**데이터베이스 정리**

목차

1. 소개
2. Relation model
3. SQL
4. Intermediate SQL
5. Advanced SQL(디비 이용해서 다른 프로그램과 사용)
6. Formal Relational Query languages
7. Entity-Relationship model
8. Relational Database design (정규화)
9. Storage and File Structure
10. 소개

**파일시스템**

파일(데이터의 모임)을 저장 장치에 저장하고 사용하기 위한 일종의 규칙이나 체계를 뜻하며 파일과 관련된 기능을 수행하는 시스템이다.

파일 시스템은 파일에 접근하는 방식이 응용 프로그램 내에 표현되므로 응용 프로그램과 데이터 간의 의존관계가 존재하게 되어 데이터의 구조, 접근 방법이 변경되면 기존의 프로그램과 데이터를 함께 변경해야한다.

즉 데이터 정의가 응용 프로그램에 내포되어 있음. 프로그램에서 데이터를 접근하고 조작하는 것 이외에 별도의 제어가 없음.

이로 인해 많은 단점들이 발생

1. 중복된 데이터가 발생할 수 있으며 작성 형식의 일관성이 떨어진다.
2. 중복된 데이터 변경 시 일부만 변경되는 경우 데이터 간 불일치 발생
3. 응용 프로그램과 데이터간 의존관계가 존재하므로 새로운 일을 하기 위해서는 새로운 프로그램을 만들어야함. 즉 data 접근성이 떨어진다.
4. 다수의 파일과 여러 형식 사이에 데이터 고립이 발생
5. 데이터 무결성 유지 어려움 – 파일 내의 데이터가 만족시켜야 하는 무결성 제약조건들을 명시하기 위해 프로그램 내부에 표현해야 하므로 무결성 유지 어려움. 새로운 제약조건 추가 및 수정이 어렵다.
6. 원자성을 지원하지 않는다. – 실행을 중간까지만 할 수 없다. 항상 완성하거나 못하거나 둘 중 하나.
7. 다수 사용자를 위한 동시성 제어가 제공되지 않는다. (충돌 제어가 불가능)
8. 보안 미흡 – 파일 단위로만 검색, 갱신, 실행 권한을 부여할 수 있어서 사용자의 권한에 따른 세밀한 접근 제어를 시행하기 어렵다. (부분적 권한 막는 것이 불가능)

과거에는 직접 file에 대해 응용프로그램을 설계했으며 현재에도 데이터가 많지 않거나 DBMS가 적절하지 않은 경우(응용이 단순, 다수 사용자의 데이터 접근 필요하지 않은 경우 등) DBMS를 사용하지 않는다. (ex. MS Word, 한글, CAD 등)

**DBMS(Database Management System): 데이터베이스 관리 시스템**

데이터베이스를 관리하며 응용프로그램들이 데이터베이스를 공유하며 사용할 수 있는 환경을 제공하는 소프트웨어

종류: Oracle, MySQL 등

DB: 여러 사람에 의해 공유되어 사용될 목적으로 통합하여 관리되는 데이터들의 집합. 자료항목의 중복을 없애고 자료를 구조화하여 저장함으로써 자료 검색과 갱신의 효율성을 높인다.

DBMS의 장점은 곧 File system의 단점의 역

**Database system의 구조 (추상화 level)**

Physical level – 디스크에 저장하는 방법을 묘사한 단계

Logical level (Conceptual level) – DB에 저장된 데이터와 데이터간 관계를 묘사한 단계 / 즉 table 구조를 표시하는 단계로 우리가 실질적으로 작성하는 단계

View level – 각 사용자에게 DB를 보여주는 단계로 보안 목적에 의해 전제 DB의 모습 중 일부분만을 사용자에게 보여준다.

Schema: 디비의 논리적 structure로 프로그래밍 언어에서 변수의 type과 유사하다.

Instance: 디비의 실제 content로 프로그래밍 언어에서 변수의 값과 유사하다.

Database system의 구조가 3가지 level로 잘 나뉘어져 있기 때문에 physical data independence를 만족한다. 프로그램은 logical schema에 의존하므로 저장위치나 파티션(physical schema)이 바껴도 위 level은 영향을 받지 않는다. 즉 프로그램을 바꾸지 않아도 된다.

**Data Models (logical level models)**

데이터 모델링할 때 필요 요소: Data, Data간 관계, Data 의미(ex. Salary가 원인지 달러인지), Data 제약조건

Model 유형: Relational model, Entity-Relationship data model, Object-based data models, Semistructured data model(XML) 등

**Data Language**

DML (Data Manipulation Language): data 조작하는 언어로 정의된 데이터베이스에 입력된 레코드를 조회, 수정 또는 삭제하는 등의 역할을 하는 언어이다. 데이터베이스 사용자가 응용프로그램이나 질의어를 통하여 저장된 데이터를 실질적으로 처리하는데 사용

Query language(SQL)로도 잘 알려져 있다.

언어에는 C++, JAVA가 해당되는 Procedural 유형과 SQL이 해당되는 Declarative 유형이 존재.

DDL (Data Definition Langauge): DB를 정의하는 언어로 데이터를 생성, 수정, 삭제하는 등의 데이터의 전체 골격을 결정하는 역할은 하는 언어이다. 데이터 베이스 관리자나 설계자가 사용

Data dictionary(metadata를 포함한다)에 저장되는 테이블 템블렛을 생성하는 언어.

Metadata – 디비 스케마, 무결성 제약조건, 참조무결성, 권한

DCL (Data Control Language): 데이터베이스에 접근하거나 객체에 권한을 주는 등의 역할을 하는 언어. 데이터를 제어하는 언어로 데이터의 보안, 무결성, 회복, 병행 수행제어 등을 정의하는데 사용한다.

