의 데이터베이스 분해 AWS

AWS 권장 가이드

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS 권장 가이드: 의 데이터베이스 분해 AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

소개 1
대상 독자 2
목표 2
과제 및 책임 3
일반적인 과제 3
역할 및 책임 정의 3
범위 및 요구 사항 5
코어 분석 프레임워크 5
시스템 경계 6
릴리스 주기 6
기술적 제약 조건 6
조직 컨텍스트 7
위험 평가 7
성공 기준 7
액세스 제어 8
데이터베이스 래퍼 서비스 패턴 8
이점 및 제한 사항 9
구현 10
예제 11
CQRS 패턴 13
응집 및 결합 15
응집력 및 결합 정보 15
공통 결합 패턴 17
구현 결합 패턴 17
시간적 결합 패턴 17
배포 결합 패턴 18
도메인 결합 패턴 18
일반적인 응집 패턴 19
기능적 응집 패턴 19
순차 응집 패턴 20
통신 응집 패턴 20
절차적 응집 패턴 21
임시 응집 패턴 21
논리적 또는 우연한 응집 패턴 22

구현 22
모범 사례 22
1단계: 데이터 종속성 매핑 23
2단계: 트랜잭션 경계 및 액세스 패턴 분석 23
3단계: 독립형 테이블 식별 23
비즈니스 로직 25
1단계: 분석 25
2단계: 분류 26
3단계: 마이그레이션 27
롤백 전략 27
이전 버전과의 호환성 유지 27
긴급 롤백 계획 28
테이블 관계 29
비정규화 전략 29
Reference-by-key 전략 30
CQRS 패턴 30
이벤트 기반 데이터 동기화 31
테이블 조인에 대한 대안 구현 32
시나리오 기반 예제 33
모범 사례 36
성공 측정 36
문서화 요구 사항 36
지속적인 개선 전략 37
데이터베이스 분해의 일반적인 문제 해결 37
FAQ 38
범위 및 요구 사항 FAQ 38
초기 범위 정의는 얼마나 상세해야 합니까? 39
프로젝트를 시작한 후 추가 종속성을 발견하면 어떻게 됩니까? 39
요구 사항이 충돌하는 여러 부서의 이해관계자를 처리하려면 어떻게 해야 합니까? 39
설명서가 좋지 않거나 오래된 경우 기술적 제약 조건을 평가하는 가장 좋은 방법은 무엇입니까? 39
즉각적인 비즈니스 요구 사항과 장기적인 기술 목표의 균형을 맞추려면 어떻게 해야 하나요? 40
자동 이해관계자의 중요한 요구 사항이 누락되지 않도록 하려면 어떻게 해야 합니까? 40
이러한 권장 사항이 모놀리식 메인프레임 데이터베이스에 적용되나요? 40
데이터베이스 액세스 FAQ 41

래퍼 서비스가 새로운 병목 현상이 되지 않나요? 41
기존 저장 프로시저는 어떻게 되나요? 41
전환 중에 스키마 변경을 관리하려면 어떻게 해야 합니까? 41
응집 및 결합 FAQ 42
결합 분석 시 적절한 수준의 세분화를 식별하려면 어떻게 해야 합니까? 42
데이터베이스 결합 및 응집을 분석하는 데 사용할 수 있는 도구는 무엇입니까? 42
결합 및 응집 결과를 문서화하는 가장 좋은 방법은 무엇입니까? 43
먼저 해결해야 할 결합 문제의 우선순위를 지정하려면 어떻게 해야 하나요? 43
여러 작업에 걸쳐 있는 트랜잭션을 처리하려면 어떻게 해야 합니까? 44
비즈니스 로직 마이그레이션 FAQ 44
먼저 마이그레이션할 저장 프로시저를 식별하려면 어떻게 해야 합니까? 45
로직을 애플리케이션 계층으로 이동하면 어떤 위험이 있습니까? 45
데이터베이스에서 로직을 이동할 때 성능을 유지하려면 어떻게 해야 합니까? 45
여러 테이블이 포함된 복잡한 저장 프로시저를 사용하려면 어떻게 해야 하나요? 46
마이그레이션 중에 데이터베이스 트리거를 처리하려면 어떻게 해야 합니까? 46
마이그레이션된 비즈니스 로직을 테스트하는 가장 좋은 방법은 무엇입니까? 46
데이터베이스와 애플리케이션 로직이 모두 존재하는 경우 전환 기간을 관리하려면 어떻게 해야 합니까? 47
이전에 데이터베이스에서 관리한 애플리케이션 계층의 오류 시나리오를 처리하려면 어떻게 해야 합니까? 47
다음 단계 48
증분 전략 48
기술적 고려 사항 48
조직 변경 사항 49
리소스 50
AWS 권장 가이드 50
AWS 블로그 게시물 50
AWS 서비스 50
기타 도구 50
기타 리소스 51
문서 기록 52
..... liii

의 데이터베이스 분해 AWS

Philippe Wanner와 Saurabh Sharma, Amazon Web Services

2025년 10월(문서 기록)

데이터베이스 현대화, 특히 모놀리식 데이터베이스의 분해는 데이터 관리 시스템의 민첩성, 확장성 및 성능을 개선하려는 조직에 중요한 워크스트림입니다. 기업이 성장하고 데이터 요구가 더 복잡해짐에 따라 기존 모놀리식 데이터베이스는 속도를 유지하는 데 어려움을 겪는 경우가 많습니다. 이로 인해 성능 병목 현상, 유지 관리 문제 및 변화하는 비즈니스 요구 사항에 적응하기가 어려워집니다.

모놀리식 데이터베이스의 일반적인 문제는 다음과 같습니다.

- 비즈니스 도메인 오정렬 - 모놀리식 데이터베이스는 종종 기술을 고유한 비즈니스 도메인과 조정하지 못하여 조직의 성장을 제한할 수 있습니다.
- 확장성 제약 조건 - 시스템은 규모 조정 한도에 자주 도달하여 비즈니스 확장에 장벽을 만듭니다.
- 아키텍처 견고성 - 단단히 결합된 구조로 인해 전체 시스템에 영향을 주지 않고 특정 구성 요소를 업데이트하기가 어렵습니다.
- 성능 저하 - 데이터 로드가 증가하고 사용자 동시성이 증가하면 시스템 성능이 저하되는 경우가 많습니다.

데이터베이스 분해의 이점은 다음과 같습니다.

- 비즈니스 민첩성 향상 - 분해를 통해 변화하는 비즈니스 요구 사항에 신속하게 적응할 수 있으며 독립적인 규모 조정을 지원합니다.
- 성능 최적화 - 분해를 통해 특정 사용 사례에 맞게 조정되고 각 데이터베이스를 독립적으로 확장하는 특수한 데이터베이스 솔루션을 만들 수 있습니다.
- 비용 관리 개선 - 분해를 통해 리소스 사용률을 높이고 운영 비용을 절감할 수 있습니다.
- 유연한 라이선스 옵션 - 분해를 통해 비용이 많이 드는 독점 라이선스에서 오픈 소스 대안으로 전환할 수 있습니다.
- 혁신 지원 - 분해를 통해 특정 워크로드에 맞게 특별히 구축된 데이터베이스를 쉽게 채택할 수 있습니다.

대상 독자

이 가이드는 데이터베이스 아키텍트, 클라우드 솔루션 아키텍트, 애플리케이션 개발 팀 및 엔터프라이즈 아키텍트를 지원합니다. 모놀리식 데이터베이스를 마이크로서비스 정렬 데이터 스토어로 분해하고, 도메인 기반 데이터베이스 아키텍처를 구현하고, 데이터베이스 마이그레이션 전략을 계획하고, 증가하는 비즈니스 수요에 맞게 데이터베이스 운영을 확장하는 데 도움이 되도록 설계되었습니다. 이 가이드의 개념과 권장 사항을 이해하려면 관계형 및 NoSQL 데이터베이스 원칙, AWS 관리형 데이터베이스 서비스 및 마이크로서비스 아키텍처 패턴을 숙지해야 합니다. 이 가이드는 데이터베이스 분해 프로젝트의 초기 단계에 있는 조직을 돕기 위한 것입니다.

목표

이 가이드는 조직이 다음 목표를 달성하는 데 도움이 될 수 있습니다.

- 대상 아키텍처를 분해하기 위한 요구 사항을 수집합니다.
- 위험을 평가하고 소통하기 위한 체계적인 방법론을 개발합니다.
- 분해 계획을 생성합니다.
- 성공 지표, 핵심 성과 지표(KPIs), 완화 전략 및 비즈니스 연속성 계획을 정의합니다.
- 비즈니스 수요를 따르는 데 도움이 되는 더 나은 워크로드 탄력성을 설정합니다.
- 혁신을 가능하게 하는 특정 사용 사례에 특수 데이터베이스를 채택하는 방법을 알아봅니다.
- 조직의 데이터 보안 및 거버넌스를 강화합니다.
- 다음을 통해 비용을 절감합니다.
  - 라이선스 요금 절감
  - 공급업체 잠금 감소
  - 광범위한 커뮤니티 지원 및 혁신에 대한 액세스 개선
  - 구성 요소마다 다른 데이터베이스 기술을 선택할 수 있는 기능
  - 점진적 마이그레이션으로 위험을 줄이고 시간이 지남에 따라 비용을 분산합니다.
  - 리소스 사용률 개선

데이터베이스 분해에 대한 일반적인 과제 및 책임 관리

데이터베이스 분해는 신중한 계획, 실행 및 관리가 필요한 복잡한 프로세스입니다. 조직이 데이터 인프라를 현대화하려고 할 때 프로젝트의 성공에 영향을 미칠 수 있는 수많은 문제에 직면하는 경우가 많습니다. 이 섹션에서는 일반적인 장애물을 설명하고 이러한 장애물을 극복하기 위한 구조화된 접근 방식을 소개합니다.

일반적인 과제

데이터베이스 분해 프로젝트는 기술, 인력 및 비즈니스 차원에서 여러 가지 과제에 직면합니다. 기술 측면에서 분산 시스템 간의 데이터 일관성을 보장하는 것은 상당한 장애물입니다. 또한 전환 기간 동안 잠재적인 성능 및 안정성 영향을 미칠 수 있으므로 기존 시스템과 원활하게 통합해야 합니다. 사람과 관련된 문제에는 새 시스템과 관련된 학습 곡선, 직원의 변화에 대한 잠재적 저항, 필요한 리소스의 가용성이 포함됩니다. 비즈니스 관점에서 볼 때 프로젝트는 타임라인 오버런 위험, 예산 제약, 마이그레이션 프로세스 중 비즈니스 중단 가능성과 경쟁해야 합니다.

역할 및 책임 정의

기술, 사람 및 비즈니스 차원에 걸친 이러한 복잡한 문제를 고려할 때 프로젝트 성공에 명확한 역할과 책임을 설정하는 것이 중요합니다. Responsible, Accountable, Consulted, and Informed(RACI) 매트릭스는 이러한 문제를 해결하는 데 필요한 구조를 제공합니다. 결정을 내리는 사람, 작업을 수행하는 사람, 의견을 제공하는 사람, 분해의 각 단계에서 최신 정보를 유지해야 하는 사람을 명시적으로 정의합니다. 이러한 명확성은 모호한 의사 결정으로 인한 지연을 방지하는 데 도움이 되며, 적절한 이해관계자 참여를 장려하고, 주요 결과물에 대한 책임을 생성합니다. 이러한 프레임워크가 없으면 팀은 중복된 책임, 누락된 커뮤니케이션 및 명확하지 않은 에스컬레이션 경로로 인해 어려움을 겪을 수 있습니다. 이러한 문제는 기존의 기술적 복잡성을 악화시키고 관리 문제를 변경하는 동시에 타임라인 및 예산 오버런 위험을 높일 수 있습니다.

다음 샘플 RACI 매트릭스는 조직의 잠재적 역할과 책임을 명확히 하는 데 도움이 될 수 있는 출발점입니다.

작업 또는 활동	프로젝트 관리자	아키텍트	개발자	이해관계자
비즈니스 성과 및 과제 식별	A/R	R	C	–

범위 정의 및 요구 사항 식별	A	R	C	C/I
프로젝트 성공 지표 식별	A	R	C	I
커뮤니케이션 계획 생성 및 실행	A/R	C	C	I
대상 아키텍처 정의	I	A/R	C	–
데이터베이스 액세스 제어	I	A/R	R	–
비즈니스 연속성 계획 생성 및 실행	A/R	C	I	–
응집력 및 결합 분석	I	A/R	R	I
데이터베이스에서 애플리케이션 계층으로 비즈니스 로직(예: 저장 프로시저) 이동	I	A	R	–
조인이라고 하는 테이블 관계 분리	I	A	R	–

데이터베이스 분해의 범위 및 요구 사항 정의

범위를 정의하고 데이터베이스 분해 프로젝트의 요구 사항을 식별할 때는 조직의 요구 사항에서 역방향으로 작업해야 합니다. 이를 위해서는 기술적 타당성과 비즈니스 가치의 균형을 맞추는 체계적인 접근 방식이 필요합니다. 이 초기 단계는 전체 프로세스의 기반을 설정하고 프로젝트의 목표가 조직의 목표 및 역량과 일치하는지 확인하는 데 도움이 됩니다.

이 섹션은 다음 주제를 포함합니다:

- 핵심 분석 프레임워크 설정
- 데이터베이스 분해를 위한 시스템 경계 정의
- 릴리스 주기 고려
- 데이터베이스 분해에 대한 기술적 제약 조건 평가
- 조직 컨텍스트 이해
- 데이터베이스 분해 위험 평가
- 데이터베이스 분해에 대한 성공 기준 정의

핵심 분석 프레임워크 설정

범위 정의는 네 가지 상호 연결된 단계를 통해 분석을 안내하는 체계적인 워크플로로 시작됩니다. 이 포괄적인 접근 방식을 통해 데이터베이스 분해 작업은 기존 시스템 및 운영 요구 사항을 철저히 이해할 수 있습니다. 다음은 핵심 분석 프레임워크의 단계입니다.

1. 작업자 분석 - 데이터베이스와 상호 작용하는 모든 시스템 및 애플리케이션을 철저히 식별합니다. 여기에는 쓰기 작업을 수행하는 생산자와 읽기 작업을 처리하는 소비자를 모두 매핑하는 동시에 액세스 패턴, 빈도 및 최대 사용 시간을 문서화하는 작업이 포함됩니다. 이 고객 중심 보기는 변경 사항의 영향을 이해하고 분해 중에 특별한 주의가 필요한 중요한 경로를 식별하는 데 도움이 됩니다.
2. 활동 분석 - 각 액터가 수행하는 특정 작업에 대해 자세히 알아봅니다. 각 시스템에 대한 세부 생성, 읽기, 업데이트 및 삭제(CRUD) 매트릭스를 생성하고 액세스하는 테이블과 방법을 식별합니다. 이 분석을 통해 분해의 자연 경계를 발견하고 현재 아키텍처를 간소화할 수 있는 영역을 강조할 수 있습니다.
3. 종속성 매핑 - 시스템 간의 직접 및 간접 종속성을 모두 문서화하여 데이터 흐름 및 관계를 명확하게 시각화합니다. 이를 통해 신뢰를 얻기 위해 신중한 계획이 필요한 잠재적 중단점과 영역을 식별할 수 있습니다. 분석은 공유 테이블 및 외래 키와 같은 기술적 종속성과 워크플로 시퀀스 및 보고 요구 사항과 같은 비즈니스 프로세스 종속성을 모두 고려합니다.

4. 일관성 요구 사항 - 높은 표준으로 각 작업의 일관성 요구 사항을 검사합니다. 금융 거래와 같이 즉각적인 일관성이 필요한 작업을 결정합니다. 분석 업데이트와 같은 다른 작업은 최종 일관성으로 작동할 수 있습니다. 이 분석은 프로젝트 전반의 분해 패턴 및 아키텍처 결정 선택에 직접적인 영향을 미칩니다.

데이터베이스 분해를 위한 시스템 경계 정의

시스템 경계는 데이터 소유권, 액세스 패턴 및 통합 지점을 포함하여 한 시스템이 끝나고 다른 시스템이 시작되는 위치를 정의하는 논리적 경계입니다. 시스템 경계를 정의할 때는 포괄적인 계획과 실제 구현 요구 사항의 균형을 맞추는 사려 깊지만 결정적인 선택을 하세요. 데이터베이스를 여러 물리적 데이터베이스 또는 스키마에 걸쳐 있을 수 있는 논리적 단위로 간주합니다. 이 경계 정의는 다음과 같은 중요한 목표를 달성합니다.

- 모든 외부 액터와 해당 상호 작용 패턴을 식별합니다.
- 인바운드 및 아웃바운드 종속성을 모두 포괄적으로 매핑합니다.
- 기술 및 운영 제약 조건 문서화
- 분해 작업의 범위를 명확하게 설명합니다.

릴리스 주기 고려

릴리스 주기를 이해하는 것은 데이터베이스 분해를 계획하는 데 매우 중요합니다. 대상 시스템과 종속 시스템 모두에 대한 갱신 시간을 검토합니다. 조정된 변경 기회를 식별합니다. 분해 전략에 영향을 미칠 수 있으므로 연결된 시스템의 계획된 폐기를 고려하세요. 기존 변경 기간 및 배포 제약 조건을 고려하여 비즈니스 중단을 최소화합니다. 구현 계획이 연결된 모든 시스템의 릴리스 일정에 맞는지 확인합니다.

