모듈 17: 트랜잭션

데이터베이스 시스템 개념, 7 판.

개요

§ 거래 개념
§ 트랜잭션 상태
§ 동시 실행
§ 직렬화 가능성 § 복
구 가능성 § 격리 구현
§ SQL의 트랜잭션 정의

§ 직렬화 가능성 테스트.

거래 개념

§ 트랜잭션 은 액세스하고 가능한 프로그램 실행 단위입니다.
다양한 데이터 항목을 업데이트
합니다. § 예: 계좌 A에서 계좌 B로 $50를 이체하는 트랜잭션:

1. 읽기(A)
2. A := A – 50
3. 쓰기(A) 4.
읽기(B)
5. B := B + 50
6. 쓰기(ㄴ)

§ 처리해야 할 두 가지 주요 문제:

- 하드웨어 장애, 시스템 장애 등 각종 장애
충돌
- 여러 트랜잭션의 동시 실행

자금 이체의 예

§ 계좌 A에서 계좌 B로 $50를 이체하는 트랜잭션:

1. 읽기(A)
2. A := A – 50
3. 쓰기(A) 4.
읽기(B)
5. B := B + 50
6. 쓰기(ㄴ)

§ 원자성 요구 사항

- 3단계 이후와 6단계 이전에 트랜잭션이 실패하면 돈이 "손실"되어 일관성 없는 데이터베이스 상태로 이어집니다.
  - § 실패는 소프트웨어 또는 하드웨어로 인한 것일 수 있습니다.
- 시스템은 부분적으로 실행된 트랜잭션의 업데이트가 데이터베이스에 반영되지 않도록 해야 합니다.

§ 내구성 요구 사항 — 사용자에게 트랜잭션이 완료되었음을 알리면(즉, $50의 전송이 발생함) 트랜잭션 에 의한 데이터베이스 업데이트는 소프트웨어 또는 하드웨어 오류가 있는 경우에도 지속되어야 합니다.

자금 이체의 예(계속) Example of Fund Transfer (Cont.)

§ 위의 예에서 일관성 요구 사항 :
- A와 B의 합계는 트랜잭션 실행에 의해 변경되지 않습니다.

§ 일반적으로 일관성 요구 사항에는 다음이 포함됩니다.
- 기본 키 및 외부 키와 같이 명시적으로 지정된 무결성 제약 조건
  열쇠
- 암시적 무결성 제약
  § 예: 모든 계정의 잔액 합계에서 대출 금액 합계를 뺀 값은 현금의 동등한 가치
- 트랜잭션은 일관된 데이터베이스를 확인해야 합니다.
- 트랜잭션 실행 중에 데이터베이스가 일시적으로 일관성이 없을 수 있습니다.
- 트랜잭션이 성공적으로 완료되면 데이터베이스는 일관된
  § 잘못된 트랜잭션 논리는 불일치로 이어질 수 있습니다.

자금 이체의 예(계속) Example of Fund Transfer (Cont.)

§ 격리 요구 사항 — 3단계와 6단계 사이에 다른 트랜잭션 T2가 부분적으로 업데이트된 데이터베이스에 액세스할 수 있는 경우 일관성 없는 데이터베이스를 보게 됩니다(합계 A + B 가 원래보다 작음).

T1	T2
1. 읽기(A)	
2. A := A – 50	
3. 쓰기(A)	
	읽기(A), 읽기(B), 인쇄(A+B)
4. 읽기(B)	
5. B := B + 50	
6. 쓰기(비	

§ 트랜잭션을 직렬로 실행하여 간단하게 격리를 보장할 수 있습니다.
- 즉, 차례로.

§ 그러나 나중에 살펴보겠지만 여러 트랜잭션을 동시에 실행하면 상당한 이점이 있습니다.

ACID 속성

트랜잭션 은 다양한 데이터 항목에 액세스하고 업데이트할 수 있는 프로그램 실행 단위입니다. 데이터의 무결성을 유지하기 위해 데이터베이스 시스템은 다음을 보장해야 합니다.

