**데이터 베이스 기본**

**데이터베이스 개요**

**데이터베이스의 정의**

• 데이터베이스는 체계적으로 구조화된 데이터의 집합입니다.

• 데이터의 저장, 검색, 수정, 삭제 등의 작업을 효율적으로 수행하기 위해 사용됩니다.

• 데이터베이스는 다양한 응용 프로그램이 데이터에 접근하고 관리할 수 있게 합니다.

• 데이터베이스는 일반적으로 데이터베이스 관리 시스템(DBMS)에 의해 제어됩니다. 데이터베이스와 DBMS를 하나로 묶어 데이터베이스 시스템이라고 하며 단축하여 데이터베이스라고도 합니다.

데이터베이스의 특징

• **데이터 독립성 (Data Independence):** 데이터 구조 변경 시 응용 프로그램에 영향을 미치지 않도록 논리적, 물리적 독립성을 유지합니다.

• **동시성 제어 (Concurrency Control):** 여러 사용자가 동시에 데이터에 접근할 수 있게 하여 일관된 상태를 유지합니다.

**• 데이터 보안성 (Data Security):** 접근 권한 제어와 암호화를 통해 데이터를 보호하고 안전하게 관리합니다.

• **복구성 (Recoverability):** 시스템 장애 시 백업과 복구 메커니즘을 통해 데이터를 일관된 상태로 복구합니다.

• **데이터 공유성 (Data Sharing):** 여러 사용자와 응용 프로그램이 데이터를 일관성 있게 공유할 수 있게 합니다.

• **성능 최적화 (Performance Optimization):** 쿼리 최적화, 인덱스 사용 등으로 데이터 접근 속도를 향상시킵니다.

**데이터베이스와 파일 시스템의 차이점**

**• 데이터 구조화**

• **데이터베이스:** 데이터가 테이블, 뷰, 인덱스 등 구조화된 형태로 저장됩니다.

**• 파일 시스템:** 데이터가 파일과 폴더로 저장되며, 구조화되지 않은 경우가 많습니다.

**• 데이터 무결성**

**• 데이터베이스:** 무결성 제약 조건을 통해 데이터의 일관성을 유지합니다.

• **파일 시스템:** 별도의 무결성 제약 조건이 없어 데이터 일관성 유지가 어렵습니다.

**• 동시성 제어**

**• 데이터베이스:** 여러 사용자가 동시에 데이터에 접근할 수 있도록 동시성 제어를 지원합니다.

**• 파일 시스템:** 동시 접근 시 데이터 손상 가능성이 높습니다.

**• 데이터 검색**

**• 데이터베이스:** SQL과 같은 고급 쿼리 언어를 사용하여 데이터를 효율적으로 검색합니다.

• **파일 시스템:** 파일 탐색 기능에 의존하며, 대량 데이터 검색이 비효율적입니다.

**데이터베이스 모델 종류**

• 관계형 데이터베이스

• 데이터를 테이블 형식으로 저장하고, SQL을 사용해 데이터를 관리합니다.

• 예: MySQL, PostgreSQL, Oracle

• 분산 데이터베이스

• 데이터를 여러 물리적 위치에 분산하여 저장하고 관리합니다.

• 예: Google Spanner, Apache Cassandra

• NoSQL 데이터베이스

• 비정형 데이터를 효율적으로 저장하고 처리하기 위해 설계된 데이터베이스입니다.

• 예: MongoDB, CouchDB

• 그래프 데이터베이스

• 데이터를 노드, 엣지, 속성의 형태로 저장하여 복잡한 관계를 쉽게 표현하고 쿼리합니다.

• 예: Neo4j, Amazon Neptune

• 데이터 웨어하우스, OLTP 데이터베이스, 객체 지향 데이터베이스 등

**관계형 데이터베이스 (RDBMS)**

**관계형 데이터베이스 개요**

데이터를 테이블 형식으로 저장하고, SQL을 사용해 데이터를 관리합니다.

또한, 테이블 간의 관계를 이용해 데이터를 구조화할 수 있습니다.

• **테이블**

• 데이터를 행과 열로 구성된 2차원 구조로 표현합니다.