데이터베이스 분해에 대한 기술적 제약 조건 평가

데이터베이스 분해를 진행하기 전에 현대화 접근 방식을 구성하는 주요 기술적 제한 사항을 평가합니다. 데이터베이스 버전, 프레임워크, 성능 요구 사항 및 서비스 수준 계약을 포함하여 현재 기술 스택의 기능을 검토합니다. 특히 규제 산업의 경우 보안 및 규정 준수 요구 사항을 고려하세요. 현재 데이터 볼륨, 성장 예측 및 사용 가능한 마이그레이션 도구를 검토하여 조정 결정을 알립니다. 마지막으로 소스 코드 및 시스템 수정에 대한 액세스 권한을 확인합니다. 이렇게 하면 실행 가능한 분해 전략이 결정되기 때문입니다.

조직 컨텍스트 이해

데이터베이스 분해에 성공하려면 시스템이 작동하는 광범위한 조직 환경을 이해해야 합니다. 부서 간 종속성을 매핑하고 팀 간에 명확한 커뮤니케이션 채널을 설정합니다. 팀의 기술 역량을 평가하고 해결해야 할 교육 요구 사항 또는 기술 격차를 식별합니다. 전환을 관리하고 비즈니스 연속성을 유지하는 방법을 포함하여 변경 관리 영향을 고려합니다. 사용 가능한 리소스와 예산 또는 인력 배치 제한과 같은 제약 조건을 평가합니다. 마지막으로 분해 전략을 이해관계자의 기대치 및 우선순위에 맞게 조정하여 프로젝트 전반에 걸쳐 지속적인 지원을 장려합니다.

데이터베이스 분해 위험 평가

데이터베이스 분해 성공을 위해서는 포괄적인 위험 평가가 필수적입니다. 마이그레이션 중 데이터 무결성, 잠재적 시스템 성능 저하, 가능한 통합 장애, 보안 취약성과 같은 위험을 신중하게 평가합니다. 이러한 기술적 과제는 잠재적 운영 중단, 리소스 제한, 타임라인 지연, 예산 제약 등 비즈니스 위험과 균형을 이루어야 합니다. 식별된 각 위험에 대해 특정 완화 전략 및 비상 계획을 개발하여 비즈니스 운영을 보호하면서 프로젝트 추진력을 유지합니다.

잠재적 문제의 영향과 확률을 모두 평가하는 위험 매트릭스를 생성합니다. 기술 팀 및 비즈니스 이해관계자와 협력하여 위험을 식별하고, 개입에 대한 명확한 임계값을 설정하고, 특정 완화 전략을 개발합니다. 예를 들어 데이터 손실 위험을 높은 영향과 낮은 확률로 평가하며 강력한 백업 전략이 필요합니다. 사소한 성능 저하는 중간 영향과 높은 확률일 수 있으며 사전 모니터링이 필요합니다.

정기적인 위험 검토 주기를 설정하여 우선순위를 재평가하고 프로젝트가 발전함에 따라 완화 계획을 조정합니다. 이 체계적인 접근 방식을 사용하면 리소스가 새로운 문제에 대한 명확한 에스컬레이션 경로를 유지하면서 가장 중요한 위험에 집중할 수 있습니다.

데이터베이스 분해에 대한 성공 기준 정의

데이터베이스 분해의 성공 기준은 여러 차원에서 명확하게 정의되고 측정 가능해야 합니다. 비즈니스 관점에서 비용 절감, time-to-market, 시스템 가용성 및 고객 만족도를 위한 특정 목표를 설정합니다. 시스템 성능, 배포 효율성, 데이터 일관성 및 전반적인 신뢰성의 정량화 가능한 개선을 통해 기술적 성공을 측정해야 합니다. 마이그레이션 프로세스의 경우 데이터 손실 없음, 허용 가능한 비즈니스 중단 제한, 예산 규정 준수 및 타임라인 준수에 대한 엄격한 요구 사항을 정의합니다.

기준 및 대상 지표, 명확한 측정 방법론 및 정기 검토 일정을 유지하여 이러한 기준을 철저히 문서화합니다. 각 성공 지표에 대해 명확한 소유자를 할당하고 서로 다른 지표 간에 종속성을 매핑합니다. 성공을 측정하는이 포괄적인 접근 방식은 기술적 업적을 비즈니스 성과와 일치시키는 동시에 분해 여정 전반에 걸쳐 책임을 유지합니다.

분해 중 데이터베이스 액세스 제어

많은 조직이 수년 동안 유기적으로 성장하고 여러 서비스 및 팀이 직접 액세스하는 중앙 데이터베이스라는 일반적인 시나리오에 직면합니다. 이렇게 하면 다음과 같은 몇 가지 중요한 문제가 발생합니다.

- 통제되지 않은 성장 - 팀이 지속적으로 새로운 기능을 추가하고 스키마를 수정함에 따라 데이터베이스는 점점 더 복잡해지고 관리가 어려워집니다.
- 성능 문제 - 하드웨어 개선에도 불구하고 로드가 증가하면 결국 데이터베이스의 기능을 초과할 위험이 있습니다. 스키마 복잡성 또는 기술 부족으로 인해 쿼리를 조정할 수 없습니다. 시스템 성능을 예측하거나 설명할 수 없습니다.
- 분해 마비 - 여러 팀이 데이터베이스를 적극적으로 수정하는 동안 데이터베이스를 분할하거나 리팩터링하는 것은 거의 불가능합니다.

> **Note**
>
> 모놀리식 데이터베이스 시스템은 애플리케이션이나 서비스 또는 관리에 동일한 자격 증명을 재사용하는 경우가 많습니다. 이로 인해 데이터베이스 추적성이 저하됩니다. 전용 역할을 설정하고 최소 권한 원칙을 채택하면 보안 및 가용성을 높이는 데 도움이 될 수 있습니다.

무선 상태가 된 모놀리식 데이터베이스를 처리할 때 액세스를 제어하는 가장 효과적인 패턴 중 하나를 데이터베이스 래퍼 서비스라고 합니다. 복잡한 데이터베이스 시스템을 관리하는 전략적 첫 단계를 제공합니다. 제어된 데이터베이스 액세스를 설정하고 위험을 줄이면서 점진적으로 현대화할 수 있습니다. 이 접근 방식은 데이터 사용 패턴 및 종속성에 대한 명확한 가시성을 제공하여 점진적 개선을 위한 기반을 만듭니다. 전체 데이터베이스 분해를 위한 단계 역할을 하는 전환형 아키텍처입니다. 래퍼 서비스는 이러한 여정을 성공적으로 수행하는 데 필요한 안정성과 제어를 제공합니다.

이 섹션은 다음 주제를 포함합니다:

- 데이터베이스 래퍼 서비스 패턴으로 액세스 제어
- CQRS 패턴으로 액세스 제어

데이터베이스 래퍼 서비스 패턴으로 액세스 제어

래퍼 서비스는 데이터베이스의 파사드 역할을 하는 서비스 계층입니다. 이 접근 방식은 향후 분해를 준비하면서 기존 기능을 유지해야 할 때 특히 유용합니다. 이 패턴은 간단한 원칙을 따릅니다. 무언가 너

무 지저분하면 먼저 지저분한 것을 포함합니다. 래퍼 서비스는 데이터베이스에 액세스할 수 있는 유일한 승인된 방법이 되어 기본 복잡성을 숨기면서 제어된 인터페이스를 제공합니다.

복잡한 스키마로 인해 즉각적인 데이터베이스 분해가 가능하지 않거나 여러 서비스에 지속적인 데이터 액세스가 필요한 경우이 패턴을 사용합니다. 시스템 안정성을 유지하면서 신중한 리팩터링을 위한 시간을 제공하므로 전환 기간 동안 특히 유용합니다. 이 패턴은 데이터 소유권을 특정 팀에 통합하거나 새 애플리케이션에 여러 테이블에서 집계된 뷰가 필요한 경우에 적합합니다.

예를 들어 다음과 같은 경우이 패턴을 적용합니다.

- 스키마 복잡성으로 인한 즉각적인 분리 방지
- 여러 팀에 지속적인 데이터 액세스 필요
- 점진적 현대화가 선호됩니다.
- 팀 구조 조정에는 명확한 데이터 소유권이 필요합니다.
- 새 애플리케이션에는 통합 데이터 보기가 필요합니다.

데이터베이스 래퍼 서비스 패턴의 이점 및 제한 사항

다음은 데이터베이스 래퍼 패턴의 이점입니다.

- 성장 제어 - 래퍼 서비스는 데이터베이스 스키마에 대한 추가 제어되지 않는 추가를 방지합니다.
- 명확한 경계 - 구현 프로세스를 통해 명확한 소유권 및 책임 경계를 설정할 수 있습니다.
- 리팩터링 자유 - 래퍼 서비스를 사용하면 소비자에게 영향을 주지 않고 내부적으로 변경할 수 있습니다.
- 향상된 관찰성 - 래퍼 서비스는 모니터링 및 로깅을 위한 단일 지점입니다.
- 간소화된 테스트 - 래퍼 서비스를 사용하면 서비스를 더 쉽게 사용하여 테스트를 위한 간소화된 모의 버전을 생성할 수 있습니다.

다음은 데이터베이스 래퍼 패턴의 제한 사항입니다.

- 기술 결합 - 래퍼 서비스는 소비 서비스와 동일한 기술 스택을 사용할 때 가장 잘 작동합니다.
- 초기 오버헤드 - 래퍼 서비스에는 성능에 영향을 미칠 수 있는 추가 인프라가 필요합니다.
- 마이그레이션 작업 - 래퍼 서비스를 구현하려면 팀 간에 조정하여 직접 액세스에서 전환해야 합니다.
- 성능 - 래핑 서비스에서 트래픽이 많거나 사용량이 많거나 자주 액세스하는 경우 서비스를 소비하면 성능이 저하될 수 있습니다. 래퍼 서비스는 데이터베이스 위에 페이지 매김, 커서 및 데이터베이스

연결을 처리해야 합니다. 사용 사례에 따라 규모가 조정되지 않고 추출, 변환 및 로드(ETL) 워크로드에 적합하지 않을 수 있습니다.

데이터베이스 래퍼 서비스 패턴 구현

데이터베이스 래퍼 서비스 패턴을 구현하는 두 단계가 있습니다. 먼저 데이터베이스 래퍼 서비스를 생성합니다. 그런 다음 모든 액세스를 통해 전달하고 액세스 패턴을 문서화합니다.

1단계: 데이터베이스 래퍼 서비스 생성

데이터베이스에 대한 게이트키퍼 역할을 하는 경량 서비스 계층을 생성합니다. 처음에는 모든 기존 기능을 미러링해야 합니다. 이 래퍼 서비스는 직접 데이터베이스 종속성을 서비스 수준 종속성으로 변환하는 모든 데이터베이스 작업에 대한 필수 액세스 포인트가 됩니다. 이 계층에서 세부 로깅 및 모니터링을 구현하여 사용 패턴, 성능 지표 및 액세스 빈도를 추적합니다. 기존 저장 프로시저를 유지하되이 새 서비스 인터페이스를 통해서만 액세스해야 합니다.

2단계: 액세스 제어 구현

래퍼 서비스를 통해 모든 데이터베이스 액세스를 체계적으로 리디렉션한 다음 데이터베이스에 직접 액세스하는 외부 시스템에서 직접 데이터베이스 권한을 취소합니다. 서비스가 마이그레이션될 때 각 액세스 패턴 및 종속성을 문서화합니다. 이렇게 제어된 액세스를 통해 외부 소비자를 방해하지 않고 데이터베이스 구성 요소를 내부 리팩터링할 수 있습니다. 예를 들어 복잡한 트랜잭션 워크플로 대신 위험이 낮은 읽기 전용 작업으로 시작합니다.

3단계: 데이터베이스 성능 모니터링

래퍼 서비스를 데이터베이스 성능에 대한 중앙 집중식 모니터링 포인트로 사용합니다. 쿼리 응답 시간, 사용 패턴, 오류율 및 리소스 사용률을 포함한 주요 지표를 추적합니다. 성능 임계값 및 비정상적인 패턴에 대한 알림을 설정합니다. 예를 들어 실행 속도가 느린 쿼리, 연결 풀 사용률 및 트랜잭션 처리량을 모니터링하여 잠재적 문제를 사전에 식별합니다.

이 통합 보기를 사용하면 쿼리 튜닝, 리소스 할당 조정 및 사용 패턴 분석을 통해 데이터베이스 성능을 최적화할 수 있습니다. 래퍼 서비스의 중앙 집중식 특성을 통해 일관된 성능 표준을 유지하면서 모든 소비자에 걸쳐 개선 사항을 구현하고 영향을 검증할 수 있습니다.

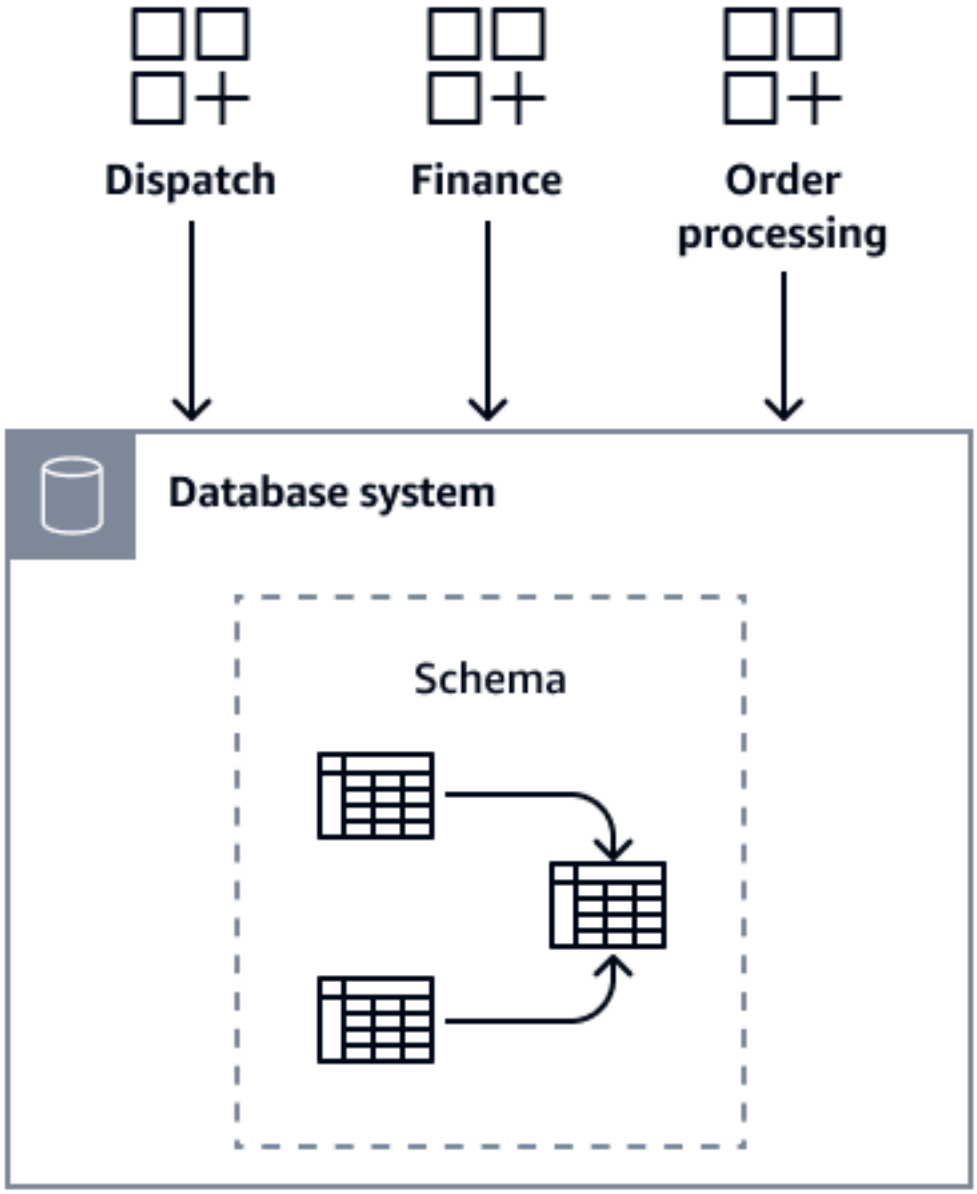
데이터베이스 래퍼 서비스 구현 모범 사례

다음 모범 사례는 데이터베이스 래퍼 서비스를 구현하는 데 도움이 될 수 있습니다.

- 작은 시작 - 기존 기능을 간단히 프록시하는 최소한의 래퍼로 시작합니다.
- 안정성 유지 - 내부 개선을 수행하면서 서비스 인터페이스를 안정적으로 유지
- 사용량 모니터링 - 포괄적인 모니터링을 구현하여 액세스 패턴 이해
- 명확한 소유권 - 래퍼와 기본 스키마를 모두 유지할 전용 팀 할당
- 로컬 스토리지 장려 - 팀이 자신의 데이터베이스에 데이터를 저장하도록 동기를 부여합니다.