§ 원자성. 트랜잭션의 모든 작업이
데이터베이스 또는 아무것도 없습니다.

§ 일관성. 독립적으로 트랜잭션을 실행하면 데이터베이스의 일관성이 유지됩니다.

§ 격리. 여러 트랜잭션이 동시에 실행될 수 있지만 각 트랜잭션은 동시에 실행되는 다른 트랜잭션을 인식하지 않아야 합니다.
중간 트랜잭션 결과는 동시에 실행되는 다른 트랜잭션으로부터 숨겨야 합니다.

- 즉, 모든 거래 쌍 Ti 및 Tj 에 대해 Tj가 Ti가 시작되기 전에 실행을 완료했거나 Tj 가 Ti 가 완료된 후에 실행을 시작한 것으로 Ti 에 나타납니다 .

§ 내구성. 트랜잭션이 성공적으로 완료된 후에는 시스템 오류가 있더라도 데이터베이스에 대한 변경 사항이 지속됩니다.

트랜잭션 상태

§ 활성 – 초기 상태. 트랜잭션이 있는 동안 이 상태를 유지합니다.
실행

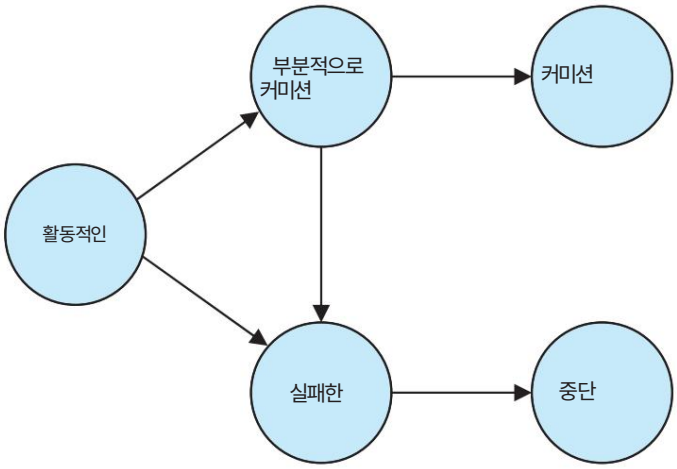
§ 부분 커밋 – 최종 명령문이 실행된 후. § 실패 -- 정상적인 실행을 더 이상 진행할 수 없음을 발견한 후.

§ 중단 – 트랜잭션이 롤백되고 데이터베이스가 종료된 후
트랜잭션 시작 전 상태로 복원됩니다. 중단된 후 두 가지 옵션:

- 거래 재개
  - § 내부 논리적 오류가 없는 경우에만 수행 가능
- 트랜잭션 종료

§ 커밋됨 – 성공적인 완료 후.

트랜잭션 상태(계속)



동시 실행

§ 여러 트랜잭션이 시스템에서 동시에 실행될 수 있습니다.
이점은 다음과 같습니다.

- 프로세서 및 디스크 활용도 증가로 인해
  트랜잭션 처리량 § 예를 들어,
  한 트랜잭션이 CPU를 사용하는 동안 다른 트랜잭션은 CPU를 사용할 수 있습니다.
  디스크에서 읽기 또는 쓰기
- 트랜잭션에 대한 평균 응답 시간 감소 : 짧은 트랜잭션이 긴 트랜잭션 뒤에서 기다릴 필요가 없습니다.

§ 동시성 제어 체계 – 격리를 달성하기 위한 메커니즘

- 즉, 동시 트랜잭션이 데이터베이스의 일관성 을 파괴하지 않도록 하기 위해 동시 트랜잭션 간의 상호 작용을 제어합니다 .

동시 실행.

일정hedules

§ 일정 – 연대순을 지정하는 일련의 지침
동시 트랜잭션의 명령이 실행되는

- 일련의 트랜잭션에 대한 일정은 다음의 모든 지침으로 구성되어야 합니다.
  그 거래
- 지침이 각 문서에 나타나는 순서를 유지해야 합니다.
  개별 거래.

