2. DB와 사용자

▶ 용어

▷ 데이터

의미를 가지면서 기록될 수 있는 알려진 사실

기록될 수 없는 것은 데이터가 아님

▷ 데이터베이스

관련있는 데이터의 모임

-> 의미를 가지면서 기록될 수 있는 알려진 사실들의 모임

-> 여러 개가 있어야 함, 관련이 있어야 함

▷ 데이터베이스 관리시스템

데이터베이스의 생성과 관리를 담당하는 SW 패키지

▷ 데이터베이스 시스템

데이터베이스와 그것을 관리리하는 SW를 모두 칭하는 용어

▷ 작은 세계

데이터베이스 구축의 대상이 되는 실세계의 일부분

▶ 데이터마이닝

연관성이 있는 데이터를 찾아내기

(ex. 기저귀와 맥주가 같이 판매가 되었다)

▶ Stored Database가 데이터베이스

메타 데이터: 데이터가 무엇인지(어떤 식으로 정의되어 있는지 등등) 설명하는 데이터

-> 메타데이터가 있으면 데이터를 더 빠르게 엑세스할 수 있음

▶ 파일의 특징

① 물리적인 비트의 연속(비트스트림)

② 순차적인 레코드들로 구성

③ 레코드는 연관의 필드에 모임

▶ 파일 시스템

파일을 활용하여 자료를 관리하는 방법

데이터에 대한 프로그램의 의존도가 높음

-> 프로그램은 데이터에 의존함

각 행(↓) 레코드, 각 열(→) 필드

★ 데이터베이스 문제점

데이터 중복성

각종 기능 부족(데이터 모델링 개념, 질의어, 동시성 제어, 파손/회복, 보안

데이터의 불일치: 어떤 데이터가 맞는 데이터인지 알 수 없게 됨

▶ 데이터베이스 관리 시스템(DBMS)

데이터의 구조가 바뀌어도 프로그램을 바꾸려고 프로그래밍을 다시 하지 않아도 됨

▶ DB 특징(DB 철학)
-> 사용자와 관계 없이 DB 설계 가능
▷ DB 시스템 전기기술성
DB애 대한 데이터(메타 데이터)를 통하여 데이터 구조를 직접 알고 있지 않아도 데이터 액세스 가능
(ex. 롤딕에서 공격력을 액세스 하면 방어력도 액세스 가능)
▷ 프로그램과 데이터의 분리
-> 분리 안하면 데이터를 바꿀때마다 업뎃 해야함
DB 내 데이터 저장 구조 변경되도 DB 응용 프로그램은 영향을 받지 않음(변경 필요 X)
▷ 데이터 추상화
데이터 모델을 사용함으로써 저장구조와는 별도로 데이터의 의미를 표현하는 방법 제공
(ex. 이름: 주소, 학과 + 이름: 전번 이런식으로 두개로 저장될 수도 있음
-> 사용자는 어떻게 저장되어 있는지 알빠노 -> 실제 데이터의 저장 구조가 자유를 얻음)
▷ 데이터에 대한 다양한 뷰
사용자는 전체 DB보다 관심이 있는 DB 일부를 뷰로 저장 가능
데이터가 통합된 경우 보이고 싶은 데이터만 보일 수 있음 -> 이것이 "뷰"
(ex. 교수가 액세스 가능한 파일과 학생이 액세스 가능한 파일 나눌 필요 X)

▶ DB 스키마
전체적인 DB 구조(틀)
자주 변경 X
(ex. 학생에 대한 정보를 모을 때 이름, 학번, 번호를 저장하자 -> 이것이 스키마(틀))

▶ DB 상태
특정 시점의 DB 내용을 의미
시간이 지남에 따라 자주 변경
스키마 위에 저장된 실제 데이터

▶ DBMS 사용 효과
① 표준화 된 데이터
-> 기존의 여러 데이터를 하나의 데이터로 표준화
-> 즉, 업무 효율성 증대
② 데이터 구조 변경에 융통성 부여
-> 구조가 변경되도 사용자는 영향 X
③ 응용 프로그램 개발 시간 단축
-> 응용 프로그램의 상당한 부분 DBMS가 처리
④ 항상 최신 정보 제공
-> 한 명이 업뎃 하면 나머지도 즉시 변경 값 업뎃
⑤ 규모의 경제성
-> 부서 별로 데이터 관리보다 통합 DB에서 관리한 것이 전체적으로 비용 감소

▶ DBMS 단점
비용 증가
성능 감소(기능이 너무 많음)
고장의 영향 확대(북한이 이거 부수면 우리나라 망함)

3. 인덱스(파일 - 블록 - 레코드 - 필드)

▶ 레코드 특징

① 필드의 개수는 동일

② 각 필드의 크기는 고정

③ 모든 레코드의 크기는 동일

▶ 블록 안에서는 순차겁색을 해야 하나 이진검색을 해야 하나?