**SQL**

세계적으로 사용되는 선언적 언어.

Ex) select name from instructor where instructor.ID = ‘22222’

응용 프로그램들은 db접근시 내장 or 인터페이스를 사용한다.

DB를 디자인할 때는 데이터의 중복을 피해서 table을 쪼개 저장하는 것이 좋다. 효율 높음

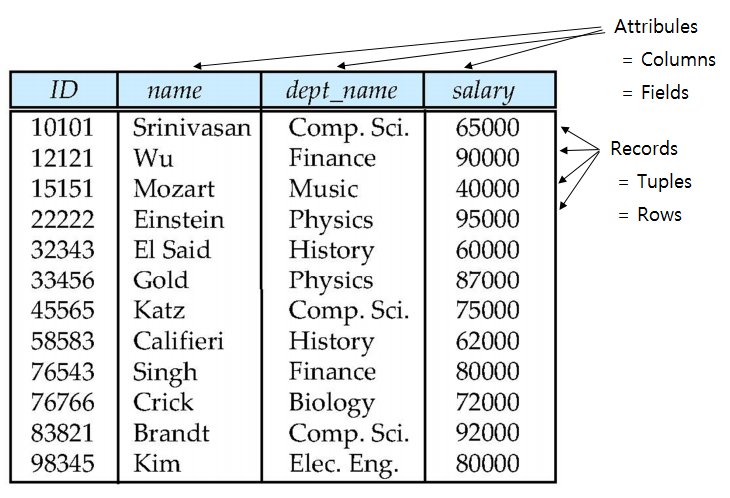
2장 – Relational model

Logical level models 중 하나이다.

Implementation model로 logical level에서 정의한 형태대로(레코드 단위로) 파일에 기록된다.

객체 속성, 객체간 관계에 관한 정보를 table 형식으로 저장

**Example of a Relation (테이블 형태로 저장된다.)**



**Attribute Types**

Relation의 각 속성은 domain이라고 하는 허가된 값의 집합을 가진다. (변수의 값의 범위, 글자 수 제한 등)

속성 값들은 atomic(원자성)이 요구된다. (각 cell에 값이 하나씩 존재)

Null은 모든 도메인에 포함되고 unknown 또는 not applicable의 의미를 갖는다. 따라서 많은 연산의 정의에서 문제를 불러일으킨다.

**Relation Schema and Instance**

Relation schema: 속성 A1, A2, …, An이 있을 때 R=(A1, A2, …, An)으로 표현된다. 위 테이블을 예시로 볼 경우 instructor = (ID, name, dept\_name, salary)가 위 테이블의 relation schema이다.

Relation Instance: 도메인 D1, D2, …, Dn이 있을 때, 하나의 relation r은 D1 X D2 X … X Dn 조합의 subset으로 이루어져 있다.

A relation is a set of n-tuples (a1, a2, …, an) where each ai ∈ Di

Tuple vs Set

* Tuple은 요소의 순서에 의미가 있다. 그리고 데이터 중복 가능
* Set은 요소의 순서에 의미가 없다. 그리고 데이터 중복이 불가능 (집합이므로)

relation에서 tuple의 순서는 의미가 없다.

Database는 다수의 relation으로 구성된다. 하나의 relation으로 구성할 수 있지만 정보의 중복과 null값을 최대한 피하기 위해 쪼개는 것이 좋다. (뒤 부분에서 다룰 것)

**Keys – 중복없이 데이터를 다른 데이터들과 구별해주는 식별자**

Superkey(슈퍼키): uniqueness를 만족하는 attributes의 집합 / 중복없이 데이터를 다른 데이터들과 구별해줄 수 있는 모든 가능한 속성 정보들의 집합

Candidate key(후보키): uniqueness를 만족하면서 irreducibility를 만족하는 속성들의 집합 / superkey 중 조합 개수가 가장 작은 조합들의 집합

Primary key: 중복없이 데이터를 다른 데이터들과 구별해주는 식별자로 실제 table에서 사용되는 key를 의미 / 후보키들 중 하나를 선택한다. Superkey -> candidate key -> primary key 순으로 key를 찾는다.

Foreign key(외래키): 다른 relation schema의 primary key를 참조하는 속성으로 정의 (즉 나에게는 외래키, 참조대상에게는 primary key)

Referencing relation – 참조하는 table (다른 table을 참조하려는 table)

Referenced relation – 참조당하는 table (참조대상)

Referential integrity constraint: Value in one relation must appear in another / referenced relation에서 어떤 tuple을 삭제하고자 할 때, 참조 무결성을 만족시키기 위해 그 tuple을 다른 relation에서 참조하지 않는지 확인하고 나서 삭제해야 한다.

**Relational Query Languages (6장에서 다룰 내용)**

**아래 3가지 모두 pure language에 해당**

Relational algebra(관계 대수)

* 절차식 언어이다.
* 하나 또는 두 개의 릴레이션을 입력으로 받아, 결과로서 새로운 릴레이션을 생성해 내는 연산들의 집합으로 구성
* 일종의 ‘방정식’ / 원하는 데이터가 있으면 그 데이터를 얻기까지 어떤 과정으로 처리되는지와 그 방법을 표현한다.
* SQL은 relational algebra의 feature들에 기반한다. SQL은 선언적 언어지만 그 속은 절차적으로 처리된다.