• 각 테이블은 고유한 이름을 가집니다.

**• 행 (레코드)**

• 테이블의 가로줄을 가리킵니다.

• 하나의 개체에 대한 정보를 포함합니다.

**• 열 (필드)**

• 테이블의 세로줄을 가리킵니다.

• 특정 속성에 대한 정보를 표현합니다.

**• 트랜잭션 (Transaction)**

• 데이터베이스에서 수행하는 작업의 논리적 단위를 말합니다.

• 트랜잭션은 데이터베이스에서 데이터를 안전하게 관리하기 위한 기본적인 개념으로, 여러 단계의 작업을 한 덩어리로 묶어서 한 번에 처리합니다.

**ACID**

**ACID**는 데이터베이스 트랜잭션의 신뢰성과 일관성을 보장하기 위한 네 가지 주요 특성을 의미합니다. 이 특성들은 트랜잭션이 데이터베이스에 올바르게 적용되고 데이터의 무결성이 유지되도록 합니다.

1. **Atomicity (원자성)**

• 트랜잭션의 모든 작업이 모두 성공하거나 모두 실패하는 것을 보장합니다.

• 부분적으로 완료된 트랜잭션은 시스템에 반영되지 않습니다.

• 예: 은행 계좌 이체 시, 돈이 출금되고 입금되는 모든 과정이 성공해야 합니다. 하나라도 실패하면 전체 트랜잭션은 취소됩니다.

1. **Consistency (일관성)**

• 트랜잭션이 완료되면 데이터베이스는 일관된 상태를 유지합니다.

• 트랜잭션이 실행되기 전과 후의 데이터 상태는 데이터베이스의 모든 제약 조건을 만족해야 합니다.

• 예: 데이터베이스의 무결성 제약 조건(예: 외래 키, 고유 키 제약 조건 등)이 항상 만족되어야 합니다.

1. **Isolation (고립성)**

• 동시에 실행되는 트랜잭션이 서로에게 영향을 미치지 않도록 합니다.

• 하나의 트랜잭션이 완료될 때까지 다른 트랜잭션이 그 중간 상태를 볼 수 없습니다.

• 예: 여러 사용자가 동시에 같은 데이터를 수정할 때, 트랜잭션 간 간섭을 방지하여 데이터 일관성을 유지합니다.

1. **Durability (지속성)**

• 트랜잭션이 성공적으로 완료되면, 그 결과는 영구적으로 저장됩니다.

• 시스템 장애가 발생하더라도 트랜잭션 결과는 손실되지 않습니다.

• 예: 트랜잭션 완료 후, 데이터는 로그 파일이나 영구 저장소에 기록되어 데이터베이스가 재시작되어도 손실되지 않습니다.

**관계형 데이터베이스의 특징**

**• 데이터 무결성 (Data Integrity)**

**• 참조 무결성:** 외래 키를 통해 테이블 간의 관계를 유지하고, 데이터의 일관성을 보장.

• **고유성**: 고유 키와 유니크 제약 조건을 통해 중복되지 않는 데이터를 보장.

• **제약 조건:** NOT NULL, CHECK 등 제약 조건을 통해 데이터의 유효성을 보장.

• **정규화 (Normalization)**

• 데이터 중복을 최소화하고 데이터 구조를 효율적으로 설계하기 위해 정규화 과정을 적용.

• 제1정규형(1NF)부터 제3정규형(3NF)까지의 정규화 단계로 데이터의 중복을 줄이고 무결성을 향상.

**• 강력한 쿼리 언어 (SQL)**

• 데이터를 정의, 조회, 수정, 삭제하기 위해 표준 SQL(Structured Query Language)을 사용.

• 복잡한 쿼리와 조인 연산을 통해 다양한 데이터 검색 및 조작 가능.

• **트랜잭션 관리 (Transaction Management)**

• BEGIN, COMMIT, ROLLBACK 명령어를 통해 트랜잭션을 관리.

• 트랜잭션 단위로 데이터 변경을 처리하여 데이터 무결성과 일관성을 유지.