시나리오 기반 예제

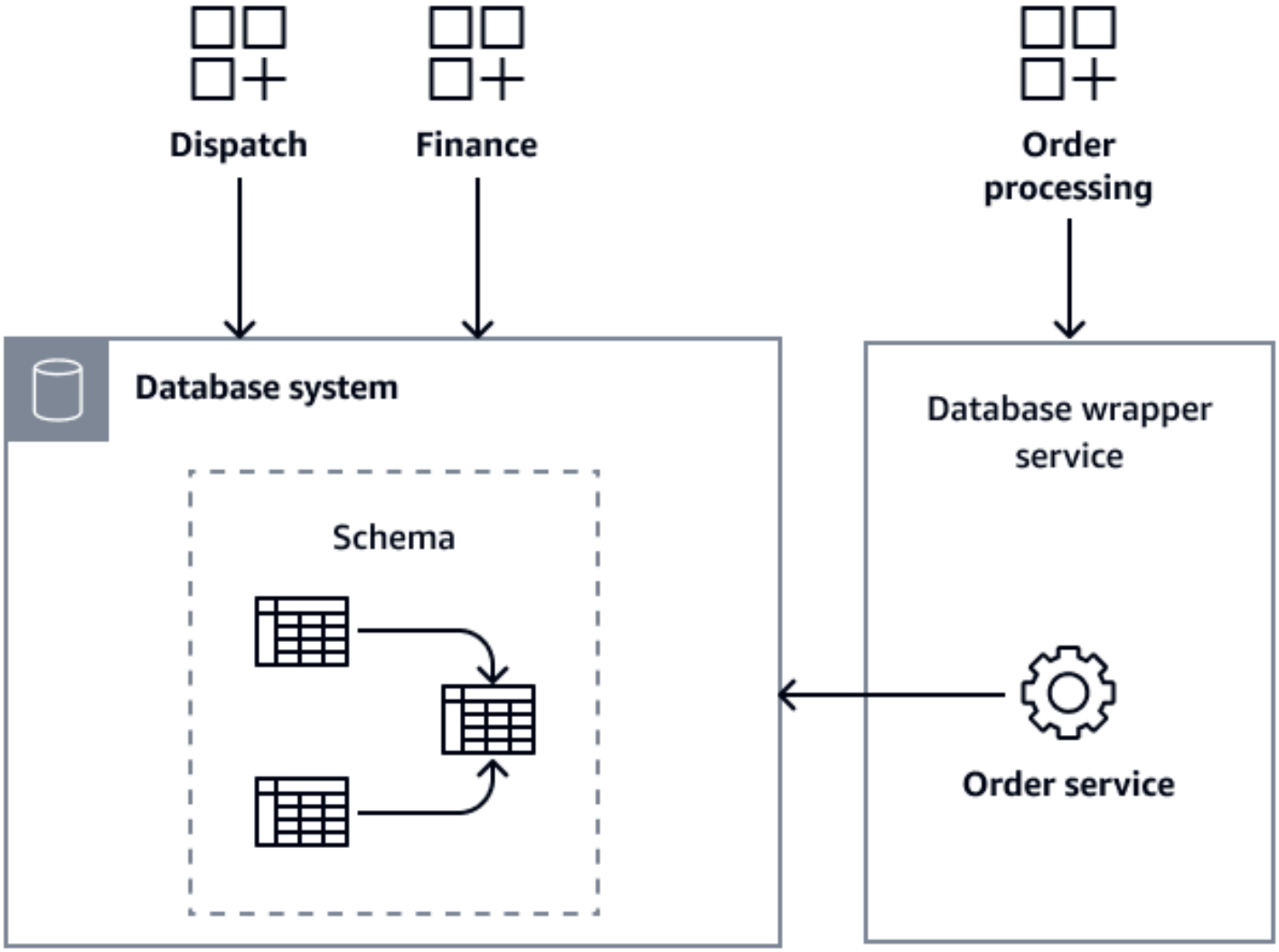
이 섹션에서는 AnyCompany Books라는 가상의 회사가 데이터베이스 래퍼 패턴을 사용하여 모놀리식 데이터베이스 시스템에 대한 액세스를 제어하는 방법의 예를 설명합니다. AnyCompany Books에는 발송, 재무 및 주문 처리라는 세 가지 중요한 서비스가 있습니다. 이러한 서비스는 중앙 데이터베이스에 대한 액세스를 공유합니다. 각 서비스는 다른 팀에서 유지 관리합니다. 시간이 지남에 따라 특정 요구 사항을 충족하도록 데이터베이스 스키마를 독립적으로 수정합니다. 이로 인해 종속성 웹이 얽히고 데이터베이스 구조가 점점 더 복잡해졌습니다.



회사의 애플리케이션 또는 엔터프라이즈 아키텍트는이 모놀리식 데이터베이스를 분해해야 할 필요성을 인식합니다. 이들의 목표는 각 서비스에 자체 전용 데이터베이스를 제공하여 유지 관리를 개선하고 팀 간 종속성을 줄이는 것입니다. 그러나 중요한 문제에 직면합니다. 세 팀 모두 진행 중인 프로젝트에 맞게 데이터베이스를 계속 적극적으로 수정하는 동안 데이터베이스를 분해하는 것은 거의 불가능합니다. 지속적인 스키마 변경과 팀 간의 조정 부족으로 인해 상당한 재구성을 시도하는 것이 매우 위험합니다.

아키텍트는 데이터베이스 래퍼 서비스 패턴을 사용하여 모놀리식 데이터베이스에 대한 액세스 제어를 시작합니다. 먼저 주문 서비스라는 특정 모듈에 대한 데이터베이스 래퍼 서비스를 설정합니다. 그런 다

음 데이터베이스에 직접 액세스하는 대신 래퍼 서비스에 액세스하도록 주문 처리 서비스를 리디렉션합니다. 다음 이미지는 수정된 인프라를 보여줍니다.

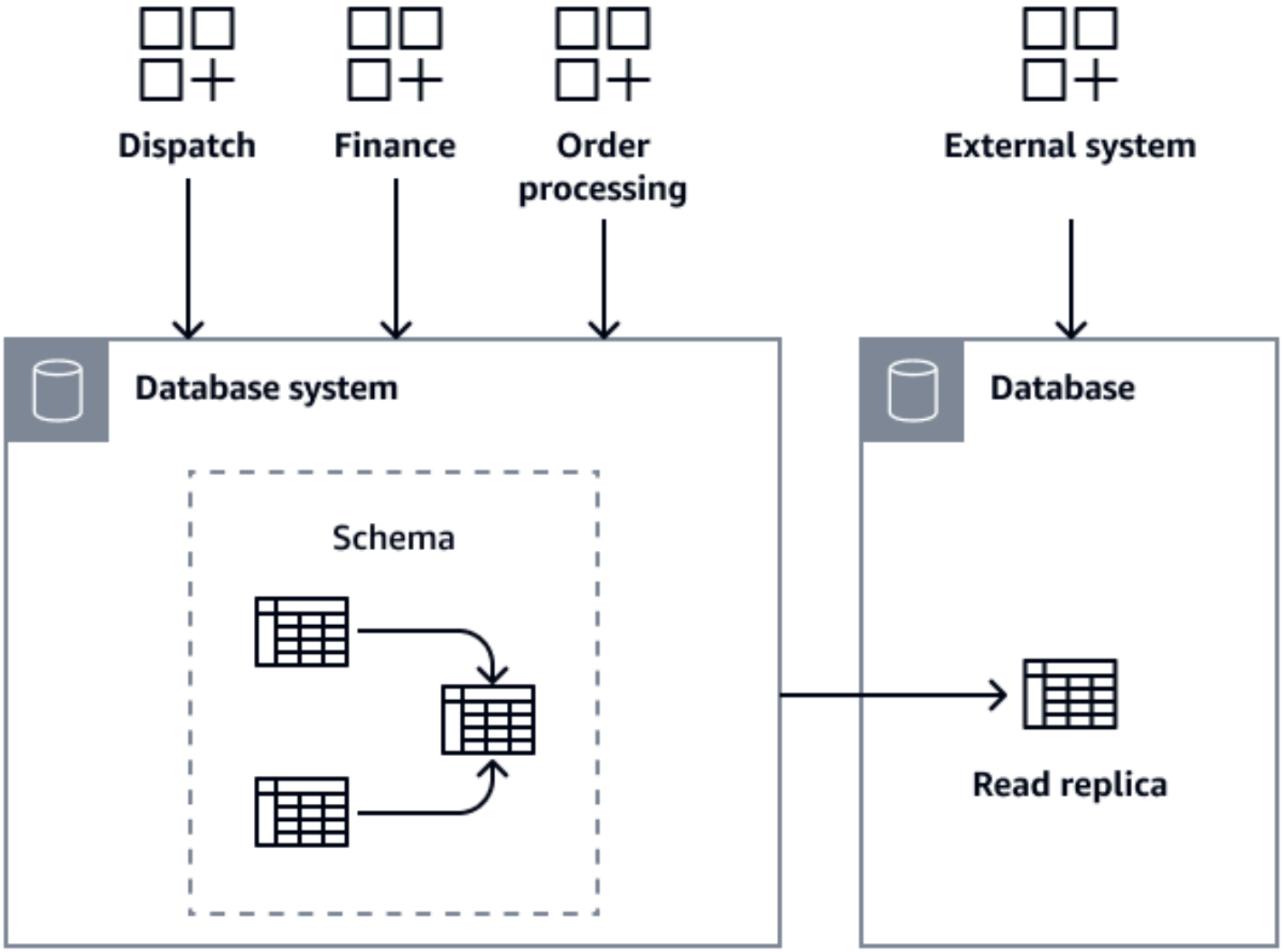


CQRS 패턴으로 액세스 제어

이 중앙 데이터베이스에 연결하는 외부 시스템을 격리하는 데 사용할 수 있는 또 다른 패턴은 명령 쿼리 책임 분리(CQRS)입니다. 일부 외부 시스템이 주로 분석, 보고 또는 기타 읽기 집약적인 작업과 같은 읽기를 위해 중앙 데이터베이스에 연결하는 경우 별도의 읽기 최적화 데이터 스토어를 생성할 수 있습니다.

이 패턴은 데이터베이스 분해 및 스키마 변경의 영향으로부터 이러한 외부 시스템을 효과적으로 격리합니다. 팀은 특정 쿼리 패턴에 대해 전용 읽기 전용 복제본 또는 특별히 구축된 데이터 스토어를 유지 관리하여 기본 데이터베이스 구조의 변경에 영향을 받지 않고 작업을 계속할 수 있습니다. 예를 들어 모놀리식 데이터베이스를 분해하는 동안 보고 시스템은 기존 데이터 보기로 계속 작업할 수 있으며 분

석 워크로드는 전용 분석 저장소를 통해 현재 쿼리 패턴을 유지할 수 있습니다. 이 접근 방식은 기술적 격리를 제공하고 조직 자율성을 활성화합니다. 여러 팀이 기본 데이터베이스의 변환 여정에 긴밀하게 결합하지 않고도 시스템을 독립적으로 발전시킬 수 있기 때문입니다.



이 패턴에 대한 자세한 내용과 테이블 관계를 분리하는 데 사용하는 예제는이 가이드 CQRS 패턴뒷부분의 섹션을 참조하세요.

데이터베이스 분해를 위한 응집력 및 결합 분석

이 섹션에서는 모놀리식 데이터베이스의 결합 및 응집 패턴을 분석하여 분해를 안내하는 데 도움이 됩니다. 데이터베이스 구성 요소가 상호 작용하고 서로 의존하는 방식을 이해하는 것은 자연스러운 중단점을 식별하고, 복잡성을 평가하고, 단계별 마이그레이션 접근 방식을 계획하는 데 매우 중요합니다. 이 분석은 숨겨진 종속성을 공개하고, 즉각적인 분리에 적합한 영역을 강조하며, 변환 위험을 최소화하면서 분해 작업의 우선순위를 정하는 데 도움이 됩니다. 결합과 응집력을 모두 검사하면 변환 프로세스 전반에 걸쳐 시스템 안정성을 유지하기 위해 구성 요소 분리 시퀀스에 대해 정보에 입각한 결정을 내릴 수 있습니다.

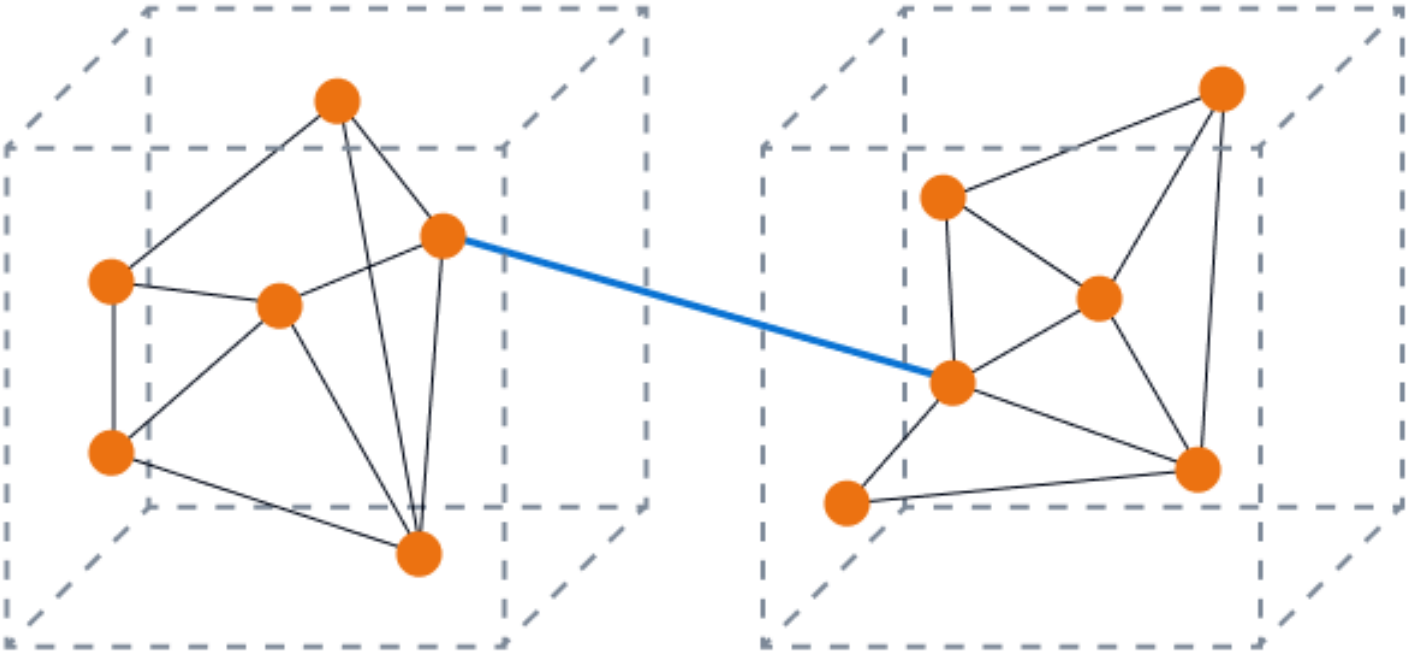
이 섹션은 다음 주제를 포함합니다:

- 응집력 및 결합 정보
- 모놀리식 데이터베이스의 공통 결합 패턴
- 모놀리식 데이터베이스의 일반적인 응집 패턴
- 낮은 결합 및 높은 응집력 구현

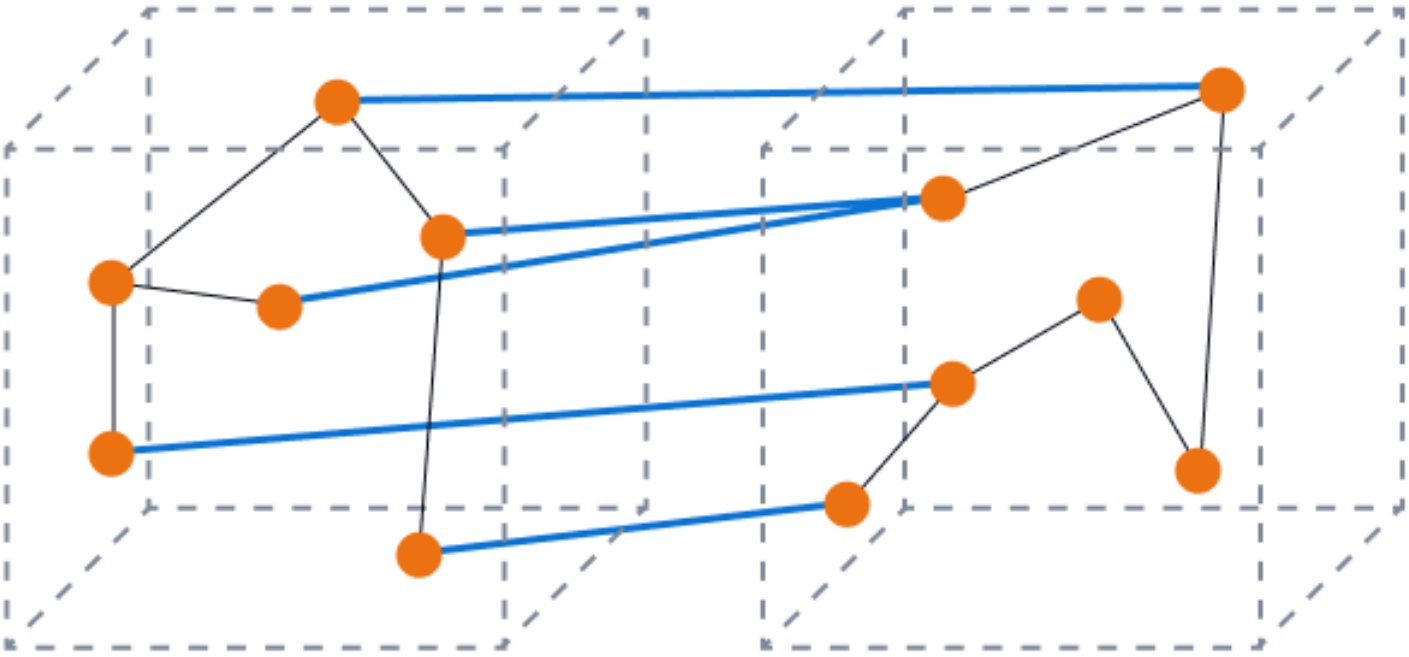
응집력 및 결합 정보

결합은 데이터베이스 구성 요소 간의 상호 의존도를 측정합니다. 잘 설계된 시스템에서는 한 구성 요소에 대한 변경 사항이 다른 구성 요소에 미치는 영향을 최소화하는 느슨한 결합을 달성하고자 합니다. 응집력은 데이터베이스 구성 요소 내의 요소가 함께 작동하여 잘 정의된 단일 목적을 제공하는 정도를 측정합니다. 응집력이 높으면 구성 요소의 요소가 강력하게 관련되고 특정 함수에 초점을 맞추고 있음을 나타냅니다. 모놀리식 데이터베이스를 분해할 때는 개별 구성 요소 내의 응집력과 구성 요소 간의 결합을 모두 분석해야 합니다. 이 분석은 시스템 무결성과 성능을 유지하면서 데이터베이스를 분류하는 방법에 대해 정보에 입각한 결정을 내리는 데 도움이 됩니다.