Tupel relational calculus(튜플 관계 해석)

* 비절차적 언어 / 가로 관계형 논리학
* 결과를 얻기 위한 특정 대수 프로시저를 사용하지 않고, 결과를 얻기 위한 술어 논리를 사용
* 일종의 “함수” / 입력으로 데이터를 넣으면 출력으로 어떤 데이터가 나오는지와 상태만 표현한다.

Domain relational calculus

* 비절차식 언어 / 세로 관계형 논리학

3장 – SQL

**Domain Types**

* char(n): 길이 n으로 고정된 문자열
* varchar(n): 최대 길이 n으로 정해진 문자열. 최대 길이를 넘을 경우 문자열이 잘린다.
* int: 정수형
* smallint: small 정수형
* numeric(p,d): 실수. Ex) numeric(8, 2): 전제 8자리, 소수 2자리
* real, double precision: 실수
* float(n): n은 유효 숫자 개수를 의미. Ex) float(5): 123456789.3412 -> 1.2346 \* 10^8

**table구조 생성, 변경, 삭제**

**생성**

**create table** *instructor* (  
 *ID* **char**(5),  
 *name* **varchar**(20) **not null, 🡨 name부분에 null값 삽입 시 오류 발생***dept\_name* **varchar**(20),  
 *salary* **numeric**(8,2))

**insert into** *instructor* **values** (‘10211’, ’Smith’, ’Biology’, 66000);

table 생성시 무결성 제약조건(Integrity Constraints)

* not null, primary key, foreign key (Am, …, An) references r
* primary key로 설정된 속성은 자동으로 not null을 보장한다.
* Example: Declare *ID* as the primary key for *instructor*
* **create table** *instructor* (  
   *ID* **char**(5),  
   *name* **varchar**(20) **not null,***dept\_name* **varchar**(20),  
   *salary* **numeric**(8,2),  
   **primary key** (*ID*),  
   **foreign key** *(dept\_name*) **references** *department)*

**삭제**

Drop table 테이블명 [cascade constraint];

cascade constraint 옵션: 해당 테이블과 관계가 있었던 참조되는 제약조건에 대해서도 삭제

**변경** – 컬럼을 추가/삭제 또는 제약조건을 추가/삭제 작업을 한다.

Alter table r add A D: 속성 A, D를 relation r에 추가. 이때 새로 추가된 속성 값은 모두 null

Alter table r drop A: 속성 A를 relation r에서 삭제

**Basic Query Structure**

Select A1, A2, …, An 🡨 속성 (모든 속성을 표시하고 싶을 경우 속성 부분에 \*를 입력)

From r1, r2, …, rm 🡨 table

Where P 🡨 조건문

SQL query의 결과는 relation

sql 이름들은 대소문자 상관없다. 모두 동일 단어로 생각

SELECT문

SQL는 relation에서 중복을 허용한다. 따라서 중복을 제거하기 위해서는 “select 속성”형식이 아닌 “select distinct 속성”형식으로 작성해야 한다. (<-> 중복을 허용하는 keyword로 all이 존재. 초기 옵션이 all이므로 생략해도 무방하다.)

select문에서는 수식 표현이 가능(+,/,-,\*,상수, tuple 속성) ex) Am/12 -> 수식에 해당하는 속성 값이 출력된다.

WHERE 문

where문은 and, or, not을 이용해 식을 연결할 수 있다.

FROM 문

From문에는 여러 relation을 포함할 수 있으며 Cartesian product 형태로 출력됨. 모든 조합을 출력한다. Relation A x Relation B에서 가능한 모든 튜플 조합

from A, B 와 같이 여러 relation을 포함할 경우 이 relation들을 join한다는 것을 의미 – cartesian product

natural join: select \* from A natural join B 입력 시 같은 이름을 가진 모든 속성들에 대해 A, B에서 같은 값들을 가진 tuple끼리 매치한다.

Natural join은 속성 이름이 같으면 무조건 같은 속성으로 생각한다. (실제로 다른 속성일지라도)

**Rename Operation (별칭 만들기)**

Relation과 속성에 대해 별칭을 만들 수 있다. 같은 relation을 여러 번 이용해 query문을 작성하게 될 경우 별칭을 써서 구분할 수 있다.

Old-name as new-name

**String Operation**

문자열 비교 시 매칭 연산이 존재. where문에서 사용 (%: substring, \_: 한 글자)

Where name like ‘%dar%’: 이름에 dar 들어가는 tuple 다 찾기.

**Tuple을 특정 속성(속성들)에 대해 정렬하기**

기본 sql문 작성 후 맨 하단에 작성. Order by 속성1, 속성2, … [desc | asc] / asc가 기본 정렬방식

**Set Operations (두 select문 연결)**

Union:합집합, or | intersect:교집합, and | except:차집합, but not

위 3가지 연산을 사용할 경우 자동으로 결과에 중복이 제거된다.

중복을 포함하고 싶은 경우 위 연산 뒤에 all을 붙여서 사용한다.

Multiset version: union all, intersect all, except all

**Null Values**

Null: unknown value or 존재하지 않는 값을 의미

Where salary is null 조건문 작성 시 null값을 검사할 수 있다.

수식에서 null이 존재할 경우 결과는 항상 null (ex. Null + 5 = null)

null값과 three valued logic

* Null이 포함된 비교연산은 unknown(false)를 반환
* Or: true이외에 모두 결과값 unknown, true의 결과값은 true
* AND: false의 결과값은 false, 이외에 모든 결과값 unknown
* NOT: not unknown = unknown

중복제거나 grouping할 때, null또한 다른 값들과 같이 다뤄진다. 2개의 null은 같은 것으로 생각(sql에서)

**Aggregate Functions (집단 함수)**

Avg, min, max, sum, count 가 존재, 특정 속성값에 해당하는 column값을 모두 함수에 맞게 계산해서 결과값을 출력한다.