**• 인덱스 (Index)**

• 데이터 검색 속도를 향상시키기 위해 인덱스를 사용.

• B-Tree, Hash 등 다양한 인덱스 구조를 지원하여 효율적인 데이터 접근 가능.

**• 확장성 (Scalability)**

• 수직적 확장(서버 성능 향상)을 통해 데이터베이스 성능을 향상 가능.

• 대규모 데이터를 처리하기 위해 클러스터링 및 분산 데이터베이스 기술을 사용할 수 있음.

**관계형 데이터베이스를 수평적 확장을 하는 방법**

**샤딩 (Sharding)**

• 데이터베이스를 여러 개의 분할된 부분(샤드)으로 나누어 분산 저장합니다.

• 각 샤드는 고유한 데이터 하위 집합을 저장하고 독립적으로 쿼리를 처리합니다.

• 샤딩 키를 선택하여 데이터를 균등하게 분배합니다.

**리플리케이션 (Replication)**

• 데이터를 여러 노드에 복제하여 읽기 성능을 향상할 수 있습니다.

• 마스터-슬레이브 복제를 통해 쓰기 작업은 마스터 노드에서, 읽기 작업은 슬레이브 노드에서 처리합니다.

• 다중 마스터 복제를 통해 쓰기 작업도 분산할 수 있습니다

**관계형 데이터베이스의 수평적 확장이 어려운 이유**

**• 데이터 일관성 유지의 어려움**

• 분산된 여러 노드에서 데이터의 일관성을 유지하는 것이 복잡하고 어렵습니다.

• 트랜잭션이 여러 노드에 걸쳐 있는 경우, ACID(Atomicity, Consistency, Isolation, Durability) 속성을 보장하기 어려습니다.

**• 복잡한 조인 연산**

• 관계형 데이터베이스는 조인을 통해 여러 테이블에서 데이터를 결합하는 경우가 많습니다.

• 데이터가 분산되면 조인 연산이 매우 복잡해지고, 성능이 저하될 수 있습니다. **트랜잭션 처리**

• RDBMS는 트랜잭션 처리 시 일관성을 보장하기 위해 강력한 잠금 메커니즘을 사용합니다.

• 분산 환경에서 이러한 잠금 메커니즘이 병목 현상을 초래할 수 있습니다.

**-> 따라서 데이터베이스의 수평적 확장이 필요한 경우 분산 SQL 데이터베이스 를 고려하는 것이 좋습니다. (**예: Google Spanner, Apache Cassandra)

**비관계형 데이터베이스 (NoSQL)**

**비관계형 데이터베이스 개요**

고정된 스키마를 따르지 않고 다양한 데이터 모델을 지원하는 데이터베이스 유형입니다. Not Only SQL 의 약자로, SQL 외의 다양한 데이터 접근 방식을 포함합니다.

**다양한 데이터 모델**

NoSQL 데이터베이스는 다양한 데이터 모델을 지원하여 다양한 데이터 형태를 효율적으로 저장하고 처리할 수 있습니다.

• **문서형 데이터베이스 (Document Store)**

• JSON 또는 BSON 형식의 문서(Document) 데이터를 저장합니다.

• 각 도큐먼트는 고유한 키로 식별되며, 내부에는 다양한 필드와 값을 포함할 수 있습니다.

• 문서의 집합을 컬렉션(Collection) 이라고 합니다.

**키-값 스토어 (Key-Value Store)**

• 키와 값의 쌍으로 데이터를 저장합니다.

• 빠른 읽기와 쓰기 성능을 제공하며, 메모리 기반의 데이터베이스로서 캐싱에 매우 유리합니다.

• 예: Redis, DynamoDB

• **와이드 칼럼 스토어 (Wide Column Store)**

• 테이블을 키와 여러 컬럼으로 구성하여 데이터를 저장합니다.

• 각 행은 여러 개의 컬럼 패밀리로 구성되며, 데이터 모델이 유연하여 다양한 쿼리에 적합합니다.

• 예: Apache Cassandra, HBase

• **그래프 데이터베이스 (Graph Database)**

• 노드(Node)와 엣지(Edge)로 데이터를 저장하며, 복잡한 관계를 효율적으로 표현합니다. • 네트워크와 관계 구조를 사용하여 데이터를 저장하고 쿼리할 수 있습니다.

