Char3. 데이터 입출력 구현

데이터 모델

현실 세계의 정보들을 컴퓨터에 표현하기 위해서 단순화, 추상화하여 체계적으로 표현한 개념적 모형

데이터베이스 설계과정에서 데이터 구조를 논리적으로 표현하기 위해 사용되는 지능적 도구

- 구성요소

1. **객체 Entity**

   데이터 베이스에 표현하려는 것으로 **사람이 생각하는 개념이나 정보 단위 같은 현실 세계의 대상체**

2. **속성 Attribute**

   데이터의 가장 작은 논리적 단위로서 파일 구조상의 데이터 항목 또는 데이터 필드에 해당

3. **관계 Relationship**

   개체간의 관계 또는 속성 간의 논리적인 연결을 의미

- 종류

1. **개념적 데이터 모델**

   현실 세계에 대한 인간의 이해를 돕기 위해 현실 세계에 대한 인식을 추상적 개념으로 표현한 과정

   Ex) E-R모델

2. **논리적 데이터 모델**

   개념적 모델링 과정에서 얻은 개념적 구조는 컴퓨터가 이해하고 처리할 수 있는 컴퓨터 세계의 환경에 맞도록 변환하는 과정

   EX) 관계모델, 계층모델, 네트워크모델

   논리적 데이터 모델 품질 검증

   완성된 논리 데이터 모델이 기업에 적합한지를 확인하기 위해 품질을 검증하는 것

3. **물리적 데이터 모델**

- 데이터 모델에 표시할 요소

1. **구조 Structure**

   논리적으로 표현된 개체 타입들 간의 관계로서 **데이터 구조 및 정적 성질을 표현**한다.

2. **연산 Operation**

   데이터베이스에 저장된 실제 데이터를 처리하는 작업에 대한 명세로서 **데이터베이스를 조작하는 기본도구**

3. **제약조건 Constraint**

   데이터 베이스에 저장될 수 있는 **실제 데이터의 논리적인 제약조건**

이상 Anomaly

테이블에서 일반 속성들의 종속으로 인해 데이터 중복이 발생하고 이 중복Redundancy으로 인해 테이블 조작 시 문제가 발생하는 현상을 의미

- **삽입 이상 Insertion Anomaly**

  테이블에 데이터를 삽입할 때 의도와는 상관없이 원하지 않는 값들로 인해 삽입할 수 없게 되는 현상

- **삭제 이상 Deletion Anomaly**

  테이블에서 한 튜플을 삭제시 의도와는 상관없는 값들도 함께 삭제되는, 즉 연쇄 삭제가 발생하는 현상

- **갱신 이상 Update Anomaly**

  테이블에서 튜플에 있는 속성 값을 갱신할 때 일부 튜플의 정보만 갱신되어 정보에 불일치성 Inconsistency이 생기는 현상

함수적 종속 Functional Dependency

데이터의 의미를 표현하는 것으로 현실 세계를 표현하는 제약조건이 되는 동시에 데이터 베이스에서 항상 유지되어야 할 조건

- 완전 함수적 종속 Full Functional Dependency
- 부분 함수적 종속 Partial Functional Dependency

X→Y: X(결정자 Determinant)가 Y(종속자 Dependent)를 함수적으로 결정한다. Or **Y는 X에 함수적 종속**

정규화 Normalization

- **속성간의 종속성 분해하여 하나의 종속성은 하나의 릴레이션에 표현하도록 분해하는 과정**
- 테이블 속성들이 상호 종속적인 관계를 갖는 특성을 이용해 테이블을 무손실 분해하는 과정
- 목적: 가능한 한 중복을 제거하여 삽입, 삭제, 갱신 이상의 발생 가능성을 줄이는 것

1NF(제 1 정규형)

릴레이션에 속한 모든 값들이 원자값으로만 구성

2NF(제 2 정규형)