Select avg(salary): salary속성 column에 속한 where조건에 해당하는 모든 값들의 평균을 구해 table로 표현

Select count(\*): count경우 tuple의 개수를 출력하므로 \*이외에 다른 속성이 와도 똑같다.

Group By

기본 SQL문에 의해 나타난 결과 table을 특정 속성에 대해 그룹별로 결과를 표시한다.

이때 select문에는 group되는 기준 속성과 aggregate func이외에 다른 속성들은 들어가면 안된다.

Having Clause

그룹의 조건으로 group by밑에 작성한다.

Ex) select ~ from ~ where ~(tuple의 조건)

group by ~ (위 조건들을 통해 나온 tuple들을 그룹)

having ~ (그룹의 조건 ex. Avg(salary) > 42000)

count(\*)을 제외한 모든 Aggregate 연산은 null 값을 무시한다. 단, non-null인 값이 존재하지 않을 경우 결과에 null 출력 (count(표현식)인 경우 non-null이 없으면 0을 출력)

**Nested subqueries: 중첩된 sub질의**

Set comparison: all, some | where조건문에서 하나의 tuple의 값과 다른 sub질의에 의해 생성된 tuple들의 값과 비교 시 사용. 즉 하나의 tuple과 여러 tuple의 값을 비교할 때 사용한다.

All: 모든 경우 만족

Same: any or at least one (하나라도 있으면 만족)

Empty relations에 대한 test : exists, not exists | where not exists (sub 질의) 형태로 작성. Exist 경우 sub질의에 의해 생성된 tuple이 하나로 존재하면 true, not exist경우 sub질의에 의해 생성된 tuple이 없을 경우 즉 공집합일 경우 true

바깥 질의와 연관된 sub질의를 correlated subquery라 한다.

중복 tuple 존재 test : unique | where unique (sub 질의) 형태. 중복이 있는지 check하여 없으면 true, 있으면 false를 반환한다. Distinct와 구별할 것(이는 중복을 제거하는 것)

**Derived Relations (유도된 table)**

from절에 sub질의가 포함되어 sub질의에 의해 유도된 table을 바깥 질의에서 사용한다.

**With Clause**

임시테이블을 만드는 것으로 view와 쓰임새가 비슷하지만 한번 만들면 drop할 때까지 여러 번 사용가능한 view와 달리 with절은 한번 실행할 쿼리문내에 정의되며 해당 쿼리문안에서만 실행된다. (하나의 query문 안에서는 재사용 가능)

사용이유: 복잡한 sql에서 동일 블록에 대해 반복적으로 sql문을 사용하는 경우 그 블록에 이름을 부여하여 재사용 할 수 있게 함으로서 쿼리 성능을 높일 수 있는데 with절을 이용하여 미리 이름을 부여해서 query block을 만들 수 있다. 자주 실행되는 경우 한번만 parsing되고 plan 계획이 수립되므로 쿼리의 성능향상에 도움이 된다.

**Scalar Subquery**

select문에서 사용되는 subquery(?)

select문에서 sub질의 값이 여러 개면 table이 된다. 즉 relational table의 각 속성값은 atomic하다는 것을 위반하게 되므로 select문에 sub질의가 존재할 경우 결과 table은 tuple이 1개 존재해야 한다. 이를 scalar subquery라 부름.

**Modification of the DB**

Deletion

Delete from r [where ~]: table속 tuple을 삭제한다. 모든 tuple을 삭제해도 table은 유지된다. (drop r 경우 table 자체를 삭제한다. 구분하기)

Insertion

Insert into r values (속성1, … , 속성n); : table속에 새로운 tuple을 삽입한다.

Update

Update r set 변경할 속성 값 where 조건: 속성값을 바꿈

같은 속성에 대해 조건별로 변경될 값이 다른 경우 update문을 두 번 쓴다 🡪 이 경우 이미 업데이트 된 값이 또다시 이후 조건에 만족되어 이중 업데이트 될 수 있다. 이러한 문제를 위해 조건별 update해야할 경우 case문을 사용

4장 – Intermediate SQL

**Outer Join**

정보 손실을 피하기(방지) 위한 join연산의 확장된 연산, 일반 natural join 사용시 정보 손실이 발생하 수 있다. (서로 매칭 안되는 정보를 가지고 있을 경우)

서로 매칭 안되는 정보를 가지고 있을 때 null을 사용해서 그 값을 채운다.

종류

* Natural Left Outer Join: 왼쪽 table정보 모두 적기
* Natural Right Outer Join: 오른쪽 table정보 모두 적기
* Natural Full outer join: 양쪽 table정보 모두 적기

**Views**

사용자에게 접근이 허용된 자료만을 제한적으로 보여주기 위해 하나 이상의 기본 테이블로부터 유도된, 이름을 가지는 가상 테이블이다.

뷰는 저장장치 내에 물리적으로 존재하지 않지만 사용자에게 있는 것처럼 간주된다. 즉 view는 data가 저장된 것이 아니라 정의한 식만 저장되며 data는 복사되지 않는다. 사용자가 query문 작성시 view를 사용할 경우 view에 저장된 식으로 대체되어 수행된다.

뷰는 데이터 보정작업, 처리과정 시험 등 임시적인 작업을 위한 용도로 활용된다. 여러 번 사용되는 질의를 view로 만들면 사용시 편리하다.

조인문의 사용 최소화로 사용상의 편의성을 최대화한다.