다음 이미지는 응집력이 높은 느슨한 결합을 보여줍니다. 데이터베이스의 구성 요소는 함께 작동하여 특정 함수를 수행하며, 변경이 단일 구성 요소에 미치는 영향을 최소화합니다. 이상적인 상태입니다.



다음 이미지는 응집력이 낮은 높은 결합을 보여줍니다. 데이터베이스 구성 요소의 연결이 끊어지고 변경 사항이 다른 구성 요소에 영향을 미칠 가능성이 높습니다.



모놀리식 데이터베이스의 공통 결합 패턴

모놀리식 데이터베이스를 마이크로서비스별 데이터베이스로 분해할 때 일반적으로 발견되는 몇 가지 결합 패턴이 있습니다. 이러한 패턴을 이해하는 것은 성공적인 데이터베이스 현대화 이니셔티브에 매우 중요합니다. 이 섹션에서는 각 패턴, 문제 및 결합 감소 모범 사례를 설명합니다.

구현 결합 패턴

정의: 구성 요소는 코드 및 스키마 수준에서 긴밀하게 상호 연결됩니다. 예를 들어 `customer` 테이블 구조를 수정하면 `order`, `inventory`및 `billing` 서비스에 영향을 미칩니다.

현대화 영향: 각 마이크로서비스에는 자체 전용 데이터베이스 스키마와 데이터 액세스 계층이 필요합니다.

해결 과제:

- 공유 테이블에 대한 변경 사항은 여러 서비스에 영향을 미칩니다.
- 의도하지 않은 부작용의 위험이 높음
- 테스트 복잡성 증가
- 개별 구성 요소를 수정하기 어려움

결합 감소 모범 사례:

- 구성 요소 간의 명확한 인터페이스 정의
- 추상화 계층을 사용하여 구현 세부 정보 숨기기
- 도메인별 스키마 구현

시간적 결합 패턴

정의: 작업은 특정 순서로 실행되어야 합니다. 예를 들어 인벤토리 업데이트가 완료될 때까지 주문 처리를 진행할 수 없습니다.

현대화 영향: 각 마이크로서비스에는 자율적인 데이터 제어가 필요합니다.

해결 과제:

- 서비스 간 동기 종속성 중단

- 성능 병목 현상
- 최적화하기 어려움
- 제한된 병렬 처리

결합 감소 모범 사례:

- 가능한 경우 비동기 처리 구현
- 이벤트 기반 아키텍처 사용
- 적절한 경우 최종 일관성을 위한 설계

배포 결합 패턴

정의: 시스템 구성 요소를 단일 단위로 배포해야 합니다. 예를 들어 결제 처리 로직을 약간 변경하려면 전체 데이터베이스를 재배포해야 합니다.

현대화 영향: 서비스당 독립적인 데이터베이스 배포

해결 과제:

- 고위험 배포
- 제한된 배포 빈도
- 복잡한 롤백 절차

결합 감소 모범 사례:

- 독립적으로 배포 가능한 구성 요소로 분류
- 데이터베이스 샤딩 전략 구현
- 블루-그린 배포 패턴 사용

도메인 결합 패턴

정의: 비즈니스 도메인은 데이터베이스 구조와 로직을 공유합니다. 예를 들어 , `customer` `order`및 `inventory` 도메인은 테이블과 저장 프로시저를 공유합니다.

현대화 영향: 도메인별 데이터 격리

해결 과제:

- 복잡한 도메인 경계
- 개별 도메인을 확장하기 어려움
- 얽힌 비즈니스 규칙

결합 감소 모범 사례:

- 명확한 도메인 경계 식별
- 도메인 컨텍스트별로 데이터 분리
- 도메인별 서비스 구현

모놀리식 데이터베이스의 일반적인 응집 패턴

분해를 위해 데이터베이스 구성 요소를 평가할 때 일반적으로 발견되는 몇 가지 응집 패턴이 있습니다. 이러한 패턴을 이해하는 것은 잘 구성된 데이터베이스 구성 요소를 식별하는 데 매우 중요합니다. 이 섹션에서는 각 패턴, 특성 및 응집력 강화 모범 사례를 설명합니다.

기능적 응집 패턴

정의: 모든 요소는 잘 정의된 단일 함수를 직접 지원하고 수행하는 데 기여합니다. 예를 들어 결제 처리 모듈의 모든 저장 프로시저와 테이블은 결제 관련 작업만 처리합니다.

현대화 영향: 마이크로서비스 데이터베이스 설계에 이상적인 패턴

해결 과제:

- 명확한 기능 경계 식별
- 용도가 혼합된 구성 요소 분리
- 단일 책임 유지

응집력을 강화하는 모범 사례:

- 관련 함수를 함께 그룹화
- 관련 없는 기능 제거
- 명확한 구성 요소 경계 정의

순차 응집 패턴

정의: 한 요소의 출력이 다른 요소의 입력이 됩니다. 예를 들어 주문에 대한 검증 결과는 주문 처리에 적용됩니다.

현대화 영향: 신중한 워크플로 분석 및 데이터 흐름 매핑 필요

해결 과제:

- 단계 간 종속성 관리
- 실패 시나리오 처리
- 프로세스 순서 유지

응집력을 강화하는 모범 사례:

- 명확한 데이터 흐름 문서화
- 적절한 오류 처리 구현
- 단계 간 명확한 인터페이스 설계

통신 응집 패턴

정의: 요소는 동일한 데이터에서 작동합니다. 예를 들어 고객 프로필 관리 함수는 모두 고객 데이터와 함께 작동합니다.

현대화 영향: 서비스 분리를 위한 데이터 경계를 식별하여 모듈 간의 결합 감소

해결 과제:

- 데이터 소유권 결정
- 공유 데이터 액세스 관리
- 데이터 일관성 유지

응집력을 강화하는 모범 사례:

- 명확한 데이터 소유권 정의
- 적절한 데이터 액세스 패턴 구현
- 효과적인 데이터 파티셔닝 설계

절차적 응집 패턴

정의: 요소는 특정 순서로 실행되어야 하지만 기능적으로 관련이 없을 수 있으므로 함께 그룹화됩니다. 예를 들어 주문 처리에서 주문 검증과 사용자 알림을 모두 처리하는 저장 프로시저는 다른 목적을 제공하고 별도의 서비스에서 처리할 수 있더라도 순서대로 발생하기 때문에 간단히 그룹화됩니다.

현대화 영향: 프로세스 흐름을 유지하면서 절차를 신중하게 분리해야 함

해결 과제:

- 분해 후 올바른 프로세스 흐름 유지
- 절차상 종속성과 비교하여 실제 기능 경계 식별

응집력을 강화하는 모범 사례:

- 실행 순서가 아닌 기능적 목적에 따라 절차를 구분합니다.
- 오케스트레이션 패턴을 사용하여 프로세스 흐름 관리
- 복잡한 시퀀스를 위한 워크플로 관리 시스템 구현
- 프로세스 단계를 독립적으로 처리할 수 있는 이벤트 기반 아키텍처 설계

임시 응집 패턴

정의: 요소는 타이밍 요구 사항에 따라 관련이 있습니다. 예를 들어 주문이 이루어지면 재고 확인, 결제 처리, 주문 확인, 배송 알림 등 여러 작업을 함께 실행하여 일관된 주문 상태를 유지해야 합니다.

현대화 영향: 분산 시스템에서 특별한 처리가 필요할 수 있음

해결 과제:

- 분산 서비스 전반의 타이밍 종속성 조정
- 분산 트랜잭션 관리
- 여러 구성 요소에서 프로세스 완료 확인

응집력을 강화하는 모범 사례:

- 적절한 예약 메커니즘 및 제한 시간 구현
- 명확한 시퀀스 처리와 함께 이벤트 기반 아키텍처 사용

- 보상 패턴과의 최종 일관성을 위한 설계
- 분산 트랜잭션을 위한 Saga 패턴 구현

논리적 또는 우연한 응집 패턴

정의: 요소는 관계가 약하거나 의미 있는 관계가 없더라도 동일한 작업을 수행하도록 논리적으로 분류됩니다. 예를 들어 고객 주문 데이터, 창고 인벤토리 수 및 마케팅 이메일 템플릿은 액세스 패턴, 수명 주기 관리 및 규모 조정 요구 사항이 다르더라도 모두 영업 운영과 관련이 있기 때문에 동일한 데이터베이스 스키마에 저장하는 것입니다. 또 다른 예는 주문 결제 처리와 제품 카탈로그 관리를 동일한 데이터베이스 구성 요소 내에서 결합하는 것입니다. 둘 다 운영 요구 사항이 다른 고유한 비즈니스 기능을 제공하더라도 전자 상거래 시스템의 일부이기 때문입니다.

현대화 영향: 리팩터링하거나 재구성해야 함

해결 과제:

- 더 나은 조직 패턴 식별
- 불필요한 종속성 깨기
- 임의로 그룹화된 구성 요소 재구성

응집력을 강화하는 모범 사례:

- 실제 기능 경계 및 비즈니스 도메인을 기반으로 재구성
- 피상적 관계를 기반으로 임의의 그룹 제거
- 비즈니스 역량에 따라 적절한 요소 분리 구현
- 데이터베이스 구성 요소를 특정 운영 요구 사항에 맞게 조정

낮은 결합 및 높은 응집력 구현

모범 사례

다음 모범 사례는 낮은 결합 달성에 도움이 될 수 있습니다.

- 데이터베이스 구성 요소 간의 종속성 최소화
- 구성 요소 상호 작용을 위해 잘 정의된 인터페이스 사용

- 공유 상태 및 글로벌 데이터 구조 방지

다음 모범 사례는 높은 응집력을 달성하는 데 도움이 될 수 있습니다.

- 관련 데이터 및 작업을 함께 그룹화
- 각 구성 요소에 하나의 명확한 책임이 있는지 확인합니다.
- 서로 다른 비즈니스 도메인 간의 명확한 경계 유지

1단계: 데이터 종속성 매핑

데이터 관계를 매핑하고 자연 경계를 식별합니다. 와 같은 도구를 사용하여 개체 관계(ER) 다이어그램에 테이블을 표시하여 데이터베이스를 시각화SchemaSpy할 수 있습니다. 이는 데이터베이스에 대한 정적 분석을 제공하고 데이터베이스 내의 명확한 경계 및 종속성 중 일부를 나타냅니다.

그래프 데이터베이스 또는 Jupiter 노트북에서 데이터베이스 스키마를 내보낼 수도 있습니다. 그런 다음 클러스터링 또는 상호 연결된 구성 요소 알고리즘을 적용하여 자연 경계와 종속성을 식별할 수 있습니다. 와 같은 다른 AWS Partner 도구는 데이터베이스 종속성을 이해하는 데 도움이 될 CAST Imaging수 있습니다.

2단계: 트랜잭션 경계 및 액세스 패턴 분석

트랜잭션 패턴을 분석하여 원자성, 일관성, 격리, 내구성(ACID) 속성을 유지하고 데이터에 액세스하고 데이터를 수정하는 방법을 이해합니다. Oracle Automatic Workload Repository (AWR) 또는와 같은 데이터베이스 분석 및 진단 도구를 사용할 수 있습니다PostgreSQL pg_stat_statements. 이 분석은 데이터베이스에 액세스하는 사용자와 트랜잭션 경계를 이해하는 데 도움이 됩니다. 또한 런타임 시 테이블 간의 응집력과 결합을 이해하는 데 도움이 될 수 있습니다. 와 같은 코드 및 데이터베이스 실행 프로파일을 연결할 수 있는 모니터링 및 프로파일링 도구를 사용할 수도 있습니다Dynatrace AppEngine.

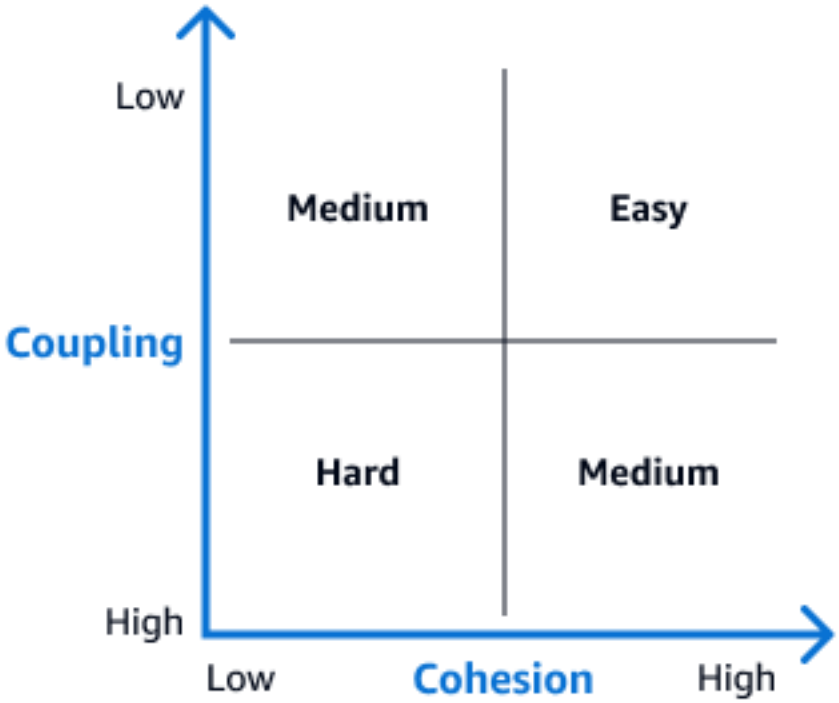
와 같은 AI 도구는 애플리케이션의 기능 및 도메인 경계를 분석하여 도메인 경계를 식별하는 데 도움이 될 vFunction수 있습니다. 는 vFunction 주로 애플리케이션 계층을 분석하지만 인사이트는 애플리케이션과 데이터베이스의 분해를 안내하여 비즈니스 도메인과의 조정을 지원할 수 있습니다.

3단계: 독립형 테이블 식별

두 가지 주요 특성을 보여주는 테이블을 찾습니다.

- 높은 응집력 - 테이블의 내용이 서로 밀접하게 관련되어 있습니다.
- 낮은 결합 - 다른 테이블에 대한 종속성을 최소화합니다.

다음 결합 응집 행렬은 각 테이블의 분리 어려움을 식별하는 데 도움이 될 수 있습니다. 이 매트릭스의 오른쪽 상단 사분면에 나타나는 테이블은 분리하기 가장 쉽기 때문에 초기 분리 작업에 적합한 후보입니다. ER 다이어그램에서 이러한 테이블은 외래 키 관계 또는 기타 종속성이 거의 없습니다. 이러한 테이블을 분리한 후에는 관계가 더 복잡한 테이블로 진행합니다.



> **Note**
>
> 데이터베이스 구조는 종종 애플리케이션 아키텍처를 미러링합니다. 데이터베이스 수준에서 분리하기 쉬운 테이블은 일반적으로 애플리케이션 수준에서 마이크로서비스로 변환하기 쉬운 구성 요소에 해당합니다.

데이터베이스에서 애플리케이션 계층으로 비즈니스 로직 마이그레이션

데이터베이스 저장 프로시저, 트리거 및 함수에서 애플리케이션 계층 서비스로 비즈니스 로직을 마이그레이션하는 것은 모놀리식 데이터베이스를 분해하는 중요한 단계입니다. 이 변환은 서비스 자율성을 개선하고 유지 관리를 간소화하며 확장성을 향상시킵니다. 이 섹션에서는 데이터베이스 로직 분석, 마이그레이션 전략 계획, 비즈니스 연속성을 유지하면서 변환 구현에 대한 지침을 제공합니다. 또한 효과적인 롤백 계획 수립에 대해서도 설명합니다.

