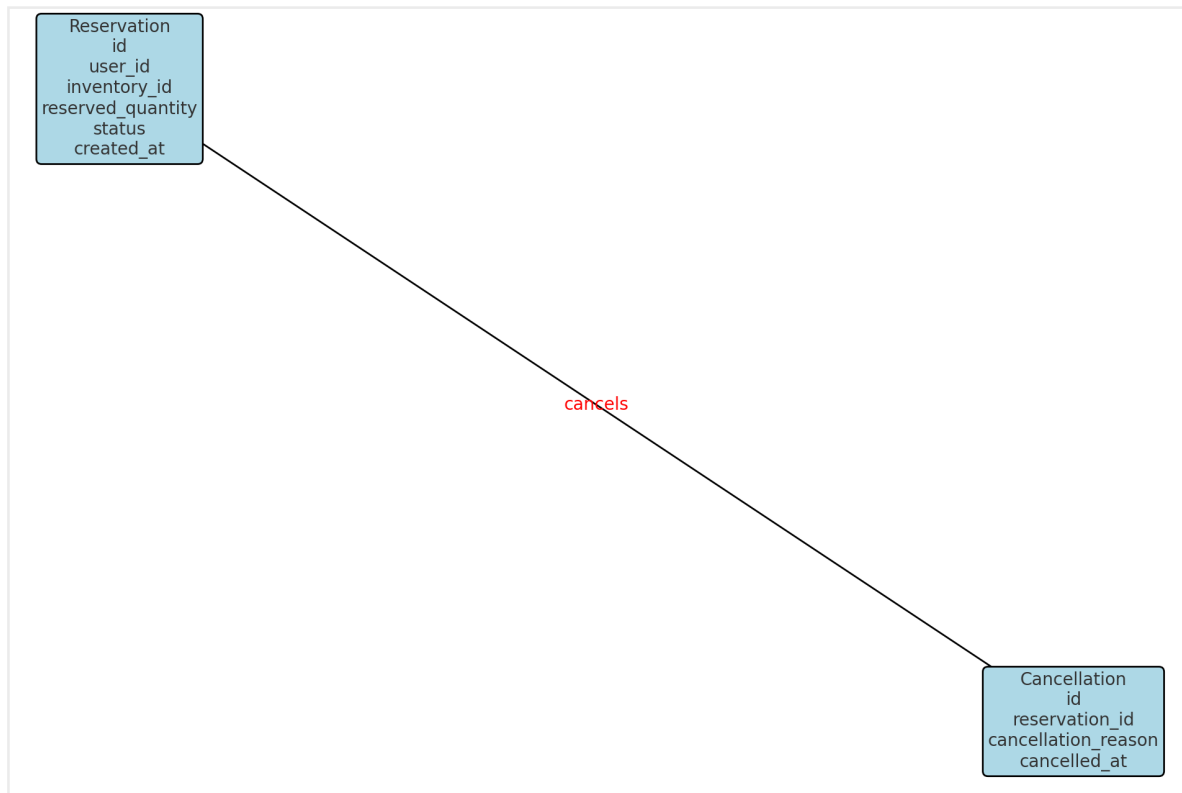
- **Inventory:** id, name, available_quantity, version
- **Reservation:** id, user_id, inventory_id, reserved_quantity, status, created_at
- **Cancellation:** id, reservation_id, cancellation_reason, cancelled_at

4. 전체 통합 ERD

통합된 ERD

- **User:** id, name, email
- **Inventory:** id, name, available_quantity, version
- **Reservation:** id, user_id, inventory_id, reserved_quantity, status, created_at
- **Cancellation:** id, reservation_id, cancellation_reason, cancelled_at





위에서 각 시나리오에 대한 ERD와 통합된 ERD를 보여드렸습니다.

1. 다수의 사용자가 다수의 인벤토리 예약을 하는 경우의 ERD: Inventory와 Reservation 엔터티 간의 관계를 표현하였으며, 다중 인벤토리에 대한 예약을 처리할 수 있도록 설계되었습니다.
2. 다수의 사용자가 동일한 인벤토리를 예약하는 경우의 ERD: 동일한 Inventory와 Reservation 구조를 사용하며, 동시성 문제를 해결하기 위한 데이터 모델을 보여줍니다.
3. 예약된 서비스에 대한 취소 프로세스의 ERD: Reservation과 Cancellation 엔터티 간

의 관계를 표현하여 예약 취소 처리를 위한 구조를 설계하였습니다.

4. **통합된 ERD:** User, Inventory, Reservation, Cancellation 엔터티를 포함한 전체 시스템의 통합된 데이터 모델을 보여줍니다.