정의: create view v as <query expression> | v: view name

특징

* 뷰는 기본테이블로부터 유도된 테이블이기 때문에 기본 테이블과 같은 형태의 구조를 사용하며, 조작도 기본 테이블과 거의 같다.
* 뷰는 가상 테이블이기 때문에 물리적으로 구현되어 있지 않다.
* 데이터의 논리적 독립성을 제공할 수 있다.
* 필요한 데이터만 뷰로 정의해서 처리할 수 있기 때문에 관리가 용이하고 명령문이 간단해진다.
* 뷰를 통해서만 데이터에 접근하게 하면 뷰에 나타나지 않는 데이터를 안전하게 보호하는 효율적인 기법으로 사용할 수 있다.
* 기본 테이블의 기본키를 포함한 속성(열) 집합으로 뷰를 구성해야지만 삽입, 삭제, 갱신 그리고 연산이 가능하다.
* 일단 정의된 뷰는 다른 뷰의 정의에 기초가 될 수 있다.
* 뷰가 정의된 기본 테이블이나 뷰를 삭제하면 그 테이블이나 뷰를 기초로 정의된 다른 뷰도 자동으로 삭제된다.
* 노란 표시: 장점을 의미
* 단점: 독립적인 인덱스를 가질 수 없으며 뷰의 정의를 변경할 수 없고 뷰로 구성된 내용에 대한 삽입, 삭제, 갱신, 연산에 제약이 따른다.
* View update 시 제약: 2개 이상의 relation을 join한 경우 금지, select문에는 오직 해당 relation에 존재하는 속성이름만 포함되어야한다. (수식이나 애매한 경우 금지), select문에 목록화된 속성값이 아닌 경우 not null로 설정되어 있으면 update 금지, 질의는 group by or having 문을 가질 수 없다.

**Transactions**

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

특징

데이터베이스 시스템에서 병행 제어 및 회복 작업시 처리되는 작업의 논리적 단위이다.

사용자가 시스템에 대한 서비스 요구 시 시스템이 응답하기 위한 상태 변환 과정의 작업단위이다.

하나의 트랜잭션은 commit되거나 Rollback된다. (원자성)

성질

* Atomicity(원자성): 트랜잭션 내의 모든 명령은 반드시 완벽히 수행되어야 하며, 어느 부분 하나라도 오류가 발생하면 한 트랜잭션 내의 모든 명령이 전부 취소되어야 한다.
* Consistency(일관성): 시스템이 가지고 있는 고정요소는 트랜잭션 수행전과 완료 후의 상태가 같아야 한다. 즉 실행이 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 변환한다.
* Isolation(독립성, 격리성): 둘 이상의 트랜잭션이 동시에 병행 실행되는 경우 서로에게 영향을 끼치면 안되므로 분리되어야 한다. 하나의 트랜잭션이 실행 중일 경우 다른 트랜잭션의 연산이 끼어들 수 없다. 수행 중인 트랜잭션은 완전히 완료될 때까지 다른 트랜잭션에서 수행 결과를 참조할 수 없다.
* Durability(영속성, 지속성): 성공적으로 완료된 트랜잭션의 결과는 시스템이 고장나도 영구적으로 반영되야 한다.

commit연산: 트랜잭션이 성공적으로 끝나고 데이터베이스가 다시 일관된 상태에 있을 때, 해당 트랜잭션에 의한 갱신 연산이 완료된 것을 관리자에게 알려주는 연산

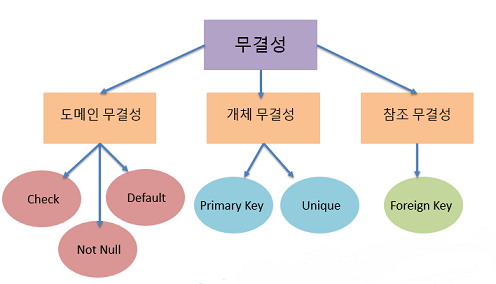
Rollback연산: 하나의 트랜잭션이 비정상적으로 종료되어 데이터베이스의 일관성을 깨뜨렸을 때, 트랜잭션의 원자성을 구현하기 위해 모든 연산을 취소(Undo)하는 연산이다. 해당 트랜잭션을 재시작하거나 폐기한다.

**Integrity Constraints(무결성 제약조건)**

무결성이란 데이터베이스에 저장된 데이터 값과 그것이 표현하는 현실 세계의 실제값이 일치하는 정확성을 의미한다.

무결성 제약조건은 일관된 데이터베이스 상태를 정의하는 규칙들을 묵시적 또는 명시적으로 정의하는 것이다. 데이터베이스에 들어있는 데이터의 정확성과 일관성을 보장하기 위해 부정확한 자료가 데이터베이스 내에 저장되는 것을 방지하기 위한 제약 조건을 의미한다.

무결성의 종류: 개체 무결성, 참조 무결성, 도메인 무결성



도메인 무결성

* 속성이 정의되어 있는 영역을 벗어나지 않도록 규정하는 것
* Check, default, not null 제약이 있다.
* Check 제약: 값의 범위를 정해 놓는 제약, 조건에 맞는 데이터만을 입력 받을 수 있다.
* Default 제약: 입력 값을 지정하지 않았을 때 열의 값이 기본값으로 입력되도록 지정하는 값
* Not null 제약: 데이터를 입력받을 때 NULL을 허용하지 않는다.