기본키가 아닌 모든 속성이 기본키에 대하여 완전 함수적 종속을 만족

완전 함수적 종속 : 기본키에 의해서 속성이 결정

부분 함수적 종속 : 기본키의 일부에 의해 속성이 결정

아래와 같이 학번과 과목코드가 기본키인 릴레이션이 있을 때

- ◆ 과목 점수는 기본키(학번, 과목코드)를 가지고 알수 있음 = 완전 함수적 종속
- ◆ 이름은 기본키의 일부(학번)를 가지고 알수 있음 = 부분 함수적 종속

학번	이름	과목코드	과목점수
2015xxxx	박xx	A01	95
2017xxxx	김xx	A02	80

3NF(제 3 정규형)

기본키가 아닌 모든 속성이 기본키에 대해 이행적 종속을 만족하지 않음

이행적 종속 : A → B, B → C 일 때 A → C 를 만족하는 관계

BCNF(Boyce-Codd 정규형)

결정자가 모두 후보키

4NF(제 4 정규형)

릴레이션에 **다치종속**이 성립하는 경우 모든 속성이 함수적 종속 관계를 만족

다치종속 Multi Valued Dependency

3 개의 속성 A,B,C 를 가진 테이블 R 에서 어떤 복합속성(A,C)에 대응하는 B 값의 집합이 A 값에만 종속되고 C 값에 무관하다면 B 는 A 에 다치종속이라 하고 A→→B 로 표기한다.

5NF(제 5 정규형)

모든 **조인 종속**이 후보키를 통해서만 성립

테이블 Table

데이터를 저장하는 데이터베이스의 가장 기본적인 오브젝트

로우 Row	튜플, 인스턴스, 어커런스라고도 한다.
컬럼 Column	각 속성 항목에 대한 값을 저장한다.
기본키 Primary key	기본키는 후보키 중에서 선택한 주키이다. 한 릴레이션에서 특정 튜플을 유일하게 구별할 수 있는 속성이다.
외래키 Foreign key	다른 릴레이션의 기본키를 참조하는 속성 또는 속성들의 집합이다. 한 릴레이션에 속한 속성A와 참조릴레이션의 기본키인 B가 동일한 도메인 상에서 정의 되었을 때의 속성A를 외래키라고 한다.

엔티티Entity를 테이블Table로 변환

엔티티를 테이블로 변환한 후 **테이블 목록 정의서**를 작성

테이블 목록 정의서: 전체 테이블을 목록으로 요약 관리하는 문서로 테이블 목록이라고도 한다.

논리적 설계(데이터 모델링)	물리적 설계
엔티티 Entity	**테이블 Table**
속성 Attribite	**컬럼 Column**
주 식별자 Primary Identifier	**기본키 Primary key**
외부 식별자 Foreign Identifier	**외래키 Forreign key**
관계 Relatioinship	**관계 Relationship**

슈퍼타입 기준 테이블 변환

서브타입을 슈퍼타입에 통합하여 **하나의 테이블**로 만드는 것

서브타입에 속성이나 관계가 적을 경우 적용하는 방법으로 하나의 통합된 테이블에는 서브타입의 모든 속성이 포함되어야 한다.

- 장점
  1. 데이터의 액세스가 상대적으로 용이하다.
  2. 뷰를 이용하여 각각의 서브타입만을 액세스하거나 수정할 수 있다.
  3. 서브타입 구분이 없는 임의 집합에 대한 처리가 용이하다.
  4. 여러 테이블을 조인하지 않아도 되므로 수행 속도가 빨라진다.
  5. SQL 문장 구성이 단순해진다.
- 단점
  1. 테이블의 컬럼이 증가하므로 디스크 저장 공간이 증가한다.
  2. 처리마다 서브타입에 대한 구분Type이 필요한 경우가 많이 발생한다.
  3. 인덱스 크기의 증가로 인덱스의 효율이 떨어진다.