§ 실행을 성공적으로 완료한 트랜잭션은 마지막 명령문으로 커밋 명령을 갖게 됩니다.

- 기본적으로 커밋 명령을 마지막으로 실행하는 것으로 간주되는 트랜잭션
  단계

§ 실행을 성공적으로 완료하지 못한 트랜잭션은
abort 명령을 마지막 명령문으로

일정 1edule 1

§ T1이 A에서 B로 $50를 이체하고 T2가 A에서 B로 잔액의 10%를 이체하도록 합니다 .
A에서 B로

§ T1 다음에 T2 가 오는 직렬 일정 :

T1	T2
읽기 (A) A := A – 50 쓰기 (A) 읽 기 (B) B := B + 50 쓰기 (B) 커 밋	(A) 온도 읽기 := A * 0.1 A := 임사 쓰기 (A) 읽기 (B) B := B + 임시 쓰기 (B) 커 밋

일정 2

§ T2 다음에 T1이 오는 직렬 일정

T1	T2
	(A) 온도 읽기 := A * 0.1
	A := 임사 쓰기
	(A) 읽기 (B)
	B := B + 임시
	쓰기 (B) 커 밋
읽기 (A)	
A := A – 50	
쓰기 (A) 읽 기 (B)	
B := B + 50	
쓰기 (B) 커 밋	

일정 3

§ T1 과 T2를 이전에 정의한 트랜잭션이라고 합니다 . 다음 일정은 연속 일정이 아니지만 일정 1과 동일 합니다.

T1	T2
읽기 (A)	
A := A – 50	
쓰다 (A)	
	(A) 온도 읽기 := A * 0.1
	A := 임사 쓰기 (A)
읽기 (B)	
B := B + 50	
쓰기 (B) 커 밋	
	읽기 (B)
	B := B + 임시
	쓰기 (B) 커 밋

§ 일정 1, 2 및 3에서 합계 A + B가 유지됩니다.

일정 4

§ 다음 동시 스케줄은 (A + B)의 값을 보존하지 않습니다.

T1	T2
읽기 (A) A := A – 50	
	(A) 온도 읽기 := A * 0.1 A := 임시 쓰기 (A) 읽기 (B)
쓰기 (A) 읽기 (B) B := B + 50 쓰기 (B) 커밋	
	B := B + 임시 쓰기 (B) 커밋

직렬화 가능성

§ 기본 가정 - 각 트랜잭션은 데이터베이스 일관성을 유지합니다. § 따라서 일련의 트랜잭션을 직렬로 실행하면 데이터베이스가 보존됩니다.
일관성.

§ A(동시 가능) 일정은 직렬과 동일한 경우 직렬화 가능합니다.
일정. 다양한 형태의 일정 등가성은 다음과 같은 개념을 야기합니다.

1. 충돌 직렬 가능성 2. 보기
직렬 가능성

간소화된 트랜잭션 보기

§ 읽기 및 쓰기 명령 이외의 작업은 무시합니다. § 트랜잭션이 데이터에 대해 임의의 계산을 수행할 수 있다고 가정합니다.
읽기와 쓰기 사이의 로컬 버퍼.

§ 단순화된 일정은 읽기 및 쓰기 지침으로만 구성됩니다.

상충되는 지침

§ 트랜잭션 Ti 및 Tj 의 명령 Ii 및 Ij는 각각 Ii 및 Ij 에 의해 액세스되는 일부 항목 Q가 존재 하고 이러한 명령 중 적어도 하나가 Q를 작성한 경우에만 충돌합니다 .

1. Ii = 읽기(Q), Ij = 읽기(Q). Ii 과 Ij는 충돌하지 않습니다. 2. Ii = 읽기(Q), Ij = 쓰기(Q). 그들은 충돌합니다. 3. Ii = 쓰기(Q), Ij = 읽기(Q). 4. Ii = 쓰기(Q), Ij = 쓰기(Q). 그들은 충돌

§ 직관적으로, Ii 과 Ij 사이의 충돌은 그들 사이에 (논리적) 시간적 순서를 강요합니다.