블록 안에서 검색을 하는 것은 하드디스크 액세스가 아니고 메모리 액세스임

(위에 여행 경비 예시와 동일)자질구리한 것은 크게 신경쓰지 않음

▶ 데이터 파일에 새로운 레코드를 삽입하는 방법은?

① 삽입할 자리 찾기

② 찾은 블록 (A)에 공간이 있으면 삽입하고 종료

③ A에 공간이 없으면

-1. 새로운 블록 (B)을 할당하여 A뒤에 연결

-2. A에 저장되어 있던 레코드와 삽입할 레코드를 A와 B에 절반씩 나누어 저장하고 종료

문제점: 블럭의 분할이 일어날 수 있음

▶ 순서 파일의 삭제(재분배)

블록 사용률이 50% 이하가 되면

다음 블록 B를 읽어 블록 사용률을 계산

a+b <= 100%이면 합치고

a+b > 100%이면 B에서 A로 일부 레코드를 이동하여 A, B 모두 50% 이상이 되도록 함

▶ 인덱스

레코드를 '빠르게' 찾을 수 있도록 도와주는 보조 '파일'

파일에 대한 또 다른 접근 경로

▶ 일반적인 인덱스 구조

해당 인덱스의 키값을 가지고 있는 블럭의 첫번째 주소를 가르킴

▶ 인덱스가 있는 경우의 검색 알고리즘

인덱스를 순차검색 후, 데이터 파일을 순차검색

★ 기본 인덱스

인덱스 파일이 데이터 파일의 첫 번째 엔트리를 가르킴

인덱스 파일이 작고

데이터 파일의 순차 검색이 필요 없음(디스크 액세스 ↓)

★ 밀집 인덱스

데이터 화일 내 모든 탐색 키 값(모든 레코드)에 대한 인덱스 엔트리를 정의

★ 희소 인덱스

탐색 값의 일부에 대해서만 인덱스 엔트리를 정의

★ 파일 크기 계산(가정: 각 블록은 레코드로 차있음)
파일 크기 = 블록 개수 * 블록 크기
블록 개수 = 레코드 개수 / bf(하나의 블록에 드갈 수 있는 레코드 최대 개수)
bf = 블록 크기 / 레코드 크기 (소수점은 버림)

▶ 보조 인덱스(무명 인덱스)
정렬되지 않은 키값을 찾을 때 보조해주는 인덱스
있으면 좋은 편

▶ 중복되는 키값이 있는 데이터 파일을 가르키는 키들을 모아둔 데이터 파일을 가르키는 인덱스 파일의 경우
중복이 되면 끝까지 순차탐색을 해야 하기 때문에 그래도 있는게 좋음

▶ 무얼 만들지?
중복 X 정렬 O: 기본 인덱스
중복 O 정렬 O: 클러스터링 인덱스
중복 X 정렬 X: 보조 인덱스
중복 O 정렬 X: 중복되는 키값이 있는 데이터 파일을 가르키는 키들을 모아둔 인덱스를 가르키는 보조 인덱스
-> 이 인덱스들의 특징은 중복 X 정렬 O
-> 그럼 이 인덱스들의 엔트리를 찾는 방법은? -> 인덱스의 인덱스(기본 인덱스)를 생성

▶ 다단계 인덱스
인덱스 파일에 대한 인덱스 파일을 여러 단계로 나눈 인덱스

▶ B 트리
이진트리를 확장해 하나의 노드가 가질 수 있는 자식 노드의 최대 숫자가 2보다 큰 트리 구조

▶ B+ 트리
모든 key, data가 리프노드에 모여 있음
모든 리프노드가 연결리스트 형태를 띄고 있음
리프노드의 부모 키는 리프노드의 첫 번째 키보다 작거나 같음

▶ B 트리 잘 그리는 법
① 각 노드에는 50%까지만 허용하는데, 루트 노드의 경우에는 50% 이하일 수 있음, 나머지는 무조건 50% 이상이여야 함
② 각 노드의 키값이 k개가 있으면 포인터는 k+1개가 반드시 있어야 함
③ 상위 노드에서 내려오는 포인터가 두 개면 안됨
④ 레벨을 두 개 이상 건너뛸 수 없음