이 섹션은 다음 주제를 포함합니다:

- 1단계: 비즈니스 로직 분석
- 2단계: 비즈니스 로직 분류
- 3단계: 비즈니스 로직 마이그레이션
- 비즈니스 로직에 대한 롤백 전략

1단계: 비즈니스 로직 분석

모놀리식 데이터베이스를 현대화할 때는 먼저 기존 데이터베이스 로직에 대한 포괄적인 분석을 수행해야 합니다. 이 단계에서는 세 가지 기본 범주에 중점을 둡니다.

- 저장 프로시저에는 데이터 조작 로직, 비즈니스 규칙, 검증 검사 및 계산을 비롯한 중요한 비즈니스 운영이 포함되는 경우가 많습니다. 애플리케이션 비즈니스 로직의 핵심 구성 요소로서 신중한 분해가 필요합니다. 예를 들어 금융 조직의 저장 프로시저는 관심 계산, 계정 조정 및 규정 준수 검사를 처리할 수 있습니다.
- 트리거는 감사 추적, 데이터 검증, 계산 및 테이블 간 일관성을 처리하는 주요 데이터베이스 구성 요소입니다. 예를 들어 소매 조직은 트리거를 사용하여 주문 처리 시스템 전체에서 인벤토리 업데이트를 관리할 수 있으며, 이는 자동화된 데이터베이스 작업의 복잡성을 보여줍니다.
- 데이터베이스의 함수는 주로 데이터 변환, 계산 및 조회 작업을 관리합니다. 종종 여러 절차와 애플리케이션에 포함됩니다. 예를 들어 의료 기관은 함수를 사용하여 환자 데이터를 정규화하거나 의료 코드를 조회할 수 있습니다.

각 범주는 데이터베이스 계층 내에 포함된 비즈니스 로직의 다양한 측면을 나타냅니다. 애플리케이션 계층으로 마이그레이션하려면 각 항목을 신중하게 평가하고 계획해야 합니다.

이 분석 단계에서 고객은 일반적으로 세 가지 중요한 문제에 직면합니다. 먼저 중첩된 프로시저 호출, 교차 스키마 참조 및 암시적 데이터 종속성을 통해 복잡한 종속성이 나타납니다. 둘째, 특히 다단계 트랜잭션을 처리하고 분산 시스템에서 데이터 일관성을 유지할 때 트랜잭션 관리가 중요합니다. 셋째, 성능 고려 사항을 신중하게 평가해야 합니다. 특히 배치 처리 작업, 대량 데이터 업데이트 및 현재 데이터에 가까워지면 도움이 되는 실시간 계산의 경우 더욱 그렇습니다.

이러한 문제를 효과적으로 해결하기 위해 초기 분석에 AWS Schema Conversion Tool (AWS SCT)를 사용한 다음 세부 종속성 매핑 도구를 사용할 수 있습니다. 이 접근 방식은 데이터베이스 로직의 전체 범위를 이해하고 분해 중에 비즈니스 연속성을 유지하는 포괄적인 마이그레이션 전략을 만드는 데 도움이 됩니다.

이러한 구성 요소와 과제를 철저히 이해하면 현대화 여정을 더 잘 계획하고 마이크로서비스 기반 아키텍처로 마이그레이션하는 동안 우선순위를 지정할 요소에 대해 정보에 입각한 결정을 내릴 수 있습니다.

데이터베이스 코드 구성 요소를 분석할 때 각 저장 프로시저, 트리거 및 함수에 대한 포괄적인 설명서를 생성합니다. 먼저 구현하는 비즈니스 규칙을 포함하여 목적과 핵심 기능을 명확하게 설명합니다. 모든 입력 및 출력 파라미터를 자세히 설명하고 데이터 유형과 유효한 범위를 기록해 둡니다. 다른 데이터베이스 객체, 외부 시스템 및 다운스트림 프로세스에 대한 종속성을 매핑합니다. 트랜잭션 경계 및 격리 요구 사항을 명확하게 정의하여 데이터 무결성을 유지합니다. 응답 시간 요구 사항 및 리소스 사용률 패턴을 포함하여 성능 기대치를 문서화합니다. 마지막으로 사용량 패턴을 분석하여 최대 부하, 실행 빈도 및 중요한 비즈니스 기간을 파악합니다.

2단계: 비즈니스 로직 분류

효과적인 데이터베이스 분해를 위해서는 복잡성, 비즈니스 영향, 종속성, 사용 패턴 및 마이그레이션 난이도와 같은 주요 차원에서 데이터베이스 로직을 체계적으로 분류해야 합니다. 이 분류는 고위험 구성 요소를 식별하고, 테스트 요구 사항을 결정하고, 마이그레이션 우선순위를 설정하는 데 도움이 됩니다. 예를 들어 비즈니스에 큰 영향을 미치고 자주 사용하는 복잡한 저장 프로시저에는 신중한 계획과 광범위한 테스트가 필요합니다. 그러나 종속성이 최소인 단순하고 거의 사용되지 않는 함수는 초기 마이그레이션 단계에 적합할 수 있습니다.

이 구조화된 접근 방식은 시스템 안정성을 유지하면서 비즈니스 중단을 최소화하는 균형 잡힌 마이그레이션 로드맵을 생성합니다. 이러한 상호 관계를 이해하면 분해 작업의 순서를 개선하고 리소스를 적절하게 할당할 수 있습니다.

3단계: 비즈니스 로직 마이그레이션

비즈니스 로직을 분석하고 분류한 후에는 마이그레이션해야 합니다. 모놀리식 데이터베이스에서 비즈니스 로직을 마이그레이션할 때는 데이터베이스 로직을 애플리케이션 계층으로 이동하거나 비즈니스 로직을 마이크로서비스의 일부인 다른 데이터베이스로 이동하는 두 가지 접근 방식이 있습니다.

비즈니스 로직을 애플리케이션으로 마이그레이션하면 데이터베이스 테이블은 데이터만 저장하고 데이터베이스에는 비즈니스 로직이 포함되지 않습니다. 이것이 권장되는 접근 방식입니다. Amazon Q Developer 또는와 같은 Ispirer 또는 생성형 AI 도구를 사용하여 JavaKiro로의 변환과 같은 애플리케이션 계층의 데이터베이스 비즈니스 로직을 변환할 수 있습니다. 자세한 내용은 더 빠른 혁신과 유연성을 위해 데이터베이스에서 애플리케이션으로 비즈니스 로직 마이그레이션(AWS 블로그 게시물)을 참조하세요.

비즈니스 로직을 다른 데이터베이스로 마이그레이션하는 경우 AWS Schema Conversion Tool (AWS SCT)를 사용하여 기존 데이터베이스 스키마 및 코드 객체를 대상 데이터베이스로 변환할 수 있습니다. Amazon DynamoDB, Amazon Aurora 및 Amazon Redshift와 같이 특별히 구축된 AWS 데이터베이스 서비스를 지원합니다. 포괄적인 평가 보고서와 자동화된 변환 기능을 제공하여는 전환 프로세스를 간소화하여 성능 및 확장성 향상을 위해 새 데이터베이스 구조를 최적화하는 데 집중할 수 있도록 AWS SCT 지원합니다. 현대화 프로젝트를 진행하면서는 단계별 접근 방식을 지원하기 위해 증분 변환을 AWS SCT 처리할 수 있으므로 데이터베이스 변환의 각 단계를 검증하고 미세 조정할 수 있습니다.

비즈니스 로직에 대한 롤백 전략

분해 전략의 두 가지 중요한 측면은 이전 버전과의 호환성을 유지하고 포괄적인 롤백 절차를 구현하는 것입니다. 이러한 요소는 함께 작동하여 전환 기간 동안 작업을 보호하는 데 도움이 됩니다. 이 섹션에서는 분해 프로세스 중에 호환성을 관리하고 잠재적 문제를 방지하는 효과적인 비상 롤백 기능을 설정하는 방법을 설명합니다.

이전 버전과의 호환성 유지

데이터베이스 분해 중에는 원활한 전환을 위해 이전 버전과의 호환성을 유지하는 것이 필수적입니다. 새로운 기능을 점진적으로 구현하면서 기존 데이터베이스 절차를 일시적으로 유지합니다. 버전 관리를 사용하여 모든 변경 사항을 추적하고 여러 데이터베이스 버전을 동시에 관리할 수 있습니다. 소스 시스템과 대상 시스템이 모두 안정적으로 작동해야 하는 장기 공존 기간을 계획합니다. 이렇게 하면 레거시 구성 요소를 사용 중지하기 전에 새 시스템을 테스트하고 검증할 수 있습니다. 이 접근 방식은 비즈니스 중단을 최소화하고 필요한 경우 롤백을 위한 안전망을 제공합니다.

긴급 롤백 계획

안전한 데이터베이스 분해를 위해서는 포괄적인 롤백 전략이 필수적입니다. 코드에 기능 플래그를 구현하여 어떤 버전의 비즈니스 로직이 활성 상태인지 제어합니다. 이를 통해 배포 변경 없이 새 구현과 원래 구현 간에 즉시 전환할 수 있습니다. 이 접근 방식은 전환을 세밀하게 제어하고 문제가 발생할 경우 신속하게 롤백하는 데 도움이 됩니다. 원래 로직을 확인된 백업으로 유지하고 트리거, 책임 및 복구 단계를 지정하는 자세한 롤백 절차를 유지합니다.

다양한 조건에서 이러한 롤백 시나리오를 정기적으로 테스트하여 효과를 검증하고 팀이 비상 절차에 익숙한지 확인합니다. 기능 플래그는 특정 사용자 그룹 또는 트랜잭션에 대한 새 기능을 선택적으로 활성화하여 점진적인 롤아웃도 가능하게 합니다. 이렇게 하면 전환 중에 추가적인 위험 완화 계층이 제공됩니다.

데이터베이스 분해 중 테이블 관계 분리

이 섹션에서는 모놀리식 데이터베이스 분해 중에 복잡한 테이블 관계 및 JOIN 작업을 분석하는 방법에 대한 지침을 제공합니다. 테이블 조인은 둘 이상의 테이블에 있는 행을 둘 사이의 관련 열을 기반으로 결합합니다. 이러한 관계를 분리하는 목적은 마이크로서비스 전반에서 데이터 무결성을 유지하면서 테이블 간의 높은 결합을 줄이는 것입니다.

이 섹션은 다음 주제를 포함합니다:

- 비정규화 전략
- Reference-by-key 전략
- CQRS 패턴
- 이벤트 기반 데이터 동기화
- 테이블 조인에 대한 대안 구현
- 시나리오 기반 예제

비정규화 전략

비정규화는 테이블 간에 데이터를 결합하거나 복제하여 의도적으로 중복성을 도입하는 데이터베이스 설계 전략입니다. 큰 데이터베이스를 작은 데이터베이스로 분리할 때는 서비스 간에 일부 데이터를 복제하는 것이 합리적일 수 있습니다. 예를 들어 마케팅 서비스와 주문 서비스 모두에 이름 및 이메일 주소와 같은 기본 고객 세부 정보를 저장하면 지속적인 교차 서비스 조회가 필요하지 않습니다. 마케팅 서비스는 캠페인 타겟팅을 위해 고객 기본 설정과 연락처 정보가 필요할 수 있지만 주문 서비스는 주문 처리 및 알림을 위해 동일한 데이터가 필요합니다. 이로 인해 일부 데이터 중복이 생성되지만 서비스 성능과 독립성이 크게 향상되어 마케팅 팀이 실시간 고객 서비스 조회에 의존하지 않고 캠페인을 운영할 수 있습니다.

비정규화를 구현할 때는 데이터 액세스 패턴을 신중하게 분석하여 식별하는 자주 액세스하는 필드에 집중합니다. Oracle AWR 보고서 또는와 같은 도구를 사용하여 일반적으로 함께 검색되는 데이터를 pg_stat_statements파악할 수 있습니다. 도메인 전문가는 자연 데이터 그룹에 대한 귀중한 인사이트를 제공할 수도 있습니다. 비정규화는 all-or-nothing 않는 접근 방식이 아니라 시스템 성능을 확실히 개선하거나 복잡한 종속성을 줄이는 중복 데이터만 사용한다는 점을 기억하세요.

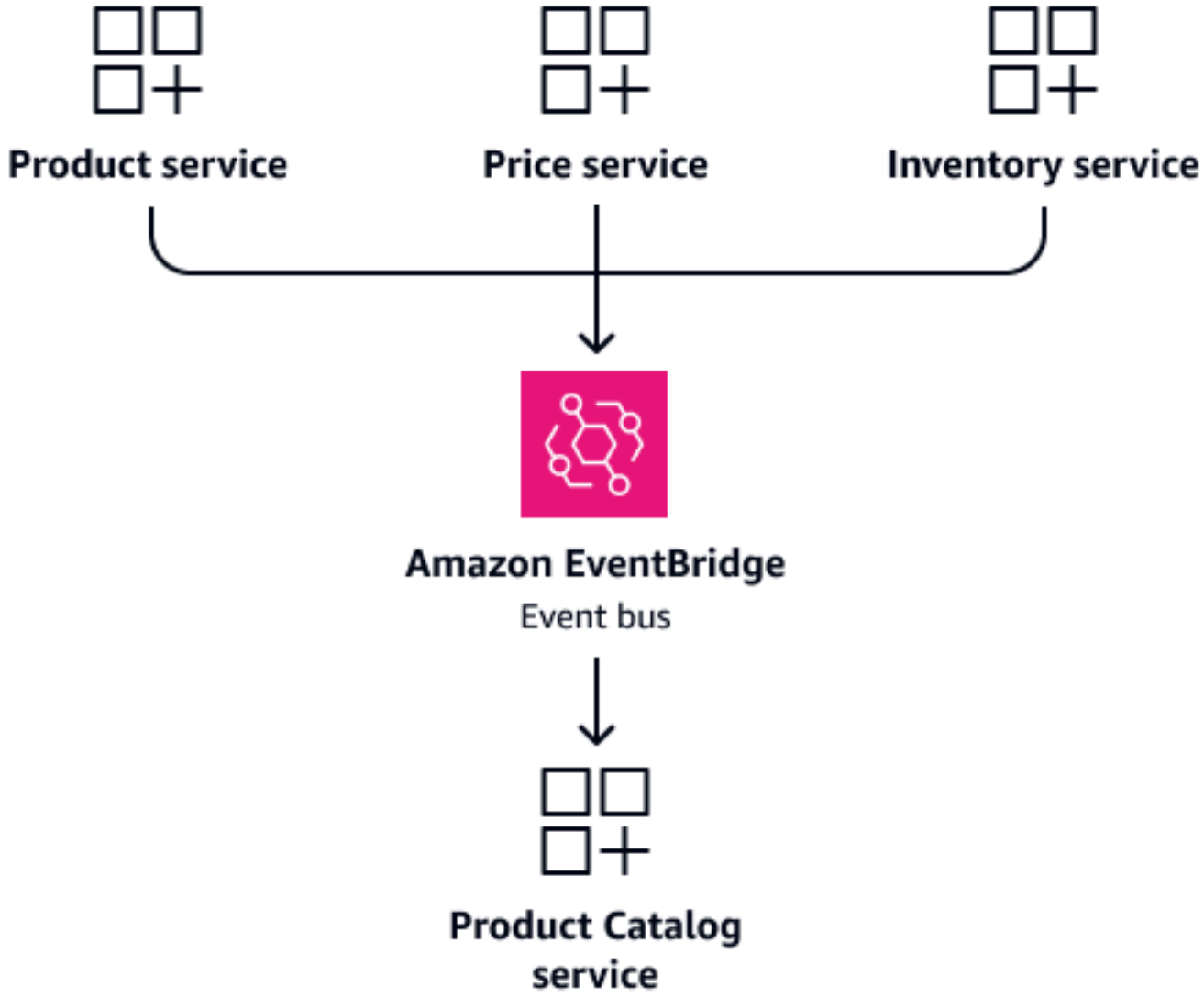
Reference-by-key 전략

reference-by-key 전략은 실제 관련 데이터를 저장하지 않고 고유한 키를 통해 엔터티 간의 관계를 유지하는 데이터베이스 설계 패턴입니다. 최신 마이크로서비스는 기존의 외래 키 관계 대신 관련 데이터의 고유 식별자만 저장하는 경우가 많습니다. 예를 들어 주문 테이블에 모든 고객 세부 정보를 보관하는 대신 주문 서비스는 고객 ID만 저장하고 필요한 경우 API 직접 호출을 통해 추가 고객 정보를 검색합니다. 이 접근 방식은 서비스 독립성을 유지하면서 관련 데이터에 대한 액세스를 보장합니다.

