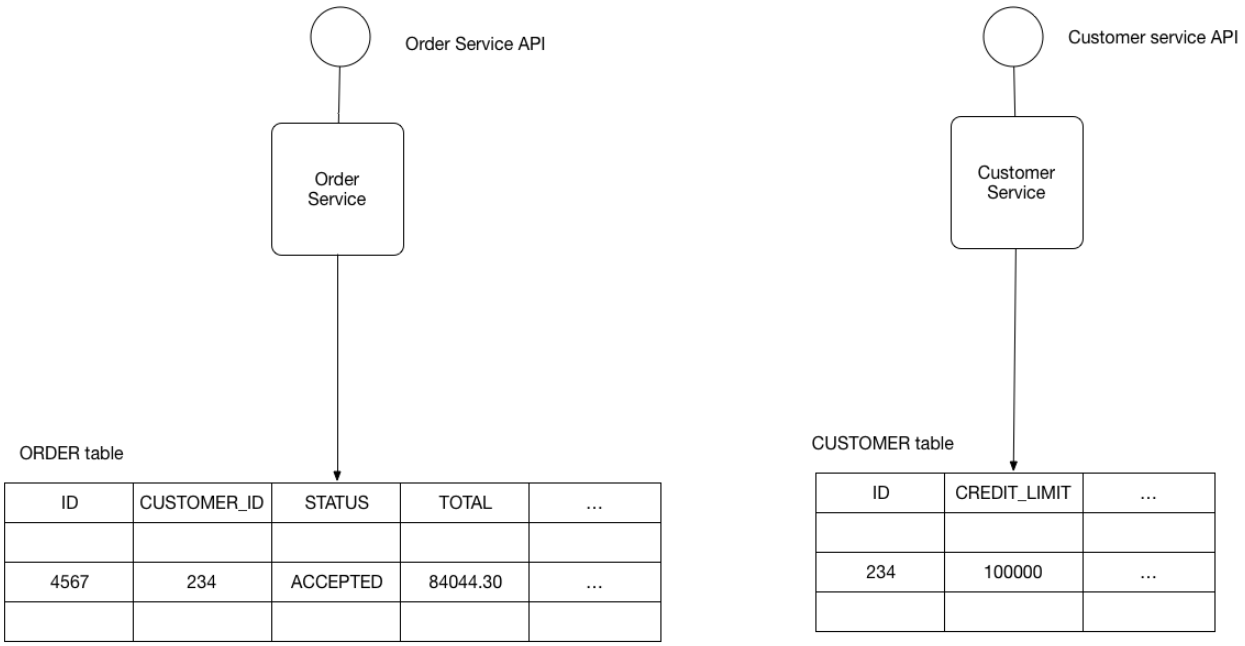
Pattern: Shared Database

version 2019.30

상황

온라인 쇼핑몰을 마이크로서비스 아키텍처를 적용하여 개발한다고 생각해봅시다. 대부분의 서비스는 특정 데이터베이스에 데이터를 유지해야합니다. 예를 들어, `Order Service`는 주문에 대한 정보를 저장하고 `Customer Service`는 고객에 대한 정보를 저장합니다.



문제

마이크로서비스 아키텍처에서는 어떤 데이터베이스 아키텍처를 사용할 것인가?

Forces

- 서비스는 느슨하게 결합되어 독립적으로 개발, 배포 및 확장될 수 있어야 합니다.
- 일부 비즈니스 트랜잭션은 여러 서비스에 걸쳐있는 invariant를 강제해야 합니다. 예를 들어, `Place Order` 유스 케이스는 새로운 주문이 고객의 신용 한도를 초과하지 않는지 확인해야 합니다. 다른 비즈니스 트랜잭션은 여러 서비스가 소유한 데이터를 업데이트해야 합니다.
- 일부 비즈니스 트랜잭션은 여러 서비스가 소유하는 데이터를 조회해야합니다. 예를 들어, `View Available Credit`의 유스 케이스는 `creditLimit`을 확인하기 위해 Customer를 조회 해야하며 총 주문 금액을 계산하기 위해 Order를 조회해야합니다.

- 일부 쿼리는 여러 서비스가 소유하고 있는 데이터를 조인해야합니다. 예를 들어, 몇몇 지역의 고객과 그들의 최근 주문내역을 확인하기 위해서는 Customer와 Order를 조인해야합니다.
- 데이터베이스는 때때로 확장을 위하여 복제되고 공유되어야합니다. Scale Cube를 참고하세요.
- 서비스들은 서로 다른 데이터 저장 요구 사항을 가집니다. 몇몇 서비스에서는 관계형 데이터베이스가 최고의 선택입니다. 다른 서비스들에서는 MongoDB와 같은 NoSQL 데이터베이스가 최선이 될 수 도 있습니다. 이는 복잡하고 구조화 되지 않은 데이터 또는 Neo4J를 저장하기에 적합합니다. 또한 그래프 데이터를 조회하고 저장하는데에 효율적입니다.

해결책

- 여러 서비스가 공유하는 단일 데이터베이스를 사용하십시오. 각 서비스는 로컬 ACID 트랜잭션을 사용하여 다른 서비스가 소유한 데이터에 자유롭게 액세스합니다.

예제

`OrderService`와 `CustomerService`는 자유롭게 각자의 테이블에 엑세스합니다. 예를 들어, `OrderService`는 아래의 ACID 트랙잭션을 사용할 수 있으며 새로운 주문이 고객의 신용한도를 위반하지 않도록 보장합니다.

```
BEGIN TRANSACTION
…
SELECT ORDER_TOTAL
 FROM ORDERS WHERE CUSTOMER_ID = ?
…
SELECT CREDIT_LIMIT
FROM CUSTOMERS WHERE CUSTOMER_ID = ?
…
INSERT INTO ORDERS …
…
COMMIT TRANSACTION
```

데이터베이스는 동시에 같은 고객에 대해 주문을 생성하는 트랜잭션이 있을 때도 신용한도에 초과 되지 않을 것을 보장합니다.

예상되는 상황

이 패턴은 이점은 다음과 같습니다.

- 개발자는 데이터의 일관성을 유지하기 위해 친숙하고 간단한 ACID 트랜잭션을 사용할 수 있습니다.
- 단일 데이터베이스는 더 간단하게 작동합니다.

이 패턴은 결점은 다음과 같습니다.

- Development time coupling - 예를 들어, `OrderService`를 작업하는 개발자는 스키마 변경에 대해 동일한 테이블을 액세스하는 서비스의 개발자와 협력해야합니다. 이러한 결합 및 추가적인 협력은 개발 속도를 느리게 만듭니다.
- Runtime coupling - 모든 서비스는 같은 데이터베이스에 액세스하기 때문에 잠재적으로 충돌할 가능성이 있습니다. 예를 들어, 실행중인 `CustomerService`의 트랜잭션이 `ORDER`에 대한 락을 선점하고 있다면 `OrderService`은 블락될 것입니다.
- 하나의 데이터베이스는 모든 서비스의 데이터 저장 및 액세스 요구사항을 만족시키지 못 할 수 있습니다.

관련 패턴

- Database per Service 패턴을 대안으로 사용할 수 있습니다.

> **참고 문헌**
>
> 본 문서는 Microservices.io에서 작성한 A pattern language for microservices를 번역한 문서입니다.
>
> **Reference**
>
> This document is a translation of A pattern language for microservices written by Microservices.io.