§ Ii 및 Ij 가 일정에서 연속적이고 충돌하지 않는 경우
결과는 일정에서 교환되더라도 동일하게 유지됩니다.

충돌 직렬화 가능성

§ 일정 S가 충돌하지 않는 명령어의 일련의 교환에 의해 일정 S'로 변환될 수 있는 경우 S와 S'는 충돌 등가 라고 합니다 .

§ 일정 S가 충돌 등가인 경우 충돌 직렬화 가능하다고 말합니다.
직렬 일정으로

충돌 직렬화 가능성(계속)

§ 스케줄 3 은 충돌하지 않는 명령의 일련의 교환에 의해 T2가 T1 다음에 오는 직렬 스케줄인 스케줄 6으로 변환될 수 있습니다 . 따라서 스케줄 3은 충돌 직렬화 가능입니다.

T1	T2
읽기 (A)	
쓰기 (A)	
	읽기 (A)
	쓰기 (A)
읽기 (B)	
쓰기 (B)	
	읽기 (B)
	쓰기 (B)

일정 3

T1	T2
읽기 (A)	
쓰기 (A) 읽	
기 (B) 쓰	
기 (B)	
	읽기 (A)
	쓰기 (A) 읽
	기 (B) 쓰
	기 (B)

스케줄 6

충돌 직렬화 가능성(계속)

§ 충돌 직렬화 가능하지 않은 일정의 예:

T3	T4
읽기 (질문)	
	쓰기 (Q)
쓰기 (Q)	

§ 직렬 일정 < T3, T4 > 또는 직렬 일정 < T4, T3 >을 얻기 위해 위 일정의 지침을 교환할 수 없습니다.

직렬화 가능성 보기

§ S와 S'를 동일한 트랜잭션 집합을 가진 두 개의 일정이라고 합니다. S와 S'는 각 데이터 항목 Q, 1 에 대해 다음 세 가지 조건이 충족되는 경우 보기와 동일 합니다. 스케줄 S에서 트랜잭션 Ti가 Q의 초기 값을 읽으면 스케줄 S'에
서도 트랜잭션 Ti가 초기 값을 읽어야 합니다. Q의

2. 스케줄 S에서 트랜잭션 Ti가 read(Q) 를 실행 하고 그 값이 트랜잭션 Tj (있는 경우) 에 의해 생성된 경우 스케줄 S'에서도 트랜잭션 Ti는 동일한 쓰기(Q) 에 의해 생성된 Q의 값을 읽어야 합니다.) 트랜잭션 Tj 의 작업 .

3. 최종 쓰기(Q) 작업을 수행하는 트랜잭션(있는 경우)
스케줄 S는 또한 스케줄 S'에서 최종 쓰기(Q) 작업을 수행해야 합니다.

§ 볼 수 있듯이 보기 동등성은 순전히 읽기 및 쓰기 에 기반합니다.
홀로.

직렬 가능성 보기(계속)

§ 일정 S는 직렬과 동일한 보기인 경우 보기 직렬화 가능입니다.
일정.

§ 모든 충돌 직렬화 가능 일정도 보기 직렬화 가능입니다.

§ 다음은 뷰 직렬화 가능하지만 충돌 직렬화 가능하지 않은 일정입니다.

T27	T28	T29
읽기 (질문)		
	쓰기 (Q)	
쓰기 (Q)		
		쓰기 (Q)

§ 위에 해당하는 직렬 일정은 무엇입니까? § 충돌 직렬화 가능하지
않은 모든 뷰 직렬화 가능 스케줄은 블라인드
씁니다.

직렬화 가능성의 다른 개념

§ 아래 일정은 연속 일정 < T1, T5 >와 동일한 결과를 생성하지만 이에 상응하는 충돌 또는 보기는 아
닙니다.