CQRS 패턴

명령 쿼리 책임 분리(CQRS) 패턴은 데이터 스토어의 읽기 및 쓰기 작업을 구분합니다. 이 패턴은 고성능 요구 사항, 특히 비대칭 읽기/쓰기 로드가 있는 복잡한 시스템에서 특히 유용합니다. 애플리케이션에 여러 소스의 데이터가 자주 필요한 경우 복잡한 조인 대신 전용 CQRS 모델을 생성할 수 있습니다. 예를 들어 모든 요청에 `Product`, `Pricing`및 `Inventory` 테이블을 조인하는 대신 필요한 데이터가 포함된 통합 `Product Catalog` 테이블을 유지 관리합니다. 이 접근 방식의 이점은 추가 테이블의 비용을 능가할 수 있습니다.

`Product`, `Price`및 `Inventory` 서비스에 제품 정보가 자주 필요한 시나리오를 생각해 보세요. 공유 테이블에 직접 액세스하도록 이러한 서비스를 구성하는 대신 전용 `Product Catalog` 서비스를 생성합니다. 이 서비스는 통합 제품 정보가 포함된 자체 데이터베이스를 유지합니다. 제품 관련 쿼리에 대한 단일 정보 소스 역할을 합니다. 제품 세부 정보, 가격 또는 인벤토리 수준이 변경되면 각 서비스는 이벤트를 게시하여 `Product Catalog` 서비스를 업데이트할 수 있습니다. 이렇게 하면 서비스 독립성을 유지하면서 데이터 일관성을 유지할 수 있습니다. 다음 이미지는 Amazon EventBridge가 이벤트 버스 역할을 하는이 구성을 보여줍니다.



다음 섹션[이벤트 기반 데이터 동기화]인에서 설명한 대로 이벤트를 통해 CQRS 모델을 최신 상태로 유지합니다. 제품 세부 정보, 가격 또는 인벤토리 수준이 변경되면 각 서비스가 이벤트를 게시합니다. `Product Catalog` 서비스는 이러한 이벤트를 구독하고 통합 보기를 업데이트합니다. 이는 복잡한 조인 없이 빠른 읽기를 제공하며 서비스 독립성을 유지합니다.

이벤트 기반 데이터 동기화

이벤트 기반 데이터 동기화는 데이터 변경 사항이 캡처되고 이벤트로 전파되는 패턴으로, 이를 통해 다양한 시스템 또는 구성 요소가 동기화된 데이터 상태를 유지할 수 있습니다. 데이터가 변경되면 모든 관련 데이터베이스를 즉시 업데이트하는 대신 이벤트를 게시하여 구독한 서비스에 알립니다. 예를 들어 고객이 `Customer` 서비스의 배송 주소를 변경하면 `CustomerUpdated` 이벤트가 각 `Order` 서비스의 일정에 따라 서비스 및 `Delivery` 서비스에 대한 업데이트를 시작합니다. 이 접근 방식은 강체 테이

블 조인을 유연하고 확장 가능한 이벤트 기반 업데이트로 대체합니다. 일부 서비스에는 잠시 오래된 데이터가 있을 수 있지만 단점은 시스템 확장성과 서비스 독립성이 향상된다는 것입니다.

테이블 조인에 대한 대안 구현

일반적으로 마이그레이션 및 검증이 더 간단하므로 읽기 작업으로 데이터베이스 분해를 시작합니다. 읽기 경로가 안정되면 더 복잡한 쓰기 작업을 처리합니다. 중요한 고성능 요구 사항의 경우 CQRS 패턴 구현을 고려하세요. 쓰기를 위해 다른 데이터베이스를 유지하면서 읽기에 최적화된 별도의 데이터베이스를 사용합니다.

교차 서비스 호출을 위한 재시도 로직을 추가하고 적절한 캐싱 계층을 구현하여 복원력이 뛰어난 시스템을 구축합니다. 서비스 상호 작용을 면밀히 모니터링하고 데이터 일관성 문제에 대한 알림을 설정합니다. 최종 목표는 모든 곳에서 완벽한 일관성이 아니라 비즈니스 요구 사항에 적합한 데이터 정확도를 유지하면서 잘 작동하는 독립적인 서비스를 만드는 것입니다.

마이크로서비스의 분리된 특성은 데이터 관리에 다음과 같은 새로운 복잡성을 도입합니다.

- 데이터가 배포됩니다. 이제 데이터는 독립 서비스에서 관리하는 별도의 데이터베이스에 상주합니다.
- 서비스 간 실시간 동기화는 종종 비실용적이므로 최종 일관성 모델이 필요합니다.
- 단일 데이터베이스 트랜잭션 내에서 이전에 발생한 작업은 이제 여러 서비스에 걸쳐 있습니다.

이러한 문제를 해결하려면 다음을 수행합니다.

- 이벤트 기반 아키텍처 구현 - 메시지 대기열 및 이벤트 게시를 사용하여 서비스 전체에 데이터 변경 사항을 전파합니다. 자세한 내용은 서버리스 랜드에서 이벤트 중심 아키텍처 구축을 참조하세요.
- Saga 오케스트레이션 패턴 채택 -이 패턴은 분산 트랜잭션을 관리하고 서비스 전반에서 데이터 무결성을 유지하는 데 도움이 됩니다. 자세한 내용은 AWS 블로그에서 Saga 오케스트레이션 패턴을 사용하여 서버리스 분산 애플리케이션 구축을 참조하세요.
- 장애에 대한 설계 - 재시도 메커니즘, 회로 차단기 및 보상 트랜잭션을 통합하여 네트워크 문제 또는 서비스 장애를 처리합니다.
- 버전 스탬핑 사용 - 데이터 버전을 추적하여 충돌을 관리하고 최신 업데이트가 적용되었는지 확인합니다.
- 정기적인 조정 - 정기적인 데이터 동기화 프로세스를 구현하여 불일치를 포착하고 수정합니다.

시나리오 기반 예제

다음 스키마 예제에는 테이블과 Customer 테이블이라는 두 개의 Order 테이블이 있습니다.

```
-- Customer table
CREATE TABLE customer (
    customer_id INT PRIMARY KEY,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    email VARCHAR(255),
    phone VARCHAR(20),
    address TEXT,
    created_at TIMESTAMP
);

-- Order table
CREATE TABLE order (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES customers(id)
);
```

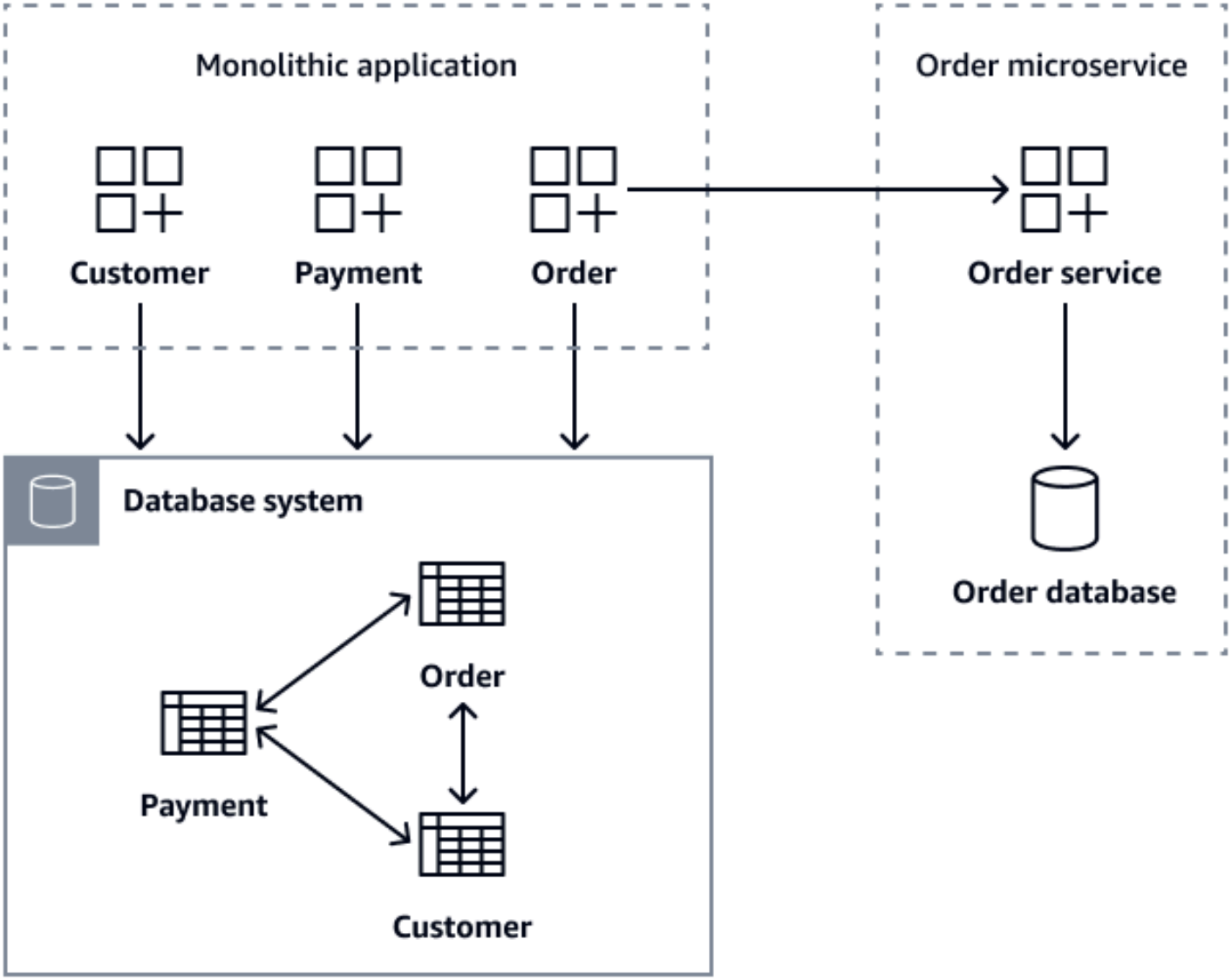
다음은 비정규화된 접근 방식을 사용하는 방법의 예입니다.

```
CREATE TABLE order (
    order_id INT PRIMARY KEY,
    customer_id INT,                    -- Reference only
    customer_first_name VARCHAR(100), -- Denormalized
    customer_last_name VARCHAR(100),  -- Denormalized
    customer_email VARCHAR(255),      -- Denormalized
    order_date TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(50)
);
```

새 Order 테이블에는 정규화되지 않은 고객 이름과 이메일 주소가 있습니다. customer_id가 참조되고 Customer 테이블에 외래 키 제약이 없습니다. 다음은이 비정규화된 접근 방식의 이점입니다.

- Order 서비스는 고객 세부 정보와 함께 주문 기록을 표시할 수 있으며 Customer 마이크로서비스에 대한 API 호출이 필요하지 않습니다.
- Customer 서비스가 다운된 경우 Order 서비스는 완전히 작동합니다.
- 주문 처리 및 보고에 대한 쿼리는 더 빠르게 실행됩니다.

다음 다이어그램은 Order 마이크로서비스에 대한 getOrder(customer_id), getOrder(order_id), getCustomerOders(customer_id)및 createOrder(Order order) API 호출을 사용하여 주문 데이터를 검색하는 모놀리식 애플리케이션을 보여줍니다.



마이크로서비스 마이그레이션 중에 이전 안전 조치로 모놀리식 데이터베이스의 Order 테이블을 유지하여 레거시 애플리케이션이 계속 작동하도록 할 수 있습니다. 그러나 모든 새로운 주문 관련 작업은 백업으로 레거시 데이터베이스에 쓰는 동시에 자체 데이터베이스를 유지 관리하는 Order 마이크

로서비스 API를 통해 라우팅되는 것이 중요합니다. 이 이중 쓰기 패턴은 안전망을 제공합니다. 이를 통해 시스템 안정성을 유지하면서 점진적으로 마이그레이션할 수 있습니다. 모든 고객이 새 마이크로서비스로 성공적으로 마이그레이션한 후에는 모놀리식 데이터베이스의 레거시 `Order` 테이블을 더 이상 사용하지 않을 수 있습니다. 모놀리식 애플리케이션과 데이터베이스를 별도의 `Customer` `Order` 마이크로서비스로 분해한 후 데이터 일관성을 유지하는 것이 주요 과제가 됩니다.

데이터베이스 분해 모범 사례

모놀리식 데이터베이스를 분해할 때 조직은 진행 상황을 추적하고, 시스템 지식을 유지하고, 새로운 문제를 해결하기 위한 명확한 프레임워크를 설정해야 합니다. 이 섹션에서는 분해 성공을 측정하고, 중요한 문서를 유지 관리하고, 지속적인 개선 프로세스를 구현하고, 일반적인 문제를 해결하는 모범 사례를 제공합니다. 이러한 지침을 이해하고 준수하면 데이터베이스 분해 작업이 운영 중단과 기술 부채를 최소화하면서 의도한 이점을 제공할 수 있습니다.

이 섹션은 다음 주제를 포함합니다:

- 성공 측정
- 문서화 요구 사항
- 지속적인 개선 전략
- 데이터베이스 분해의 일반적인 문제 해결

성공 측정

기술, 운영 및 비즈니스 지표를 혼합하여 분해 성공을 추적합니다. 기술적으로 쿼리 응답 시간, 시스템 가동 시간 개선 및 배포 빈도 증가를 모니터링합니다. 운영 측면에서 인시던트 감소, 문제 해결 속도 및 리소스 사용률 개선을 측정합니다. 개발을 위해 기능 구현 속도, 릴리스 주기 가속화 및 팀 간 종속성 감소를 추적합니다. 비즈니스에 미치는 영향으로 인해 운영 비용이 절감되고 time-to-market 단축되며 고객 만족도가 향상됩니다. 이러한 지표는 범위 단계에서 종종 정의됩니다. 자세한 내용은이 가이드의 데이터베이스 분해 범위 및 요구 사항 정의를 참조하세요.

문서화 요구 사항

명확한 서비스 경계, 데이터 흐름 및 인터페이스 사양을 사용하여 up-to-date 시스템 아키텍처 설명서를 유지 관리합니다. 아키텍처 결정 레코드(ADRs)를 사용하여 컨텍스트, 결과 및 고려된 대안을 포함한 주요 기술적 결정을 캡처합니다. 예를 들어 특정 서비스를 먼저 분리한 이유 또는 특정 데이터 일관성 절충이 이루어진 방법을 문서화합니다.

월별 아키텍처 검토를 예약하여 성능 추세, 보안 규정 준수, 교차 서비스 종속성 등의 주요 지표를 통해 시스템 상태를 평가합니다. 통합 문제 및 운영 문제에 대한 개발 팀의 피드백을 포함합니다. 이 정기 검토 주기는 새로운 문제를 조기에 식별하고 분해 노력이 비즈니스 목표에 부합하는지 검증하는 데 도움이 됩니다.

지속적인 개선 전략

데이터베이스 분해를 일회성 프로젝트가 아닌 반복 프로세스로 취급합니다. 시스템 성능 지표 및 서비스 상호 작용을 모니터링하여 최적화 기회를 식별합니다. 분기마다 운영 영향 및 유지 관리 비용을 기반으로 기술 부채를 해결하는 데 우선순위를 둡니다. 예를 들어 자주 수행되는 데이터베이스 작업을 자동화하고 모니터링 범위를 개선하며 학습된 패턴을 기반으로 배포 절차를 세분화합니다.

데이터베이스 분해의 일반적인 문제 해결

성능 최적화에는 다면적 접근 방식이 필요합니다. 서비스 경계에서 전략적 캐싱을 구현하고, 실제 사용량을 기반으로 쿼리 패턴을 최적화하고, 주요 지표를 지속적으로 모니터링합니다. 추세를 분석하고 개입을 위한 명확한 임계값을 설정하여 성능 병목 현상을 사전에 해결합니다.

데이터 일관성 문제에는 신중한 아키텍처 선택이 필요합니다. 교차 서비스 업데이트를 위한 이벤트 기반 패턴을 구현하고 복잡한 트랜잭션에 Saga 오케스트레이션 패턴을 사용합니다. 명확한 서비스 경계를 정의하고 비즈니스 요구 사항이 허용하는 경우 최종 일관성을 수락합니다. 일관성과 서비스 자율성 간의 이러한 균형은 성공적인 분해에 매우 중요합니다.

운영 우수성을 위해서는 서비스 전반에 걸쳐 일상적인 작업과 표준화된 절차를 자동화해야 합니다. 명확한 알림 임계값으로 포괄적인 모니터링을 유지하고 새로운 패턴과 도구를 위한 정기적인 팀 교육에 투자합니다. 이러한 체계적인 운영 접근 방식은 복잡성을 관리하면서 안정적인 서비스 제공을 촉진합니다.

데이터베이스 분해에 대한 FAQ

이 포괄적인 FAQ 섹션에서는 데이터베이스 분해 프로젝트를 수행할 때 조직이 직면하는 가장 일반적인 질문과 문제를 다룹니다. 초기 범위 및 요구 사항 정의부터 저장 프로시저 마이그레이션에 이르기까지 이러한 질문은 팀이 데이터베이스 현대화 여정을 성공적으로 탐색하는 데 도움이 되는 실용적인 인사이트와 전략적 접근 방식을 제공합니다. 계획 단계에 있든 이미 분해 전략을 실행 중이든 이러한 답변은 일반적인 위험을 피하고 최적의 결과를 위한 모범 사례를 구현하는 데 도움이 될 수 있습니다.