4. ER 모델

▶ ER 모델 요소

엔티티: 독립적으로 존재하면서 고유하게 식별 가능한 객체, 뚜렷하고 명확(ex. 학생, 학과, 교과목)
애트리뷰트: 엔티티를 구성하는 속성(ex. 이름, 학번, 위치, 강사)
관계: 두 개 이상의 엔티티들을 연관짓는 것(ex. 수강, 소속)

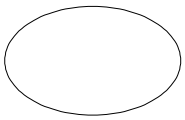
▶ 엔티티

엔티티: 자신의 애트리뷰트에 대한 특정 값을 가짐, 특정 레코드와 유사 개념
엔티티 타입: 동일한 애트리뷰트를 갖는 엔티티들의 틀, 스키마와 유사 개념
엔티티 집합: 동일한 애트리뷰트를 갖는 엔티티들의 모임, DB 상태와 유사 개념

▶ 애트리뷰트

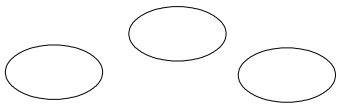
애트리뷰트: 엔티티 구성요소
도메인: 한 애트리뷰트가 가질 수 있는 모든 값들의 집합
ex) 성별 애트리뷰트 도메인: 남, 여)

▶ 단순 애트리뷰트



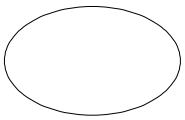
더 이상 다른 애트리뷰트로 나눌 수 없음
대부분의 애트리뷰트
단순하므로 비슷한 것끼리 비교이 안됨

▶ 복합 애트리뷰트



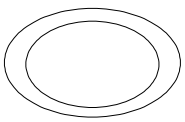
두 개 이상의 애트리뷰트로 이루어짐
동일한 엔티티 타입이나 관계 타입에 속하는 애트리뷰트들 중
밀접하게 연관된 것을 모아놓은 것
복잡하므로 비슷한 것끼리(서브 애트리뷰트) 비교 가능
복합의 부모에는 데이터가 들어가지 않음(자녀가 1개일 수 없음)

▶ 단일값 애트리뷰트



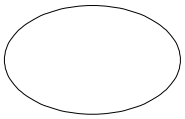
각 엔티티마다 하나의 값을 가짐
ex) 학생은 두 개 이상의 학번을 갖지 않음

▶ 다치 애트리뷰트



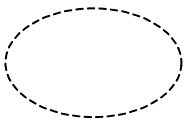
각 엔티티마다 여러 개의 값을 가짐

▶ 저장된 애트리뷰트



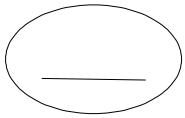
다른 애트리뷰트와 독립적으로 존재
대부분의 애트리뷰트

▶ 유도된 애트리뷰트



다른 애트리뷰트의 값으로부터 얻어짐
ex) 고정 상수 값을 박아 나이를 계산하기 보다 생년월일을 박아 매년 계산되도록 함

▶ 키 애트리뷰트

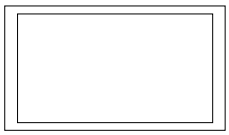


한 엔티티 타입에서 각 엔티티가 유일한 값을 갖는 애트리뷰트(ex) 학생 엔티티의 학번)
복합 애트리뷰트일 수도 있음
엔티티 타입은 한 개 이상의 키를 가져야만 함
키가 두개면 애트리뷰트 하나로 키 역할이 안될 경우 추가 설명 느낌으로 키를 추가

▶ 강한 엔티티 타입

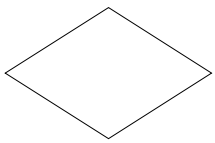
독자적으로 존재
독자적인 키 애트리뷰트를 가짐
실선 직사각형으로 표현

▶ 약한 엔티티 타입



독자적인 키 애트리뷰트 없음
특정 강한 엔티티 타입에 속함
두 줄 실선 직사각형으로 표현
강한 엔티티에서는 키가 될 수 없는 애트리뷰트(이름)라도, 강한 키 + 약한 키가 키 애트리뷰트로 표현 가능하면
강한 엔티티에서는 키가 될 수 없는 애트리뷰트라도 키로 가능
강한 엔티티와 키가 같아진다면, 약한 엔티티로써의 정체성은 없어짐

▶ 관계



두 개 이상의 엔티티들을 특정한 의미로 연관짓는 것
관계가 아닌 애트리뷰트로 표현하면 A가 있으면 B에 있어야 한다 를 표현할 수 없음

▶ 관계 타입
동일한 의미를 가진 관계들의 틀

▶ 카디날리티 제약조건
몇 대 몇?
최대 카디날리티
최소 카디널리티(0: 선택적 참여, 1이상: 의무적 참여/존재 종속)