T1	T5
읽기 (A)	
A := A – 50	
쓰다 (A)	
	읽기 (B)
	B := B 10
	쓰기 (B)
읽기 (B)	
B := B + 50	
쓰기 (B)	
	읽기 (A)
	A := A + 10
	쓰다 (A)

§ 그러한 동등성을 결정하려면 읽기 및 쓰기 이외의 작업 분석이 필요합니다.

직렬화 가능성 테스트

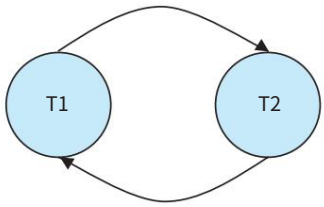
§ 트랜잭션 집합 T1, T2, ..., Tn 의 일정 을 고려하십시오 .

트랜잭션(이름).

§ 두 트랜잭션이 충돌하는 경우 Ti 에서 Tj 까지 호를 그리고 Ti는 먼저 충돌이 발생한 데이터 항목에 액세스했습니다.

§ 액세스한 항목별로 호에 레이블을 지정할 수 있습니다. § 우선 순위

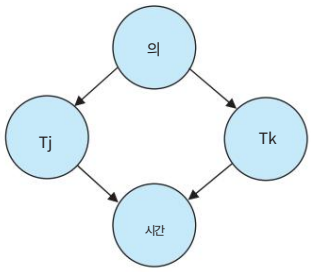
그래프의 예



충돌 직렬화 가능성 테스트

§ 일정은 우선 순위 그래프가 비순환인 경우에만 충돌 직렬화 가능합니다. § n2 시간 차수를 갖는 주기 감지

알고리즘이 존재합니다 . 여기서 n은 그래프의 정점 수입니다.



(ㅏ)

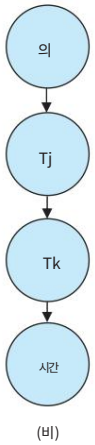
- (더 나은 알고리즘은 순서 n + e를 취합니다. 여기서 e는 에지의 수입니다.)

§ 우선 순위 그래프가 비주기적이면
그래프의 토폴로지 정렬을 통해 직렬화 순서를 얻을 수 있습니다 .

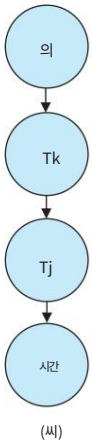
- 이것은 그래프의 부분 순서와 일치하는 선형 순서입니다.
- 예를 들어, 다음에 대한 직렬화 가능성 순서 스케줄 A는

  T5 ® T1 ® T3 ® T2 ® T4

  § 다른 사람들이 있습니까?



(비)



(씨)

뷰 직렬화 가능성 테스트

§ 충돌 직렬화 가능성에 대한 우선 순위 그래프 테스트를 사용할 수 없음
보기 직렬화 가능성을 테스트하기 위해 직접.

- 뷰 직렬화 가능성을 테스트하기 위한 확장은 우선 순위 그래프의 크기에 따라 비용이 기하급수적으로 증가합니다.

§ 일정이 보기 직렬화 가능한지 확인하는 문제는 NP-완전 문제 클래스에 속합니다.

- 따라서 효율적인 알고리즘이 존재할 가능성은 거의 없습니다.

§ 그러나 일부 충분 조건 만 확인하는 실용적인 알고리즘
뷰 직렬화를 위해 여전히 사용할 수 있습니다.

복구 가능한 일정

동시에 실행 중인 트랜잭션에 대한 트랜잭션 실패의 영향을 해결해야 합니다. § 복구 가능한 일정 — 트랜잭

션 Tj가 트랜잭션 Ti가 이전에 작성한 데이터 항목을 읽으면 Ti 의 커밋 작업이 Tj 의 커밋 작업보다 먼저 나타납니다 . § 아래 스케줄(스케줄 11)은 복구 불가

Q8	T9
읽기 (A)	
쓰기 (A)	
	읽기 (A)
	커밋
읽기 (B)	

§ T8이 중단되어야 하는 경우 T9는 일관성 없는 데이터베이스 상태를 읽었을 것입니다(사용자에게 표시되었을 수도 있음). 따라서 데이터베이스는 일정을 복구할 수 있는지 확인해야 합니다.