서브타입 기준 테이블 변환

슈퍼타입 속성들이 각각의 서브타입에 추가하여 **서브타입들을 개별적인 테이블**로 만드는 것

서브타입에 속성이나 관계가 많이 포함된 경우 적용한다.

- 장점
  1. 각 서브타입 속성들의 선택 사양이 명확한 경우에 유리하다.
  2. 처리할 때마다 서브타입 유형을 구분할 필요가 없다.
  3. 여러 개의 테이블로 통합하므로 테이블당 크기가 감소하여 전체 테이블 스캔시 유리하다.
- 단점
  1. 수행속도가 감소할 수 있다.
  2. 복잡한 처리를 하는 SQL의 통합이 어렵다.
  3. 부분 범위에 대한 처리가 곤란해진다.
  4. 여러 테이블을 통합한 뷰는 조회만 가능하다.
  5. UID(Unique Identifier 식별자)의 유지관리가 어렵다.

개별타입 기준 테이블 변환

슈퍼타입과 서브타입들을 각각의 개별적인 테이블로 변환하는 것 (슈퍼타입과 서브타입 테이블 사이에 **1:1 관계** 형성)

속성을 컬럼으로 변환

- 논리 데이터 모델에서의 **Primary UID**는 물리 데이터 모델의 **기본키**로 만든다.
- **다른 엔티티와의 관계로 인해 생성된 Primary UID**는 물리데이터모델의 **기본키**로 만든다.
- 논리 모델링에서 정의된 **Secondary UID 및 Alternate Key**는 물리 모델에서 **유니크키**로 만든다.

반정규화 Denormalization

시스템의 성능향상, 개발 및 운영의 편의성 등을 위해 정규화된 데이터 모델을 통합, 중복, 분리하는 과정

의도적으로 정규화 원칙을 위배

반정규화 방법		내용	고려사항
테이블 통합		통합할 경우 1. 두 개의 테이블에서 발생하는 프로세스가 동일하게 자주 처리되는 경우 2. 두 개의 테이블을 이용하여 항상 조회하는 경우 종류 **1. 1:1 관계 테이블 통합** **2. 1:N 관계 테이블 통합** 3. **슈퍼타입/서브타입 테이블 통합**	1. 데이터 검색은 간편하지만 레코드 증가로 인해 처리량이 증가한다. 2. 테이블 통합으로 인해 입력, 수정, 삭제 규칙이 복잡해질 수 있다. 3. Not Null, Default, Check등의 제약조건을 설계하기 어렵다.
테이블 분할	수평분할	**레코드**를 기준으로 테이블을 분할한다. 레코드 별로 사용빈도의 차이가 큰 경우 사용빈도에 따라 테이블이 분할된다.	1. 기본키의 유일성 관리가 어렵다. 2. 데이터 양이 적거나 사용빈도가 낮은 경우 테이블 분할이 필요한지를 고려해야 한다. 3. 분할된 테이블로 인해 수행 속도가 느려질 수 있다. 4. 데이터 검색에 중점을 두어 테이블 분할 여부를 결정해야 한다.
	수직분할	하나의 테이블에 **속성**이 너무 많은 경우 **속성**을 기준으로 테이블을 분할한다. 1. 갱신 위주의 속성 분할 2. 자주 조회되는 속성 분할 3. 크기가 큰 속성 분할 4. 보안을 적용해야 하는 속성 분할	
중복 테이블 추가		중복 테이블 추가해야 하는 경우 **1. 정규화로 인해 수행 속도가 느려지는 경우** **2. 많은 범위 데이터를 자주 처리해야 하는 경우** **3. 특정 범위 데이터만 자주 처리해야 하는 경우** **4. 처리 범위를 줄이지 않고는 수행 속도를 개선할 수 없는 경우** 중복 테이블 추가하는 방법 1. 집계 테이블 추가 2. 진행 테이블 추가 3. 특정 부분만을 포함하는 테이블 추가	
중복 속성 추가		중복속성 추가하면 데이터 무결성 확보가 어렵고 디스크 공간이 추가로 필요하다. 중복 속성을 추가해야 하는 경우 **1. 조인이 자주 발생하는 속성인 경우** **2. 접근 경로가 복잡한 속성인 경우** **3. 액세스의 조건으로 자주 사용되는 속성인 경우** 4. **기본키의 형태가 적절하지 않거나 여러 개의 속성으로 구성된 경우**	1. 테이블 중복과 속성의 중복을 고려해야 한다. 2. 데이터 일관성 및 무결성에 유의해야 한다. 3. SQL 그룹 함수를 이용하여 처리할 수 있어야 한다. 4. 저장 공간의 지나친 낭비를 고려한다.