이 섹션은 다음 주제를 포함합니다:

- 범위 및 요구 사항 정의FAQs
- 데이터베이스 액세스 제어FAQs
- 응집력 및 결합 분석에 대한 FAQs
- 비즈니스 로직을 애플리케이션 계층으로 마이그레이션하는 방법에 대한 FAQs

범위 및 요구 사항 정의FAQs

이 가이드의 데이터베이스 분해의 범위 및 요구 사항 정의 섹션에서는 상호 작용을 분석하고, 종속성을 매핑하고, 성공 기준을 설정하는 방법을 설명합니다. 이 FAQ 섹션에서는 프로젝트 경계 설정 및 관리에 대한 주요 질문을 다룹니다. 불분명한 기술적 제약, 부서별 요구 사항 충돌 또는 변화하는 비즈니스 요구 사항을 처리하든 관계없이 이러한 FAQs는 균형 잡힌 접근 방식을 유지하기 위한 실용적인 지침을 제공합니다.

이 섹션에는 다음 질문이 포함되어 있습니다.

- 초기 범위 정의는 얼마나 상세해야 합니까?
- 프로젝트를 시작한 후 추가 종속성을 발견하면 어떻게 됩니까?
- 요구 사항이 충돌하는 여러 부서의 이해관계자를 처리하려면 어떻게 해야 합니까?
- 설명서가 좋지 않거나 오래된 경우 기술적 제약 조건을 평가하는 가장 좋은 방법은 무엇입니까?
- 즉각적인 비즈니스 요구 사항과 장기적인 기술 목표의 균형을 맞추려면 어떻게 해야 하나요?
- 자동 이해관계자의 중요한 요구 사항이 누락되지 않도록 하려면 어떻게 해야 합니까?
- 이러한 권장 사항이 모놀리식 메인프레임 데이터베이스에 적용되나요?

초기 범위 정의는 얼마나 상세해야 합니까?

고객의 요구 사항에서 벗어나면 시스템 경계와 중요한 종속성을 식별하는 동시에 검색 유연성을 유지할 수 있도록 프로젝트 범위를 충분히 세부적으로 정의합니다. 시스템 인터페이스, 주요 이해관계자, 주요 기술적 제약 조건을 포함한 필수 요소를 매핑합니다. 측정 가능한 값을 제공하는 시스템의 위험도가 낮은 제한된 부분을 선택하여 작게 시작합니다. 이 접근 방식은 팀이 더 복잡한 구성 요소를 처리하기 전에 전략을 학습하고 조정하는 데 도움이 됩니다.

분해 작업을 주도하는 중요한 비즈니스 요구 사항을 문서화하되 구현 중에 변경될 수 있는 세부 정보를 과도하게 지정하지 마세요. 이러한 균형 잡힌 접근 방식을 통해 팀은 현대화 여정 중에 나타나는 새로운 인사이트와 문제에 적응하면서 명확성을 바탕으로 앞으로 나아갈 수 있습니다.

프로젝트를 시작한 후 추가 종속성을 발견하면 어떻게 됩니까?

프로젝트가 진행됨에 따라 추가 종속성을 찾아낼 것으로 예상합니다. 실시간 종속성 로그를 유지하고 정기적인 범위 검토를 수행하여 타임라인 및 리소스에 미치는 영향을 평가합니다. 명확한 변경 관리 프로세스를 구현하고 프로젝트 계획에 버퍼 시간을 포함하여 예상치 못한 발견을 처리합니다. 목표는 변경을 방지하는 것이 아니라 효과적으로 관리하는 것입니다. 이를 통해 팀은 프로젝트 추진력을 유지하면서 빠르게 적응할 수 있습니다.

요구 사항이 충돌하는 여러 부서의 이해관계자를 처리하려면 어떻게 해야 합니까?

비즈니스 가치 및 시스템 영향을 기반으로 하는 명확한 우선순위 지정을 통해 충돌하는 부서별 요구 사항을 처리합니다. 경영진의 후원을 확보하여 주요 결정을 내리고 충돌을 신속하게 해결합니다. 정기적인 이해관계자 조정 회의를 예약하여 장단점을 논의하고 투명성을 유지합니다. 모든 결정과 근거를 문서화하여 명확한 커뮤니케이션을 촉진하고 프로젝트 추진력을 유지합니다. 부서별 선호도보다는 정량화 가능한 비즈니스 이점에 대한 논의에 집중합니다.

설명서가 좋지 않거나 오래된 경우 기술적 제약 조건을 평가하는 가장 좋은 방법은 무엇입니까?

설명서가 좋지 않은 경우 기존 분석을 최신 AI 도구와 결합하세요. 대규모 언어 모델(LLMs)을 사용하여 코드 리포지토리, 로그 및 기존 설명서를 분석하여 패턴과 잠재적 제약 조건을 식별합니다. 숙련된 개발자와 데이터베이스 아키텍트를 인터뷰하여 AI 결과를 검증하고 문서화되지 않은 제약 조건을 파악합니다. 시스템 동작을 관찰하고 잠재적 문제를 예측하기 위해 향상된 AI 기능을 갖춘 모니터링 도구를 배포합니다.

가정을 검증하는 소규모 기술 실험을 생성합니다. AI 기반 테스트 도구를 사용하여 프로세스를 가속화할 수 있습니다. AI 지원 업데이트를 통해 지속적으로 개선할 수 있는 지식 기반에 조사 결과를 문서화합니다. 복잡한 영역에 대한 주제 전문가의 참여를 고려하고 AI 페어 프로그래밍 도구를 사용하여 분석 및 문서화 작업을 가속화하세요.

즉각적인 비즈니스 요구 사항과 장기적인 기술 목표의 균형을 맞추려면 어떻게 해야 하나요?

즉각적인 비즈니스 요구 사항을 장기적인 기술 목표에 맞게 조정하는 단계별 프로젝트 로드맵을 생성합니다. 이해관계자의 신뢰를 구축할 수 있도록 가시적 가치를 조기에 제공하는 빠른 성과를 식별합니다. 분해를 명확한 마일스톤으로 나눕니다. 각는 아키텍처 목표를 향해 나아가면서 측정 가능한 비즈니스 이점을 제공해야 합니다. 정기적인 로드맵 검토 및 조정을 통해 긴급한 비즈니스 요구 사항을 해결할 수 있는 유연성을 유지합니다.

자동 이해관계자의 중요한 요구 사항이 누락되지 않도록 하려면 어떻게 해야 합니까?

다운스트림 시스템 소유자 및 간접 사용자를 포함하여 조직 전체의 모든 잠재적 이해관계자를 매핑합니다. 구조화된 인터뷰, 워크숍 및 정기 검토 세션을 통해 여러 피드백 채널을 생성합니다. proof-of-concepts과 프로토타입을 구축하여 요구 사항을 구체화하고 의미 있는 토론을 촉진합니다. 예를 들어 시스템 종속성을 보여주는 간단한 대시보드는 처음에는 명확하지 않았던 숨겨진 이해관계자와 요구 사항을 공개하는 경우가 많습니다.

음성 및 조용한 이해관계자와 함께 정기적인 검증 세션을 수행하고 모든 관점이 캡처되도록 합니다. 중요한 인사이트는 계획 회의에서 가장 큰 목소리가 아닌 일상적인 운영에 가장 가까운 사람들에서 비롯되는 경우가 많습니다.

이러한 권장 사항이 모놀리식 메인프레임 데이터베이스에 적용되나요?

이 가이드에 설명된 방법론은 모놀리식 메인프레임 데이터베이스 분해에도 적용됩니다. 이러한 데이터베이스의 주요 과제는 다양한 이해관계자의 요구 사항을 관리하는 것입니다. 이 가이드의 기술 권장 사항은 모놀리식 메인프레임 데이터베이스에 적용될 수 있습니다. 메인프레임에 온라인 트랜잭션 처리(OLTP) 데이터베이스와 같은 관계형 데이터베이스가 있는 경우 많은 권장 사항이 적용됩니다. 비즈니스 보고서를 생성하는 데 사용되는 데이터베이스와 같은 온라인 분석 처리(OLAP) 데이터베이스의 경우 일부 권장 사항만 적용됩니다.

데이터베이스 액세스 제어FAQs

데이터베이스 래퍼 서비스 패턴을 사용하여 데이터베이스 액세스를 제어하는 방법은이 가이드의 분해 중 데이터베이스 액세스 제어 섹션에서 설명합니다. 이 FAQ 섹션에서는 성능에 미치는 잠재적 영향, 기존 저장 프로시저 처리, 복잡한 트랜잭션 관리, 스키마 변경 사항 감독 등 데이터베이스 래퍼 서비스 도입에 대한 일반적인 우려와 질문을 다룹니다.

이 섹션에는 다음 질문이 포함되어 있습니다.

- 래퍼 서비스가 새로운 병목 현상이 되지 않나요?
- 기존 저장 프로시저는 어떻게 되나요?
- 전환 중에 스키마 변경을 관리하려면 어떻게 해야 합니까?

래퍼 서비스가 새로운 병목 현상이 되지 않나요?

데이터베이스 래퍼 서비스는 추가 네트워크 홉을 추가하지만 일반적으로 영향을 최소화합니다. 서비스를 수평적으로 확장할 수 있으며, 제어된 액세스의 이점은 일반적으로 작은 성능 비용을 능가합니다. 성능과 유지 관리성 간의 일시적인 절충이라고 가정합니다.

기존 저장 프로시저는 어떻게 되나요?

처음에는 데이터베이스 래퍼 서비스가 저장 프로시저를 서비스 방법으로 노출할 수 있습니다. 시간이 지남에 따라 로직을 애플리케이션 계층으로 점진적으로 이동하여 테스트 및 버전 관리를 개선할 수 있습니다. 비즈니스 로직을 점진적으로 마이그레이션하여 위험을 최소화합니다.

전환 중에 스키마 변경을 관리하려면 어떻게 해야 합니까?

래퍼 서비스 팀을 통해 스키마 변경 제어를 중앙 집중화합니다. 이 팀은 모든 소비자에 대해 포괄적인 가시성을 유지할 책임이 있습니다. 이 팀은 시스템 전반의 영향에 대해 제안된 변경 사항을 검토하고, 영향을 받는 팀과 조정하고, 제어된 배포 프로세스를 사용하여 수정 사항을 구현합니다. 예를 들어 새 필드를 추가할 때이 팀은 기본값을 구현하거나 처음에 null을 허용하여 이전 버전과의 호환성을 유지해야 합니다.

영향 평가, 테스트 요구 사항 및 롤백 절차를 포함하는 명확한 변경 관리 프로세스를 수립합니다. 데이터베이스 버전 관리 도구를 사용하고 모든 변경 사항을 명확하게 문서화합니다. 이 중앙 집중식 접근 방식은 스키마 수정으로 인해 종속 서비스가 중단되는 것을 방지하고 시스템 안정성을 유지합니다.

응집력 및 결합 분석에 대한 FAQs

데이터베이스 결합 및 응집을 이해하고 효과적으로 분석하는 것은 성공적인 데이터베이스 분해의 기본입니다. 결합 및 응집은이 가이드의 데이터베이스 분해를 위한 응집력 및 결합 분석 섹션에서 설명합니다. 이 FAQ 섹션에서는 적절한 수준의 세부 수준을 식별하고, 올바른 분석 도구를 선택하고, 조사 결과를 문서화하고, 결합 문제의 우선순위를 지정하는 방법에 대한 주요 질문을 다룹니다.

이 섹션에는 다음 질문이 포함되어 있습니다.

- 결합 분석 시 적절한 수준의 세분화를 식별하려면 어떻게 해야 합니까?
- 데이터베이스 결합 및 응집을 분석하는 데 사용할 수 있는 도구는 무엇입니까?
- 결합 및 응집 결과를 문서화하는 가장 좋은 방법은 무엇입니까?
- 먼저 해결해야 할 결합 문제의 우선순위를 지정하려면 어떻게 해야 하나요?
- 여러 작업에 걸쳐 있는 트랜잭션을 처리하려면 어떻게 해야 합니까?

결합 분석 시 적절한 수준의 세분화를 식별하려면 어떻게 해야 합니까?

데이터베이스 관계에 대한 광범위한 분석으로 시작한 다음 체계적으로 드릴다운하여 자연 분리 지점을 식별합니다. 데이터베이스 분석 도구를 사용하여 테이블 수준 관계, 스키마 종속성 및 트랜잭션 경계를 매핑합니다. 예를 들어 SQL 쿼리의 조인 패턴을 검사하여 데이터 액세스 종속성을 이해합니다. 트랜잭션 로그를 분석하여 비즈니스 프로세스 경계를 식별할 수도 있습니다.

결합이 자연스럽게 최소화되는 영역에 집중합니다. 이는 종종 비즈니스 도메인 경계와 일치하며 최적의 분해 지점을 나타냅니다. 적절한 서비스 경계를 결정할 때는 기술적 결합(예: 공유 테이블 및 외래 키)과 비즈니스 결합(예: 프로세스 흐름 및 보고 요구 사항)을 모두 고려하세요.

데이터베이스 결합 및 응집을 분석하는 데 사용할 수 있는 도구는 무엇입니까?

자동화된 도구와 수동 분석을 조합하여 데이터베이스 결합 및 응집력을 평가할 수 있습니다. 다음 도구가이 평가에 도움이 될 수 있습니다.

- 스키마 시각화 도구 - SchemaSpy 또는와 같은 도구를 사용하여 ER 다이어그램pgAdmin을 생성할 수 있습니다. 이 다이어그램은 테이블 관계와 잠재적 결합 지점을 보여줍니다.
- 쿼리 분석 도구 - pg_stat_statements 또는 SQL Server Query Store를 사용하여 자주 조인되는 테이블과 액세스 패턴을 식별할 수 있습니다.

- 데이터베이스 프로파일링 도구 - Oracle SQL Developer 또는와 같은 도구는 쿼리 성능 및 데이터 종속성에 대한 인사이트를 MySQL Workbench 제공합니다.
- 종속성 매핑 도구 - AWS Schema Conversion Tool (AWS SCT)는 스키마 관계를 시각화하고 밀접하게 연결된 구성 요소를 식별하는 데 도움이 될 수 있습니다.는 애플리케이션의 기능 및 도메인 경계를 분석하여 도메인 경계를 식별하는 데 도움이 될 vFunction 수 있습니다.
- 트랜잭션 모니터링 도구 - Oracle Enterprise Manager 또는와 같은 데이터베이스별 도구를 사용하여 트랜잭션 경계SQL Server Extended Events를 분석할 수 있습니다.
- 비즈니스 로직 마이그레이션 도구 - Ispirer Amazon Q Developer 또는와 같은 생성형 AI 도구를 사용하여 JavaKiro로의 변환과 같은 애플리케이션 계층의 데이터베이스 비즈니스 로직을 변환할 수 있습니다.

이러한 자동 분석을 비즈니스 프로세스 및 도메인 지식에 대한 수동 검토와 결합하여 시스템 결합을 완전히 이해합니다. 이 다면적 접근 방식을 사용하면 분해 전략에서 기술적 관점과 비즈니스 관점을 모두 고려할 수 있습니다.

결합 및 응집 결과를 문서화하는 가장 좋은 방법은 무엇입니까?

데이터베이스 관계 및 사용 패턴을 시각화하는 포괄적인 설명서를 생성합니다. 다음은 결과를 기록하는 데 사용할 수 있는 자산 유형입니다.

- 종속성 매트릭스 - 테이블 종속성을 매핑하고 높은 결합 영역을 강조 표시합니다.
- 관계 다이어그램 - ER 다이어그램을 사용하여 스키마 연결 및 외래 키 관계를 표시합니다.
- 테이블 사용량 히트 맵 - 테이블 전체에서 쿼리 빈도 및 데이터 액세스 패턴을 시각화합니다.
- 트랜잭션 흐름도 - 다중 테이블 트랜잭션과 그 경계를 문서화합니다.
- 도메인 경계 맵 - 비즈니스 도메인을 기반으로 잠재적 서비스 경계를 간략하게 설명합니다.

문서에서 이러한 아티팩트를 결합하고 분해가 진행됨에 따라 정기적으로 업데이트합니다. 다이어그램의 경우 draw.io 또는와 같은 도구를 사용할 수 있습니다Lucidchart. 팀 액세스 및 협업을 쉽게 하려면 Wiki를 구현하는 것이 좋습니다. 이 다면적 설명서 접근 방식은 시스템 결합 및 응집에 대한 명확하고 공유된 이해를 제공합니다.

먼저 해결해야 할 결합 문제의 우선순위를 지정하려면 어떻게 해야 하나요?

비즈니스 및 기술 요인에 대한 균형 잡힌 평가를 기반으로 결합 문제의 우선순위를 정합니다. 비즈니스 영향(예: 수익 및 고객 경험), 기술적 위험(예: 시스템 안정성 및 데이터 무결성), 구현 노력 및 팀 역량을

기준으로 각 문제를 평가합니다. 이러한 차원에서 각 문제를 1~5점으로 채점하는 우선순위 매트릭스를 생성합니다. 이 매트릭스는 관리 가능한 위험이 있는 가장 중요한 기회를 식별하는 데 도움이 됩니다.

기존 팀 전문 지식에 맞는 영향력이 높고 위험도가 낮은 변경 사항부터 시작합니다. 이를 통해 보다 복잡한 변화에 대한 조직의 신뢰도와 추진력을 높일 수 있습니다. 이 접근 방식은 현실적인 실행을 촉진하고 비즈니스 가치를 극대화합니다. 변화하는 비즈니스 요구 사항 및 팀 용량에 맞게 우선순위를 정기적으로 검토하고 조정합니다.