계단식 롤백

§ 계단식 롤백 – 단일 트랜잭션 실패로 인해 일련의
트랜잭션 롤백. 트랜잭션이 아직 커밋되지 않은 다음 일정을 고려하십시오(따라서 일정을 복구할 수 있음).

T10	T11	T12
읽기 (A) 읽기 (B) 쓰기 (A)		
	읽기 (A) 쓰기 (A)	
		읽기 (A)
중단하다		

T10이 실패 하면 T11 및 T12 도 롤백해야 합니다.

§ 상당한 양의 작업을 취소할 수 있음

캐스케이드 없는 일정

§ 계단식 일정 - 계단식 롤백이 발생할 수 없습니다.

- Tj가 데이터 항목을 읽는 각 거래 쌍 Ti 및 Tj 에 대해
이전에 Ti 에 의해 작성된 경우 Ti 의 커밋 작업은 Tj 의 읽기 작업 전에 나타납니다 . § 캐스케이드 없는 모든 스

케줄도 복구 가능 § 캐스케이드 없는 스케줄로 제한하는 것이 바람직함

동시성 제어

§ 데이터베이스는 가능한 모든 일정을 보장하는 메커니즘을 제공해야 합니다.

- 충돌 또는 뷰 직렬화 가능, 그리고
- 복구 가능하며 캐스케이드가 없는 것이 바람직합니다.

§ 한 번에 하나의 트랜잭션만 실행할 수 있는 정책은 직렬 일정을 생성하지만 동시성 수준이 낮습니다.

- 직렬 스케줄이 복구 가능/캐스케이드리스입니까?

§ 일정이 실행된 후 직렬화 가능성을 테스트하는 것은 너무 늦습니다! § 목표 - 직렬성을 보장하는 동시성 제어 프로토콜을 개발합니다.

동시성 제어(계속)

§ 일정은 데이터베이스 일관성을 위해 충돌하거나 보기 직렬화 가능하고 복구 가능해야 하며 바람직하게는 계단식으로 연결되지 않아야 합니다. § 한 번에 하나의 트랜잭션만 실행할 수 있는 정책은 일련의 일정을 생성하지만 낮은 수준의 동시성을 제공합니다. § 동시성 제어 체계는 허용되는 동시성 양과 발생하는 오버헤드 양 사이의 절충안입니다. § 일부 체계는 충돌 직렬화 가능 일정만 생성되도록 허용하는 반면 다른 체계는 충돌 직렬화 가능하지 않은 보기 직렬화 가능 일정을 허용합니다.

동시성 제어와 직렬화 가능성 테스트

§ 동시성 제어 프로토콜은 동시 일정을 허용하지만 일정이 충돌/보기 직렬화 가능하고 복구 가능하며 캐스케이드가 없는지 확인합니다.

§ 동시성 제어 프로토콜은 (일반적으로) 우선 순위를 검사하지 않습니다.
생성되는 그래프

- 대신 프로토콜은 비직렬화를 방지하는 규율을 부과합니다.
일정.
- 16장에서 그러한 프로토콜을 연구합니다.

§ 서로 다른 동시성 제어 프로토콜은 서로 다른 장단점을 제공합니다.
허용되는 동시성의 양과 발생하는 오버헤드의 양.

§ 직렬화 가능성에 대한 테스트는 동시성 제어 프로토콜이 올바른 이유를 이해하는 데 도움이 됩니다.

약한 수준의 일관성

§ 일부 응용 프로그램은 약한 수준의 일관성을 유지하여 직렬화할 수 없는 일정을 허용합니다.