개체 무결성

* 하나의 테이블에 중복된 행이 존재하지 않도록 규정하는 것
* 기본키 제약이라고도 부른다.
* UNIQUE, Primary key가 있다. (두 제약을 구분해둘 것)
* Unique 제약: null값이 허용되며 null을 포함해서 값의 중복이 허용되지 않는 제약 (이 제약이 붙은 속성들의 조합은 후보 키 집합을 형성한다)
* Primary key: null값이 허용되지 않으며 값의 중복이 허용되지 않는 제약이다. Unique키와 not null 제약이 합쳐진 것

참조 무결성

* 행을 입력, 수정, 삭제할 때 연관되는 다른 테이블과의 데이터가 정확하게 유지되도록 규정하는 것이다.
* 즉, 참조 관계에 있는 두 테이블의 데이터가 항상 일관된 값을 가져야 한다. (릴레이션 간의 참조 관계를 선언하는 제약조건)
* Foreign key 제약이 있다.
* Foreign key: 다른 테이블의 데이터를 참조할 때 없는 값을 참조할 수 없도록 제약을 주는 것으로 참조될 테이블이 먼저 만들어지고 값이 입력되어야 한다.
* 이때, 참조될 열의 값은 참조될 테이블에서 primary key로 설정되어 있어야한다
* Cascading actions: 삭제와 갱신이 있으며, 이 기능을 사용하면 참조되는 relation에 값이 삭제 또는 갱신될 경우 그 값을 참조하는 relation에서 해당 값들 또한 삭제 또는 갱신되도록 한다.

**SQL DATA types and Schemas**

Large-object: 사진, 비디오, cad 파일 등과 같은 큰 object들은 large object로써 저장된다.

types: Blob(binary large object), Clob(character large object)

**Authorization**

Parts of the database(객체 권한): read, insert, update, delete (객체를 조작할 수 있는 권한으로 객체의 소유주나 관리자 계정으로 부여 가능)

To modify the database schema(시스템 권한): index(인덱스의 삽입 삭제), resources(릴레이션 생성), alteration(릴레이션의 속성을 더하거나 삭제), drop(릴레이션 삭제) (db접근 및 각종 객체를 생성할 수 있는 권한으로 관리자 계정에 부여되는 권한)

sql에서 authorization을 명시할 때 grant문장을 사용한다.

Grant <privilege list> on <relation name or view name> to <user list>

sql에서 객체 권한을 회수할 경우 revoke 문장을 사용

revoke <privilege list> on <relation name or view name> from <user list>

5장

**JDBC and ODBC (Dynamic SQL) – open system**

프로그램이 데이터베이스 서버와 interact하는 것을 돕는 API | client와 server가 api를 통해 주고받기가 가능하다.

JDBC(JAVA Database connectivity): java에서 DB에 접속할 수 있도록 하는 API, 데이터베이스에서 자료를 쿼리하거나 업데이트하는 방법을 제공한다.

ODBC(Open database connectivity): 모든 응용 프로그램(프로그래밍 언어 무관)에서 모든 데이터베이스 시스템과 통신하는 데 사용할 수 있는 개방형 인터페이스. JDBC가 java만 지원하는 것과 다르게 ODBC는 언어에 대해 독립적이다.

데이터베이스 서버에 보낸 sql 명령에 대한 결과는 tuple 단위로 하나씩 가져와진다. (Fetch)

**Embeded SQL (내장 SQL, 특정 DBMS를 application에 넣는다.)**

응용 프로그램 내에 데이터베이스에서 사용하는 데이터를 정의하거나 질의하는 SQL문자을 내포하여 프로그램이 실행될 때 함께 실행되도록 호스트 프로그램언어(C, JAVA, Pathon, Visual Basic)로 만든 프로그램에 삽입된 SQL이다.

호스트 언어의 실행문과 구분시키는 방법

명령어 구분: C/C++에서는 내장 SQL문은 $와 ; 사이에 기술 | visual basic에서는 내장 sql문 앞에 “EXEC SQL”을 기술

변수의 구분: 내장 SQL에서 사용하는 호스트 변수는 변수 앞에 콜론: 문자를 붙인다

Cursor(커서): 내장 SQL문의 실행 결과로 반환된 복수 개의 튜플들을 접근할 수 있도록 해주는 개념. 질의 실행 결과로 반환된 테이블의 튜플들을 순서대로 가리키는 튜플에 대한 포인터로 생각할 수 있다.

**Triggers**

특정 테이블에 insert, delete, update와 같은 dml문이 수행됐을 때, 시스템에 의해 데이터베이스에서 자동으로 수행되는 문장이다. 사용자가 직접 호출하는 것이 아님.

트리거는 테이블과 뷰 데이터베이스 작업을 대상으로 정의할 수 있으며, 전체 트랜잭션 작업에 대해 발생되는 트리거와 각 행에 대해 발생되는 트리거가 있다.

트리거를 디자인하기 위해서는 트리거가 실행될 condition과 실행 시 action을 명시해야 한다.

6장 – Formal Relational Query Languages

**Relational Algebra**

절차적 언어이며, 6가지의 기본 연산들이 있다. (select(σ), project(П), union(∪), set difference(-), Cartesian product(x), rename(ρ))

input으로 1개 or 2개 relation이 들어오며 결과로 하나의 새로운 relation을 생성한다.

Select – sql에서 where절과 동일

*σ* *p*(*r*) = {t| t ∈ r and p(t)}

튜플 단위로 선택한다. 해당 조건(p)에 맞는 튜플들을 결과로 출력

Project – sql에서 select절과 동일

 (Am은 테이블 r에서 선택할 속성들을 의미한다)

Relation r에서 원하는 속성들에 해당하는 열만 결과로 표시한다. 이때 relation은 set(집합)이므로 중복되는 행은 제거하고 표시한다.