인덱스 Index

데이터 레코드를 빠르게 접근하기 위해 **<키, 값, 포인터>쌍**으로 구성되는 데이터 구조

클러스터드 인덱스 Clustered index

레코드의 물리적 순서가 인덱스의 엔트리 순서와 일치하게 유지되도록 구성되는 인덱스

트리 기반 인덱스	**인덱스를 저장하는 블록들이 트리 구조를 이루고 있는 것으로 상용 DBMS에서는 트리 구조 기반의 B+트리 인덱스를 주로 활용한다.**
비트맵 인덱스	**인덱스 컬럼의 데이터를 Bit값인 0 또는 1로 변환하여 인덱스 키로 사용하는 방법**
함수 기반 인덱스	**컬럼의 값 대신 컬럼에 특정 함수나 수식을 적용하여 산출된 값을 사용하는 것 B+트리 인덱스 또는 비트맵 인덱스를 생성하여 사용**
비트맵 조인 인덱스	**다수의 조인된 객체로 구성된 인덱스로 단일 객체로 구성된 일반적인 인덱스와 액세스 방법이 다르다. 비트맵 인덱스와 물리적 구조가 동일**
도메인 인덱스	**개발자가 필요한 인덱스를 직접 만들어 사용하는 것. 확장형 인덱스 Extensible Index**

인덱스 설계 순서

1. 인덱스의 대상 테이블이나 컬럼 등을 선정한다.
2. 인덱스의 효율성을 검토하여 인덱스 최적화를 수행한다
3. 인덱스 정의서를 작성한다.

인덱스 설계시 고려사항

1. 새로 추가되는 인덱스는 기존 액세스 경로에 영향을 미칠 수 있다.
2. 인덱스를 지나치게 많이 만들면 오버헤드가 발생한다.
3. 인덱스를 만들면 추가적인 저장공간이 필요하다.
4. 인덱스와 테이블 데이터의 저장공간이 분리되도록 설계한다.

뷰 View

사용자에게 접근이 허용된 자료만을 제한적으로 보여주기 위해 하나 이상의 기본 테이블로부터 유도된 이름을 가지는 가상테이블

장점

1. 논리적 데이터 독립성을 제공
2. 동일 데이터에 대해 동시에 여러 사용자의 상이한 응용이나 요구를 지원해 준다.
3. 사용자의 데이터관리를 간단하게 해준다.
4. 접근제어를 통한 자동보안이 제공된다.

단점

1. 독립적인 인덱스를 가질 수 없다.
2. 뷰의 정의를 변경할 수 없다.
3. 뷰로 구성된 내영에 대한 삽입, 삭제, 갱신 연산에 제약이 따른다.

고려사항

- 테이블 구조가 단순화 될 수 있도록 **반복적 조인**을 설정하여 사용하거나 동일한 조건절을 사용하는 테이블을 뷰로 생성
- 동일한 테이블이라도 업무에 따라 **테이블을 이용하는 부분이 달라질** 수 있으므로 사용할 데이터를 다양한 관점에서 제시
- **데이터의 보안 유지**를 고려하여 설계한다.

클러스터 Cluster

데이터 저장시 데이터 액세스 효율을 향상시키기 위해 **동일한 성격의 데이터를 동일한 데이터 블록에 저장하는 물리적 저장 방법**

클러스터링키로 지정된 컬럼 값의 순서대로 저장되고, 여러 개의 테이블이 하나의 클러스터에 저장된다.