- 예: 대략적인 합계를 얻으려는 읽기 전용 트랜잭션
모든 계정의 잔액
- 예, 쿼리 최적화를 위해 계산된 데이터베이스 통계는 대략적일 수 있습니다(이유는?).
- 이러한 트랜잭션은 다른 트랜잭션과 관련하여 직렬화할 필요가 없습니다.
업무

§ 성능에 대한 절충 정확도

SQL-92의 일관성 수준

§ 직렬화 가능 — 기본값

§ 반복 가능한 읽기 — 커밋된 레코드만 읽을 수 있습니다.

- 동일한 레코드의 반복 읽기는 동일한 값을 반환해야 합니다. • 그러나

트랜잭션을 직렬화할 수 없을 수 있습니다. 트랜잭션에 의해 삽입된 일부 레코드는 찾을 수 있지만 다른 레코드는 찾지 못할 수 있습니다.

§ 커밋된 읽기 — 커밋된 레코드만 읽을 수 있습니다.

- 연속적인 레코드 읽기는 다르게 반환될 수 있습니다(하지만 커밋됨).
  가치.

§ 커밋되지 않은 읽기 — 커밋되지 않은 레코드도 읽을 수 있습니다.

일관성 수준

§ 근사치 수집에 유용한 낮은 수준의 일관성
데이터베이스에 대한 정보

§ 경고: 일부 데이터베이스 시스템은 직렬화 가능 일정을 보장하지 않습니다.
기본

§ 예: Oracle(및 버전 9 이전의 PostgreSQL)은 기본적으로 스냅샷 격리라는 일관성 수준을 지원합니다(SQL 표준의 일부가 아님).

SQL의 트랜잭션 정의

§ SQL에서 트랜잭션은 암시적으로 시작됩니다. § SQL

의 트랜잭션은 다음과 같이 종료됩니다.

- 커밋 작업은 현재 트랜잭션을 커밋하고 새 트랜잭션을 시작합니다.
- 롤백 작업 으로 인해 현재 트랜잭션이 중단됩니다.

§ 거의 모든 데이터베이스 시스템에서 기본적으로 모든 SQL 문은 성공적으로 실행되면 암시적으로 커밋 됩니다.

- 암시적 커밋은 데이터베이스 지시문에 의해 해제될 수 있습니다.

§ 예: JDBC에서 -- connection.setAutoCommit(false);

§ 격리 수준은 데이터베이스 수준에서 설정할 수 있습니다.

§ 트랜잭션 시작 시 격리 수준을 변경할 수 있습니다.

§ 예: SQL에서 트랜잭션 격리 수준 직렬화 설정 § 예: JDBC에서 --

connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE)

격리 수준 구현

§ 잠금

- 전체 데이터베이스 잠금 vs 항목 잠금
- 잠금을 유지하는 데 얼마나 걸립니까?
- 공유 잠금과 배타 잠금

§ 타임스탬프

- 할당된 트랜잭션 타임스탬프(예: 트랜잭션이 시작될 때) • 데이터 항목은 두 개의 타임스탬프 저

장 § 읽기 타임스탬프 § 쓰기 타임스탬프 • 타임

스탬프는 잘못된 액세스를

감지하는 데 사용됨 § 각

데이터 항목의 여러 버전

- 트랜잭션이 데이터베이스의 "스냅샷"에서 읽을 수 있도록 허용

SQL 문으로 트랜잭션

§ 예: 트랜잭션 1: 급여 >
90000 인 강사 로부터 ID, 이름 선택 § 예: 트랜잭션 2: 강사 값 에 삽입
('11111', 'James',
'Marketing', 100000) § 가정

- T1 시작, 인덱스를 사용하여 급여 > 90000인 튜플을 찾아 잠급니다.
- 그런 다음 T2가 실행됩니다.
- T1과 T2가 충돌합니까? 튜플 수준 잠금이 충돌을 감지합니까? • 유령 현상 의 인스턴스 §

Wu의 급여 = 90000인 아래의 T3도 고려하십시오.

강사 세트 급여 업데
이트 = 급여 * 1.1 여기서
이름 = 'Wu'

§ 핵심 아이디어: "술어" 충돌 감지 및 "술어" 형식 사용
잠금"

17장 끝