Union

*r* ∪ *s* = {*t* | *t* ∈ *r* or *t* ∈ *s*}

두 relation의 합집합을 결과로 출력

사용시 주의점: 칼럼명이 같아야하며(개수도 동일) 칼럼별 데이터타입이 같아야한다.

Set difference

*r – s* = {*t* | *t* ∈ *r* **and** t ∉ *s*}

차집합을 결과로 출력

사용시 주의사항은 union의 주의사항과 동일

Cartesian Product

*r* x *s* = {*t q* | *t* ∈ *r* **and** *q* ∈ *s*}

두 relation의 모든 조합을 결과로 출력

Rename

 E 테이블의 이름을 x로 바꾼다는 것으로 x(a1, a2, …, an)으로 왼쪽 수식과 같이 표현할 경우 해당 테이블의 속성들의 이름도 변경할 수 있다.

Query 예제

find instructor salaries that are less than some other instructor salary (i.e. not maximum) - using a copy of *instructor* under a new name *d*

🡺 ∏*instructor.salary* (σ *instructor.salary < d.salary* (*instructor x* ρ*d* *(instructor*)))

추가적인 연산자들: set intersection(∩), Natural join, Assignment, Outer join, division | 이 연산들은 기본 연산자들을 이용해 표시할 수 있다.)

Set intersection

*r* ∩ *s* = { *t* | *t* ∈ *r* **and** *t* ∈ *s* }

두 relation의 교집합을 결과로 출력

사용시 주의사항은 union의 주의사항과 동일

*r* ∩ *s* = *r* – (*r* – *s*)

Natural join



두 relation의 공통 속성들의 값이 같은 tuple들만 매칭해 결과로 출력한다. (즉, 두 테이블을 cartesian product 한 결과 테이블에서 공통 속성들의 값이 같은 tuple들만을 이용해 새로운 테이블을 생성하는 것이다. (cartesian product와 구분해서 알아 둘 것))

두 relation에 존재하는 속성들 중 속성 이름이 같으면 값이 같다고 생각한다.(같은 속성으로 취급)

결합법칙, 교환법칙이 성립한다.

*R* = (*A, B, C, D*), *S* = (*E, B, D*) 그리고 Result schema = (A, B, C, D, E)일 때

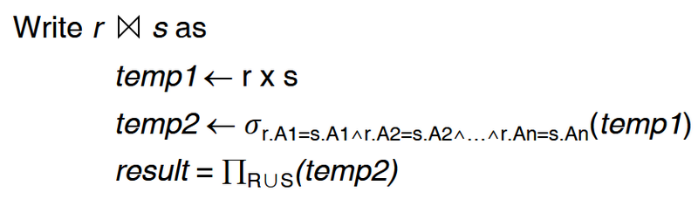
Natural join은 ∏*r.A, r.B, r.C, r.D, s.E* (σ*r.B = s.B* ∧ *r.D = s.D* (*r*  x *s*)) 로 표현 가능하다.

Theta join:  | natural join 경우 theta 자리에 등호표시가 들어가지만 (속성값이 같은 tuple들만 가져오므로) theta join은 theta자리에 등호 이외에 조건들이 들어가는 것이 특징이다.

Assignment

Assignment 연산(←)은 복잡한 쿼리문을 표현하기 위한 편리한 방법을 제공한다.

SQL에서 with문과 같이 임시 table(relation)을 만드는 것이다.



Outer join

정보의 손실을 피하기 위한 join 연산()의 확장 버전으로 null을 이용하며,

Left outer join(), right outer join(), full outer join()이 있다.

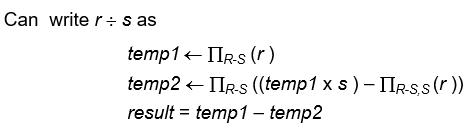


Division

S ⊂ R인 r(R)과 s(S)가 주어질 때 (R과 S는 relation r과 s의 속성들), r ÷ s는 t x s ⊆ r 인 가장 큰 relation t(R-S) 이다.

r ÷ s는 r table에서 s를 모두 포함하는 row들을 출력하는 것이다.

예를 들어, r테이블에 학생id와 과목명이 존재하고 s테이블에 특정 과의 과목명이 존재한다고 할 때, r ÷ s는 r테이블의 학생들 중 s테이블에 존재하는 모든 과목을 수강한 학생 id들이 출력된 table이다.



확장된 relational algebra operations

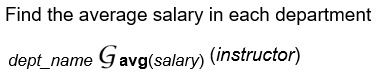
Generalized projection

속성 표현하는 자리에 arithmetic function을 허용하여 표현 가능하게 한다. 즉 salary/12같은 것

Aggregate Functions

Avg, min, max, sum이 존재하며 여러 값들을 받으면 하나의 값으로 결과를 출력한다. (realtion)

 : Gi(그룹할 속성들), Fi(Ai)(group by 함수)



Multiset relational algebra – pure relational algebra는 모두 중복을 제거하지만 multiset은 중복을 제거하지 않는다.

SQL문장을 relational algebra로 표현 가능하며 그 반대도 가능하다.

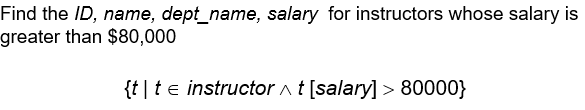
**Tuple Relational Calculus (행)**

(relational algebra(관계 대수)는 절차적, relational calculus(관계 해석)는 선언적)

선언적 query language이며 {t | P(t) }와 같은 형태를 갖는 query이다.

t는 tuple을 의미하며 각 tuple에 대해 조건 P에 대해 참인 모든 tuple들의 집합을 의미한다.