• 예: Neo4j, JanusGraph

**비관계형 데이터베이스의 특징**

**• 유연한 스키마 (Flexible Schema)**

• 데이터 구조가 고정되어 있지 않아서, 데이터 모델이 필요에 따라 동적으로 변경될 수 있습니다.

• 이는 애플리케이션의 요구사항이나 데이터 형식의 변화에 대응하는 데 유리합니다.

• **수평적 확장성 (Horizontal Scalability)**

• 데이터베이스가 여러 서버에 데이터를 분산하여 저장하고 처리할 수 있습니다.

• 이로 인해 데이터베이스가 더 많은 트래픽과 데이터 증가를 처리할 수 있습니다.

• **고성능**

• 대부분의 NoSQL 데이터베이스는 높은 처리량(throughput)을 제공하며, 특히 분산된 환경에서의 읽기 및 쓰기 성능이 뛰어납니다.

• 이는 데이터베이스가 대규모 데이터 집합을 효율적으로 처리할 수 있음을 의미합니다.

• **다양한 데이터 모델.**

• 다양한 데이터 모델을 지원하여, 문서형, 키-값 쌍, 컬럼 패밀리, 그래프 데이터 등 여러 형태의 데이터를 저장하고 쿼리할 수 있습니다.

• 각 데이터 모델은 특정한 사용 사례와 데이터 패턴에 맞게 최적화되어 있습니다.

**• 비구조적 데이터 처리**

• 비구조적인 데이터를 다루는 데 유리하며, 대용량 데이터의 저장과 분석에 적합합니다.

• 특히 웹 애플리케이션과 소셜 미디어 플랫폼 등에서 많이 사용됩니다.

**• 분산 데이터베이스**

• 데이터의 복제와 샤딩(Sharding)을 통해 데이터베이스의 가용성을 높이고 장애 복구를 지원합니다.

• 이는 데이터베이스의 신뢰성과 내결함성을 향상시킵니다.

**정규화 (Normalization)**

정규화는 데이터베이스 설계에서 중복을 최소화하고 데이터의 일관성과 무결성을 유지하기 위해 데이터를 구조화하는 과정입니다. 주로 관계형 데이터베이스에서 사용됩니다.

**주요 목적**

**• 중복 제거:** 데이터의 중복을 최소화하여 저장 공간을 절약하고, 데이터 일관성을 유지합니다.

• **데이터 의존성 제거:** 데이터가 올바른 형식으로 저장되도록 하여 삽입, 삭제, 수정 시 발생할 수 있는 이상 현상을 방지합니다.

• **검색 최적화:** 데이터베이스에서 쿼리 성능을 최적화하여 데이터를 효율적으로 검색할 수 있도록 합니다.

• 단, 조인 비용이 증가할 수 있고, 데이터가 여러 테이블에 분산되어 있을 경우 인덱스 효율성이 감소하여 데이터 읽기 성능이 저하될 수 있습니다.

**정규화의 주요 단계**

**• 제 1 정규화 (1NF):** 모든 속성 값이 원자적(Atomic)이어야 합니다. 즉, 하나의 속성이 하나의 값을 가져야 합니다.

• **제 2 정규화 (2NF):** 모든 비주요 속성이 주요 키 전체에 대해 완전 함수적 종속성을 가져야 합니다. 즉, 부분적 종속성을 제거합니다.

**• 제 3 정규화 (3NF):** 모든 비주요 속성이 후보키에 이행적으로 종속되지 않아야 합니다. 즉, 이행적 종속성을 제거합니다.

• **BCNF (Boyce-Codd 정규형):** 모든 결정자가 후보키여야 합니다. 즉, 함수적 종속성을 유지하고 분해되지 않도록 합니다.

• **제 4 정규화 (4NF):** 다치 종속성을 제거하여 다중 값 종속성을 분리합니다.

• **제 5 정규화 (5NF):** 조인 종속성을 제거하여 관련 속성을 분리합니다.텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명