1. 핵심 모듈 및 알고리즘 설명

1.1. DBEnv 및 DB 객체 구성

Berkeley DB의 환경(DBEnv)은 트랜잭션, 로그, 락, 캐시 초기화를 포함하여 전체 DB 운영의 기반이 됨. 각각의 테이블은 DB 객체로 관리되며 실제 데이터를 저장. 메타데이터는 __meta__ DB를 통해 관리 및 저장

1.2. 테이블 ID 관리

입력으로 들어오는 테이블 이름은 항상 테이블 id로 변환되어 관리됨. 실제 DB이름도 테이블 id로 입력됨. 이를 통해 rename table의 동작이 단순해지며 보안성과 추상화 측면에서 유리함. 하지만, 운영자가 직접 파일만 보고 테이블을 식별하기 어렵다는 단점도 존재.

1.3. 메타데이터 및 제약조건 저장 구조

모든 스키마 및 제약조건 정보는 __meta__ DB 내부에 저장되며,
테이블 스키마(__schema__:{table_id}),
제약조건(__constraints__:{table_id}),
참조 정보(__references__:{table_id}) 등이 관리된다. PRIMARY KEY, FOREIGN KEY, NOT NULL 등의 제약을 수동으로 파싱하고 검증하여 저장한다.
이를 통해 실제 데이터가 들어있는 db를 열지 않고도 많은 작업들을 수행할 수 있어 성능 면에서 장점을 지닌다.

1.4. 트랜잭션

모든 작업은 명시적으로 시작한 트랜잭션 안에서 처리, 성공 시 commit(), 실패 시 abort() 이를 통해 원자성, 일관성, 복구 가능성이 확보됨.

2. 구현 내용 요약

SQL기반의 구문들을 처리할 수 있는 데이터베이스를 구현함.

2.1. 테이블 관련 기능: CREATE TABLE, DROP TABLE, RENAME TABLE, TRUNCATE TABLE

2.2. 데이터 조작 기능: INSERT INTO, SELECT

2.3. 메타데이터 출력: EXPLAIN, SHOW TABLES

2.4. 제약조건 지원: PRIMARY KEY, FOREIGN KEY, NOT NULL

2.5. 트랜잭션 처리: 모든 작업이 트랜잭션 내부에서 수행되며 오류 발생 시 자동 abort

3. 느낀 점

이번 프로젝트를 통해 데이터 저장 구조와 메타데이터 관리의 필요성을 깊이 이해하게

되었고, 내부적으로 추상화된 테이블 ID 구조가 보안성은 높지만 운영 및 디버깅에서는 불편할 수 있음을 체감했다. 추후에는 내부 ID와 사용자 친화적 이름 사이의 매핑을 시각화하거나, 디버깅 툴을 개발하는 방향도 고려할 수 있다.

4. 기타 사항

4.1. 과제파일에 명시되어 있지 않은 query들은 처리하지 않음. ex) update, delete

4.2. select나 show tables등 table이름과 column 이름, 값의 길이는 20자이하라고 생각하고 구현하였다. 길이가 20자를 넘어가는 경우 20자에서 그냥 잘린다.

4.3. 추가 정의 오류 유형

4.3.1. table_id는 존재하나 그 id의 schema가 존재하지 않는 경우.
Fatal Error: Table ID exist, but schema does not exist.

4.3.2. rename시 이미 존재하는 이름으로 rename하려는 경우
Rename table has failed: table with the same name already exists

4.3.3. select시 *가 아닌 다른 것을 읽어오려는 경우
Select has failed: '*' is only supported