여러 작업에 걸쳐 있는 트랜잭션을 처리하려면 어떻게 해야 합니까?

신중하게 설계된 서비스 수준 조정을 통해 다중 작업 트랜잭션을 처리합니다. 복잡한 분산 트랜잭션을 위한 Saga 패턴을 구현합니다. 독립적으로 관리할 수 있는 더 작고 되돌릴 수 있는 단계로 나눕니다. 예를 들어 주문 처리 흐름은 인벤토리 확인, 결제 처리 및 주문 생성을 위한 별도의 단계로 분할될 수 있으며, 각 단계에는 자체 보상 메커니즘이 있습니다.

가능한 경우 작업을 더 원자적으로 재설계하여 분산 트랜잭션의 필요성을 줄입니다. 분산 트랜잭션이 불가피한 경우 강력한 추적 및 보상 메커니즘을 구현하여 데이터 일관성을 높입니다. 트랜잭션 완료율을 모니터링하고 명확한 오류 복구 절차를 구현하여 시스템 신뢰성을 유지합니다.

비즈니스 로직을 애플리케이션 계층으로 마이그레이션하는 방법에 대한 FAQs

데이터베이스에서 애플리케이션 계층으로 비즈니스 로직을 마이그레이션하는 것은 데이터베이스 현대화의 중요하고 복잡한 측면입니다. 이 비즈니스 로직 마이그레이션은이 가이드의 데이터베이스에서 애플리케이션 계층으로 비즈니스 로직 마이그레이션 섹션에서 설명합니다. 이 FAQ 섹션에서는 마이그레이션을 위한 초기 후보 선택부터 복잡한 저장 프로시저 및 트리거 처리에 이르기까지 이러한 전환을 효과적으로 관리하는 방법에 대한 일반적인 질문을 다룹니다.

이 섹션에는 다음 질문이 포함되어 있습니다.

- 먼저 마이그레이션할 저장 프로시저를 식별하려면 어떻게 해야 합니까?
- 로직을 애플리케이션 계층으로 이동하면 어떤 위험이 있습니까?
- 데이터베이스에서 로직을 이동할 때 성능을 유지하려면 어떻게 해야 합니까?
- 여러 테이블이 포함된 복잡한 저장 프로시저를 사용하려면 어떻게 해야 하나요?
- 마이그레이션 중에 데이터베이스 트리거를 처리하려면 어떻게 해야 합니까?
- 마이그레이션된 비즈니스 로직을 테스트하는 가장 좋은 방법은 무엇입니까?

- 데이터베이스와 애플리케이션 로직이 모두 존재하는 경우 전환 기간을 관리하려면 어떻게 해야 합니까?
- 이전에 데이터베이스에서 관리한 애플리케이션 계층의 오류 시나리오를 처리하려면 어떻게 해야 합니까?

먼저 마이그레이션할 저장 프로시저를 식별하려면 어떻게 해야 합니까?

먼저 위험도가 낮고 학습 수준이 높은 가치를 가장 잘 조합한 저장 프로시저를 식별합니다. 최소한의 종속성, 명확한 기능 및 중요하지 않은 비즈니스 영향이 있는 절차에 집중합니다. 이는 팀이 신뢰를 구축하고 패턴을 설정하는 데 도움이 되므로 초기 마이그레이션에 적합한 후보가 됩니다. 예를 들어 복잡한 트랜잭션이나 중요한 비즈니스 로직을 관리하는 것보다 간단한 데이터 작업을 처리하는 절차를 선택합니다.

데이터베이스 모니터링 도구를 사용하여 사용 패턴을 분석하고 자주 액세스하지 않는 절차를 초기 후보로 식별합니다. 이 접근 방식은 비즈니스 위험을 최소화하는 동시에 나중에 더 복잡한 마이그레이션을 해결할 수 있는 귀중한 경험을 제공합니다. 복잡성, 비즈니스 중요도 및 종속성 수준에 대해 각 절차를 평가하여 우선순위가 지정된 마이그레이션 시퀀스를 생성합니다.

로직을 애플리케이션 계층으로 이동하면 어떤 위험이 있습니까?

데이터베이스 로직을 애플리케이션 계층으로 이동하면 몇 가지 주요 문제가 발생합니다. 네트워크 호출 증가로 인해 시스템 성능이 저하될 수 있으며, 특히 데이터베이스 내에서 이전에 처리되었던 데이터 집약적인 작업의 경우 더욱 그렇습니다. 트랜잭션 관리는 더 복잡해지고 분산 운영 전반에 걸쳐 데이터 무결성을 유지하기 위해 신중한 조정이 필요합니다. 특히 이전에 데이터베이스 수준 제약에 의존했던 작업의 경우 데이터 일관성을 보장하기가 어려워집니다.

마이그레이션 중 발생할 수 있는 비즈니스 중단과 개발자를 위한 학습 곡선도 중요한 관심사입니다. 철저한 계획, 스테이징된 환경에서의 광범위한 테스트, 덜 중요한 구성 요소로 시작하는 점진적 마이그레이션을 통해 이러한 위험을 완화합니다. 강력한 모니터링 및 롤백 절차를 구현하여 프로덕션 문제를 신속하게 식별하고 해결합니다.

데이터베이스에서 로직을 이동할 때 성능을 유지하려면 어떻게 해야 합니까?

자주 액세스하는 데이터에 적절한 캐싱 메커니즘을 구현하고, 데이터 액세스 패턴을 최적화하여 네트워크 호출을 최소화하고, 대량 작업에 배치 처리를 사용합니다. non-time-critical 경우 비동기 처리를 고려하여 시스템 응답성을 개선합니다.

애플리케이션 성능 지표를 면밀히 모니터링하고 필요에 따라 조정합니다. 예를 들어 여러 단일 행 작업을 대량 처리로 대체하고, 자주 변경되지 않는 참조 데이터를 캐싱하고, 쿼리 패턴을 최적화하여 데이터 전송을 줄일 수 있습니다. 정기적인 성능 테스트 및 튜닝은 시스템이 허용 가능한 응답 시간을 유지하고 유지 관리 가능성과 확장성을 개선하는 데 도움이 됩니다.

여러 테이블이 포함된 복잡한 저장 프로시저를 사용하려면 어떻게 해야 하나요?

체계적인 분해를 통해 복잡한 다중 테이블 저장 프로시저에 접근합니다. 먼저 이를 더 작고 논리적으로 일관된 구성 요소로 나누고 명확한 트랜잭션 경계와 데이터 종속성을 식별합니다. 각 논리적 구성 요소에 대한 서비스 인터페이스를 생성합니다. 이렇게 하면 기존 기능을 중단하지 않고 점진적으로 마이그레이션할 수 있습니다.

최소 결합 구성 요소부터 step-by-step 마이그레이션을 구현합니다. 매우 복잡한 절차의 경우 더 간단한 부분을 마이그레이션하면서 데이터베이스에 임시로 유지하는 것이 좋습니다. 이 하이브리드 접근 방식은 아키텍처 목표를 향해 나아가는 동안 시스템 안정성을 유지합니다. 마이그레이션 중에 성능과 기능을 지속적으로 모니터링하고 결과에 따라 전략을 조정할 준비를 합니다.

마이그레이션 중에 데이터베이스 트리거를 처리하려면 어떻게 해야 합니까?

시스템 기능을 유지하면서 데이터베이스 트리거를 애플리케이션 수준 이벤트 핸들러로 변환합니다. 동기 트리거를 비동기 작업을 위해 메시지를 대기열에 보내는 이벤트 기반 패턴으로 바꿉니다. 메시지 대기열에 Amazon Simple Notification Service(Amazon SNS) 또는 Amazon Simple Queue Service(Amazon SQS)를 사용하는 것이 좋습니다. 감사 요구 사항의 경우 애플리케이션 수준 로깅을 구현하거나 데이터베이스 변경 데이터 캡처(CDC) 기능을 사용합니다.

각 트리거의 목적과 중요도를 분석합니다. 일부 트리거는 애플리케이션 로직에서 더 잘 제공될 수 있으며, 다른 트리거는 데이터 일관성을 유지하기 위해 이벤트 소싱 패턴이 필요할 수 있습니다. 비즈니스 규칙 또는 데이터 무결성을 관리하는 복잡한 트리거를 처리하기 전에 감사 로그와 같은 간단한 트리거로 시작합니다. 마이그레이션 중에 주의 깊게 모니터링하여 기능 손실이나 데이터 일관성이 없는지 확인합니다.

마이그레이션된 비즈니스 로직을 테스트하는 가장 좋은 방법은 무엇입니까?

마이그레이션된 비즈니스 로직을 배포하기 전에 다중 계층 테스트 접근 방식을 구현합니다. 새 애플리케이션 코드에 대한 단위 테스트로 시작한 다음 end-to-end 비즈니스 흐름을 다루는 통합 테스트를 추가합니다. 이전 구현과 새 구현을 병렬로 실행한 다음 결과를 비교하여 기능적 동등성을 검증합니다. 다양한 로드 조건에서 성능 테스트를 수행하여 시스템 동작이 이전 기능과 일치하거나 초과하는지 확인합니다.

기능 플래그를 사용하여 배포를 제어하면 문제가 발생할 경우 신속하게 롤백할 수 있습니다. 특히 중요한 워크플로의 경우 검증에 비즈니스 사용자를 참여시킵니다. 초기 배포 중에 주요 지표를 모니터링하고 새 구현에 대한 트래픽을 점진적으로 증가시킵니다. 전체적으로 필요한 경우 원래 데이터베이스 로직으로 되돌릴 수 있는 기능을 유지합니다.

데이터베이스와 애플리케이션 로직이 모두 존재하는 경우 전환 기간을 관리하려면 어떻게 해야 합니까?

데이터베이스와 애플리케이션 로직을 모두 사용하는 경우 트래픽 흐름을 제어하고 이전 구현과 새 구현 간의 빠른 전환을 지원하는 기능 플래그를 구현합니다. 엄격한 버전 관리를 유지하고 구현과 해당 책임을 모두 명확하게 문서화합니다. 두 시스템에 대한 포괄적인 모니터링을 설정하여 불일치 또는 성능 문제를 신속하게 식별합니다.

필요한 경우 원래 로직으로 되돌릴 수 있도록 마이그레이션된 각 구성 요소에 대해 명확한 롤백 절차를 설정합니다. 전환 상태, 잠재적 영향 및 에스컬레이션 절차에 대해 모든 이해관계자와 정기적으로 소통합니다. 이 접근 방식은 시스템 안정성과 이해관계자 신뢰도를 유지하면서 점진적으로 마이그레이션하는 데 도움이 됩니다.

이전에 데이터베이스에서 관리한 애플리케이션 계층의 오류 시나리오를 처리하려면 어떻게 해야 합니까?

데이터베이스 수준 오류 처리를 강력한 애플리케이션 계층 메커니즘으로 바꿉니다. 회로 차단기를 구현하고 일시적 장애에 대한 로직을 재시도합니다. 분산 작업 전반에서 데이터 일관성을 유지하기 위해 보정 트랜잭션을 사용합니다. 예를 들어 결제 업데이트가 실패하면 애플리케이션은 정의된 한도 내에서 자동으로 재시도하고 필요한 경우 보상 작업을 시작해야 합니다.

포괄적인 모니터링 및 알림을 설정하여 문제를 신속하게 식별하고 문제 해결을 위한 자세한 감사 로그를 유지 관리합니다. 오류 처리를 최대한 자동화하도록 설계하고 사람의 개입이 필요한 시나리오에 대한 명확한 에스컬레이션 경로를 정의합니다. 이 다계층 접근 방식은 데이터 무결성과 비즈니스 프로세스 연속성을 유지하면서 시스템 복원력을 제공합니다.

에서 데이터베이스 분해를 위한 다음 단계 AWS

데이터베이스 래퍼 서비스를 통해 초기 데이터베이스 분해 전략을 구현하고 비즈니스 로직을 애플리케이션 계층으로 이동한 후 조직은 다음 진화를 계획해야 합니다. 이 섹션에서는 현대화 여정을 계속하기 위한 주요 고려 사항을 간략하게 설명합니다.

이 섹션은 다음 주제를 포함합니다:

- 데이터베이스 분해를 위한 증분 전략
- 분산 데이터베이스 환경에 대한 기술적 고려 사항
- 분산 아키텍처를 지원하기 위한 조직 변경 사항

데이터베이스 분해를 위한 증분 전략

데이터베이스 분해는 세 가지 단계를 통해 점진적으로 진화합니다. 팀은 먼저 모놀리식 데이터베이스를 데이터베이스 래퍼 서비스로 래핑하여 액세스를 제어합니다. 그런 다음 레거시 요구 사항에 맞게 기본 데이터베이스를 유지하면서 데이터를 서비스별 데이터베이스로 분할하기 시작합니다. 마지막으로 완전히 독립적인 서비스 데이터베이스로 전환하기 위해 비즈니스 로직 마이그레이션을 완료합니다.

이 여정 전반에 걸쳐 팀은 신중한 데이터 동기화 패턴을 구현하고 서비스 간 일관성을 지속적으로 검증해야 합니다. 성능 모니터링은 잠재적 문제를 조기에 식별하고 해결하는 데 매우 중요합니다. 서비스가 독립적으로 발전함에 따라 실제 사용 패턴을 기반으로 스키마를 최적화해야 하며 시간이 지남에 따라 누적되는 중복 구조를 제거해야 합니다.

이 증분 접근 방식은 변환 프로세스 전반에 걸쳐 시스템 안정성을 유지하면서 위험을 최소화하는 데 도움이 됩니다.

분산 데이터베이스 환경에 대한 기술적 고려 사항

분산 데이터베이스 환경에서 성능 모니터링은 병목 현상을 조기에 식별하고 해결하는 데 필수적입니다. 팀은 성능 수준을 유지하기 위해 포괄적인 모니터링 시스템 및 캐싱 전략을 구현해야 합니다. 읽기/쓰기 분할은 시스템 전체에서 로드의 균형을 효과적으로 맞출 수 있습니다.

데이터 일관성을 위해서는 분산 서비스 간에 신중한 오케스트레이션이 필요합니다. 팀은 적절한 경우 최종 일관성 패턴을 구현하고 명확한 데이터 소유권 경계를 설정해야 합니다. 강력한 모니터링은 모든 서비스에서 데이터 무결성을 높입니다.

또한 분산 아키텍처를 수용하기 위해 보안을 발전시켜야 합니다. 각 서비스에는 세분화된 보안 제어가 필요하며 액세스 패턴에는 정기적인 검토가 필요합니다. 이 분산 환경에서는 향상된 모니터링 및 감사가 매우 중요합니다.

분산 아키텍처를 지원하기 위한 조직 변경 사항

팀 구조는 명확한 소유권과 책임을 정의하기 위해 서비스 경계와 일치해야 합니다. 조직은 새로운 커뮤니케이션 패턴을 수립하고 팀 내에서 추가 기술 역량을 구축해야 합니다. 이 구조는 기존 서비스의 유지 관리와 지속적인 아키텍처 진화를 모두 지원해야 합니다.

분산 아키텍처를 처리하려면 운영 프로세스를 업데이트해야 합니다. 팀은 배포 절차를 수정하고, 인시던트 대응 프로세스를 조정하고, 변경 관리 관행을 발전시켜 여러 서비스에서 조정해야 합니다.

리소스

다음과 같은 추가 리소스와 도구는 조직이 데이터베이스 분해 여정을 수행하는 데 도움이 될 수 있습니다.

AWS 권장 가이드

- Oracle 데이터베이스를 로 마이그레이션 AWS 클라우드
- Oracle Database의에 대한 리플랫포밍 옵션 AWS
- 클라우드 설계 패턴, 아키텍처 및 구현

AWS 블로그 게시물

- 더 빠른 혁신과 유연성을 위해 데이터베이스에서 애플리케이션으로 비즈니스 로직 마이그레이션

AWS 서비스

- AWS Transform MGN
- AWS Database Migration Service (AWS DMS)
- Migration Evaluator
- AWS Schema Conversion Tool (AWS SCT)
- AWS Transform

기타 도구

- AppEngine(Dynatrace 웹 사이트)
- Oracle Automatic Workload Repository(Oracle 웹 사이트)
- CAST Imaging(CAST 웹 사이트)
- Kiro(Kiro 웹 사이트)
- pgAdmin(pgAdmin 웹 사이트)
- pg_stat_statements(PostgreSQL 웹 사이트)
- SchemaSpy(SchemaSpy 웹 사이트)

- SQL Developer(Oracle 웹 사이트)
- SQLWays(Ispirer 웹 사이트)
- vFunction(vFunction 웹 사이트)

기타 리소스

- 모놀리스에서 마이크로서비스로(O'Reilly 웹 사이트)

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
메인프레임 FAQ 및 AI 도구	이러한 권장 사항이 모놀리식 메인프레임 데이터베이스에 적용되나요?를 추가했습니다. FAQ 및 데이터베이스 분해 중에 사용할 수 있는 AI 도구에 대한 추가 정보가 추가되었습니다.	2025년 10월 14일
최초 게시	—	2025년 9월 30일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.