▶ 참여 제약조건
엔티티 타입의 모든 엔티티가 관계에 참여해야 하는가 여부

▶ 순환적 관계 타입
한 개의 엔티티 타입이 어떤 관계 타입에 서로 다른 역할로 중복 참여하는 것
ex) 부하 -> 상사(둘 다 사원이라는 엔티티)

▶ ER모델 그리기 순서
① 엔티티 종류
② 애트리뷰트 설정
③ 키 애트리뷰트 선정
④ 관계 타입 그리기
⑤ 제약 조건 표시

5. 관계데이터모델

▶ 관계데이터모델의 개념

파일에서는 레코드의 순서가 중요했으나,
릴레이션에서는 투플의 순서가 중요하지 않음
중복되는 투플이 있는 경우 동일한 투플로 간주
데이터를 표의 형태로 관리

★ 제약 조건(도 키 엔 참)

모든 릴레이션들이 만족해야 하는 조건

▶ 도메인 제약조건

애트리뷰트 A의 값은 반드시 A의 도메인에 속하는 원자값이어야 함
ex) 나이라면 1 미만은 아닌지, 음수는 아닌지 확인

▶ 키 제약조건

키 애트리뷰터에 중복된 값이 존재하면 안됨
후보키(Candidate Key): 유일하게 식별 가능한 속성

▶ 엔티티 무결성 제약조건

기본키에 NULL이 포함될 수 없음

▶ 참조 무결성 제약조건

외래키: 다른 릴레이션의 기본 키를 참조하는 애트리뷰트, 두 릴레이션의 관계 표현
외래키 값은 반드시 참조되는 릴레이션의 기본키에 존재해야 함

▶ 삽입

위반 사항
▷ 도메인 제약조건: 삽입되는 튜플 t에서 애트리뷰트의 값이 도메인에 없음(나이에 -값 적음)
▷ 키 제약조건: 중복 키 삽입
▷ 엔티티 무결성 제약조건: 기본 키에 NULL 삽입
▷ 참조 무결성 제약조건: 한국 공대 의대 발생

▶ 삭제

참조 무결성 제약조건: 게임공학과가 폐지되면 참조가 안되서 위반 발생
▷ 삭제 거부(시위)
▷ 삭제되는 튜플을 참조하는 튜플들까지 삭제(폐교)
▷ 외래키 값을 널로 바꿈(무과)
▷ 유효한 튜플 참조(전과)

6. ER -> 관계데이터모델

▶ ER 스키마를 관계 모델의 릴리에션으로 사상

개념적 스키마 -> 논리적 설계 -> 논리적 스키마

☆ 모든 단계들이 다 새로운 릴레이션을 만드는 방법이 있음
일단 새로운 릴레이션을 만들고 N쪽에 선긋기

▶ 단계(유도된 애트리뷰트는 변환 X)

① 정규 엔티티 타입
② 약한 엔티티 타입
③ 2진 1:1 관계 타입
④ 정규 2진 1:N 관계 타입
⑤ 2진 M:N 관계 타입
⑥ 3진 관계 타입
⑦ 다치 애트리뷰트

▶ 1단계

엔티티가 릴레이션으로 바뀜
애트리뷰트는 릴레이션의 내용으로 바뀜
복합 애트리뷰트는 중간 단계 무시

▶ 2단계

약한 엔티티 타입도 강한 엔티티 타입변환과 동일
강한 애트리뷰트의 키 애트리뷰트를 앞에다 붙임

▶ 3단계

① 외래키 형태로 릴레이션에 추가
② 외래키 형태로 릴레이션에 추가(반대쪽으로)
③ 새로운 릴레이션 추가(키는 어느 한쪽에만 그거야 함, 둘 다 긋는건 안됨)
④ 표 하나로 합쳐버리기(키는 새로 만들거나, 앵간하면 쓰지마)

▶ 4단계

① N쪽에다가 1쪽의 기본 키를 추가
② 양쪽의 기본키를 가져와서 새로운 릴레이션 추가(키는 N쪽에서 온 키)

순환 관계타입: N쪽에 1쪽을 추가

▶ 5단계

양쪽의 기본키를 가져와서 새로운 릴레이션 추가(둘 다 선긋기)

▶ 6단계

각각의 기본키를 가져와서 새로운 릴레이션 추가(N에서 온 것에 선긋기)

▶ 7단계

일단 기본 키로 릴레이션을 만들고
다치는 기본 키로 만든 다음에 여러 값을 넣고 둘 다 선긋기