특징

- 클러스터링 된 테이블은 데이터 **조회** 속도는 향상시키지만 데이터 **입력, 수정, 삭제**에 대한 성능은 저하시킨다.
- 클러스터는 **데이터의 분포도가 넓을수록** 유리하다.
- **데이터 분포도가 넓은 테이블**을 클러스터링하면 저장 공간 절약 가능
- 클러스터링된 테이블은 **클러스터링키 열을 공유**하므로 저장공간이 줄어든다.
- 대용량을 처리하는 **트랜잭션**은 전체 테이블을 스캔하는 일이 자주 발생하므로 클러스터링을 하지 않는 것이 좋다.
- **처리범위가 넓은 경우에는 단일 테이블 클러스터링을, 조인이 많이 발생하는 경우에는 다중 테이블 클러스터링을 사용**
- **파티셔닝된 테이블**에는 클러스터링을 할 수 없다.
- 클러스터링을 하면 **비슷한 데이터가 동일한 데이터 블록에 저장**되기 때문에 디스크 I/O가 줄어든다.
- 클러스터링된 테이블에 **클러스터드 인덱스**를 생성하면 접근성능이 향상된다.

클러스터 대상 테이블

- **분포도가 넓은 테이블**
- **대량의 범위를 자주 조회하는 테이블**
- **입력, 수정, 삭제가 자주 발생하지 않는 테이블**
- **자주 조인되어 사용되는 테이블**
- **ORDER BY, GROUP BY, UNION이 빈번한 테이블**

파티션 Partition

데이터베이스에서 파티션은 대용량의 테이블이나 인덱스를

작은 논리적 단위인 파티션으로 나누는 것을 의미

장점

- **데이터 접근시 액세스 범위를 줄여 쿼리 성능이 향상된다.**
- **파티션 별로 데이터가 분산되어 저장되므로 디스크의 성능이 향상된다.**
- **파티션별로 백업 및 복구를 수행하므로 속도가 빠르다.**
- **시스템 장애 시 데이터 손상 정도를 최소화할 수 있다.**
- **데이터 가용성이 향상된다.**
- **파티션 단위로 입출력을 분산시킬 수 있다.**

단점

- **하나의 테이블을 세분화하여 관리하므로 세심한 관리가 요구된다.**
- **테이블간 조인에 대한 비용이 증가한다.**
- **용량 작은 테이블에 파티셔닝 수행하면 오히려 성능저하발생**

• 파티셔닝 방식에 따른 파티션의 종류

범위 분할 (Range Partitioning)	지정한 열의 값을 기준으로 분할
해시 분할 (Hash Partitioning)	해시 함수를 적용한 결과 값에 따라 데이터를 분할
조합 분할 (Composite Partitioning)	범위 분할로 분할한 다음 해시 함수를 적용하여 다시 분할

• 인덱스 파티션 : 파티션된 테이블의 데이터를 관리하기 위해 인덱스를 나눈 것

트랜잭션 Transaction

• 데이터베이스의 상태를 변환시키는 하나의 논리적 기능을 수행하기 위한 작업의 단위 또는 한꺼번에 모두 수행되어야 할 일련의 연산들을 의미한다.

• 특징

Atomicity (원자성)	트랜잭션의 연산은 데이터베이스에 모두 반영되도록 완료(Commit)되든지 아니면 전혀 반영되지 않도록 복구(Rollback)되어야 함
Consistency (일관성)	트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 변환함
Isolation (독립성)	둘 이상의 트랜잭션이 동시에 병행 실행되는 경우 어느 하나의 트랜잭션 실행중에 다른 트랜잭션의 연산이 끼어들 수 없음
Durability (지속성)	성공적으로 완료된 트랜잭션의 결과는 시스템이 고장나더라도 영구적으로 반영되어야 함