10. MongoDB Lock과 트랜잭션

Lock 매커니즘과 트랜잭션

. 데이터의 무결성을 위해 기본적으로 고립화 기능을 제공 → 기존의 RDBMS는 Read Committed 메커니즘

. 한 쪽에서 TX Low Lock 잡게 되면 동일 행의 변경 시 Lock (Wait)상태로 빠지며 조회 시에는 변경 전 데이터를 조회하게 됨

. MongoDB는 빠른 쓰기와 읽기 성능을 보장하기 위한 데이터 저장 기술이기 때문에 DML 수행 시 Auto-commit을 동반하게 되고 다른 사용자는 항상 변경 후 데이터만 참조 → Read uncommitted

. 대부분의 NoSQL 제품들이 Read Uncommitted 타입의 고립화 정책을 기반

MongoDB의 Lock

MongoDB의 트랜잭션

. 트랜잭션 매커니즘

- 트랜잭션이 커밋되면 트랜잭션의 모든 데이터 변경 사항이 저장되고 트랜잭션 외부에서 볼 수 있습니다. 즉, 트랜잭션은 일부 변경 사항을 커밋하지 않고 다른 일부는 롤백합니다.

  트랜잭션이 커밋 될 때까지 트랜잭션의 데이터 변경 사항은 트랜잭션 외부에서 볼 수 없습니다.

그러나 트랜잭션이 여러 샤드에 쓸 때 모든 외부 읽기 작업이 커밋 된 트랜잭션의 결과가 샤드에 표시 될 때까지 기다릴 필요는 없습니다. 예를 들어 트랜잭션이 커밋되고 쓰기 1이 샤드 A에 표시되지만 쓰기 2가 아직 샤드 B에 표시되지 않는 경우 읽기 문제의 외부 읽기 문제 `"local"` 는 쓰기 2를 보지 않고 쓰기 1의 결과를 읽을 수 있습니다.

- 트랜잭션이 중단되면 트랜잭션에서 변경된 모든 데이터가 표시되지 않고 삭제됩니다. 예를 들어 트랜잭션의 작업이 실패하면 트랜잭션이 중단되고 트랜잭션의 모든 데이터 변경 사항이 표시되지 않고 폐기
- 다중 문서 트랜잭션은 단일 문서 쓰기에 비해 성능 비용이 더 높으며 다중 문서 트랜잭션의 가용성이 효과적인 스키마 설계를 대체해서는 안됩니다. 많은 시나리오에서 비정규 화 된 데이터 모델 (내장 된 문서 및 배열) 은 계속해서 데이터 및 사용 사례에 최적입니다. 즉, 많은 시나리오에서 데이터를 적절하게 모델링하면 다중 문서 트랜잭션의 필요성이 최소화 됨
- **기존** 컬렉션 에 대해 읽기 / 쓰기 (CRUD) 작업을 지정할 수 있습니다 . CRUD 작업의 목록을 보려면 CRUD 조작 .
- 사용하는 경우 기능 호환성 버전 (FCV) `"4.4"` 이상을, 당신은 거래 컬렉션과 인덱스를 생성 할 수 있습니다. 자세한 내용 은 트랜잭션에서 컬렉션 및 인덱스 만들기를 참조 하십시오.
- 트랜잭션에 사용되는 컬렉션은 다른 데이터베이스에있을 수 있습니다.
- capped collection 에는 쓸 수 없습니다 . (MongoDB 4.2부터)

. 4버전 이후 강화된 트랜잭션

Multi-Document Transaction

- MongoDB 4.0부터 제공되었지만 ReplicaSets 환경에서만 지원 되었으며 4.2부터 Shard-Cluster 환경에서도 지원
- Multi-Document Transaction은 여러 작업 Collection, DB에 적용할 수 있으며 트랜젝션이 Commit되면 변경된 모든 데이터를 저장하고 Rollback되면 모든 데이터의 변경을 취소
- Embedded Document 및 Array 구조와 같은 단일 Document Transaction에 비해 성능지연 문제가 발생할 가능성이 있기 때문에 이를 대체하면 안됨
- featureCompatibilityVersion 4.0 이후 환경에서 사용할 수 있으며 wiredTiger 저장엔진과 In-Memory 저장엔진에서만 사용할 수 있음(admin DB에서 설정 가능)
- config, admin, local DB의 Collection을 읽기/쓰기 할 수 없으며 system.* 컬렉션은 쓰기 작업을 수행 할 수 없다. 또한 다중 트랜잭션이 진행 중인 Collection은 인덱스

를 추가 및 삭제 할 수 없음

- 트랜잭션 총크기가 16MB이상인 경우에도 필요한 만큼의 Oplog를 생성하여 처리할 수 있다.
- Transaction을 진행하기 전에 안전하게 write concern을 majority를 설정 권고
- Wired Tiger인 Primary Server와 In-memory인 Secondary Server 환경에서도 트랜잭션을 지원

Snapshot & Committed Data

. 트랜잭션을 제어하기 위해 4.0버전 부터 변경된 전 데이터를 임시로 저장해 둘 스냅샷 (SnapShot or Rollback) 데이터를 위한 별도의 저장 구조를 제공 → Rollback File

. 롤백 파일은 ReplicaSets 시스템에서 다중 도큐멘트 트랜잭션에 의해 발생된 변경 전 데이터를 트랜잭선이 종료될 때까지 임시로 보관하는 저장 파일

. 기존에 Primary Server에 대한 장애 조치 후 ReplicaSets 서버에 다시 참여 할 때 기존 쓰기 작업은 모두 롤백

. 롤백은 Secondary Server가 성공적으로 복제하지 못한 경우에만 제한적으로 허용하며 롤백 파일은 각멤버들 간에 일관성을 유지하기 위해 사용

. Primary Server에 장애가 발생하기 전에 Secondary Server에 복제 작업이 정상적으로 완료되고 대다수의 멤버를 사용할 수 있는 상태인 경우 롤백 작업이 수행되지 않음

. 롤백 파일이 생성되는 경로와 기본값

. 롤백 파일은 mongod.exe 구동 시 설정한 /dbpath 경로 밑에 rollback이름으로 생성

. 롤백 파일의 기본 파일명 포맷은 다음 예와 같음 → <db-name>.<collection>.<timestamp>.bson