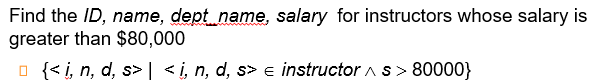
Predicate Calculus Formula(술어논리)

* 속성과 상수를 사용
* 비교 연산자를 사용
* 각 공식을 연결하기위한 연산자: and, or, not을 사용
* Implicatioin (=>): x=>y, if x is true, then y is true
* Quantifiers: ∃ *t* ∈ *r* (*Q* (*t* ))≡ 릴레이션 r에 조건 Q(t)를 만족하는 튜플이 하나라도 있으면 true | ∀*t* ∈ *r* (*Q* (*t* )) ≡ 릴레이션 r에서 모든 튜플이 조건 Q(t)를 만족하면 true
* 
* 위 query는 sql문에서 select \* from instructor where salary > 80000와 같다.

**Domain Relational Calculus (열)**

선언적 query language이며 { < *x*1*, x*2*, …, xn* > | *P* (*x*1, *x*2*, …, xn*)} 와 같은 형태를 갖는 쿼리이다.

Xi는 도메인 변수를 의미하며 <x1, x2, …, xn>은 하나의 tuple로 결과 속성들을 나타낸다.



7장 – entity relationship model (E-R model)

**Modeling – Entity set**

E-R 모델이란 개념적 모델링에 사용되는 데이터 표현 방법으로 Entity(개체)와 entity간 관계를 나타내는 모델이다.

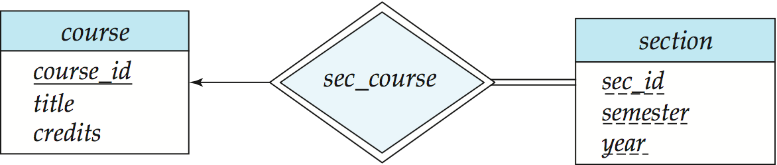
Entity: 실제로 존재하는 object(물체)이고 다른 물체와 구별이 되어야 한다. 현실세계의 요소 중 데이터베이스에서 표현하려는 것이고 주로 physical하거나 conceptually하게 존재한다. 주로, 명사로 특정한 인물, 특정 회사, 특정 식물 등이 이에 해당한다.

Entity에는 attribute가 있다. (예를 들어, 인물의 속성으로 이름과 주소 등이 있다.)

Entity set: entity의 집합으로 같은 성질을 가지고 있는 entity들을 의미한다. 예를 들어, 사람들의 집합, 회사들의 집합 등이 있다.

Weak Entity Sets

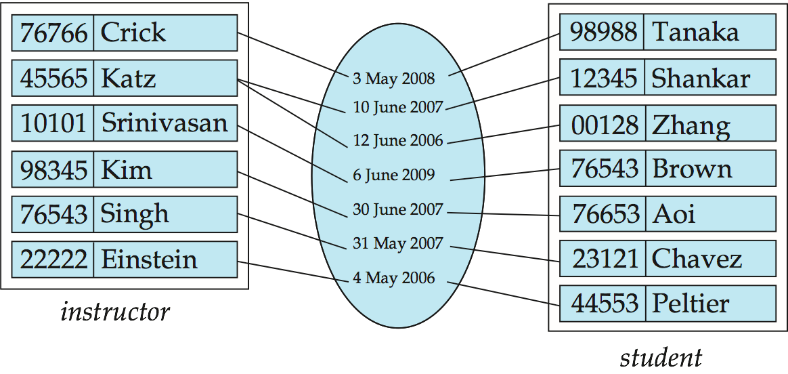
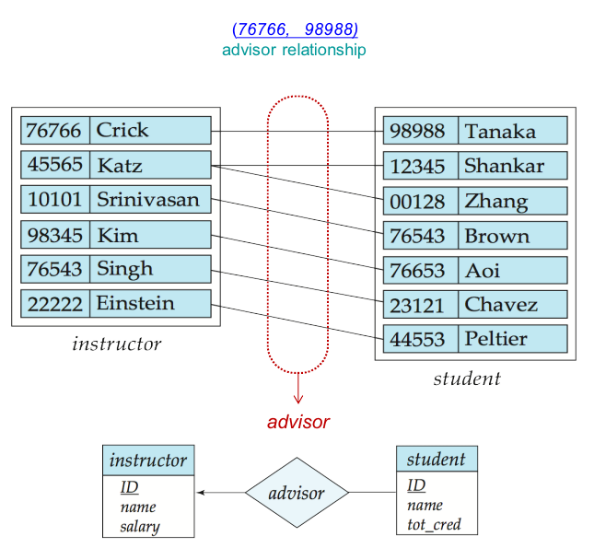
* primary key를 가지지 않는 entity set을 의미한다. Weak entity set은 identifying entity set이 존재해야 존재한다. Identifying relationship이란, 이 entity가 존재해야 weak entity가 존재하는 관계이다. 주로 identifying relationship은 다이아몬드 두 개로 이루어져 있다.
* Discriminator(partial key)는 weak entity set에서 다른 entity들을 서로 구별할 수 있게 해주는 attribute의 집합이다. 주로 discriminator는 \_ \_ \_로 표시한다. 예를 들어, section은 실제 열리는 수업이고 course는 수업번호라고 할 때, section은 course가 없이는 존재할 수 없으므로 section은 weak entity가 된다. Course는 identifying entity set이 되며, sec\_course는 identifying relationship(식별 관계)에 해당한다.
* Weak entity set의 primary key는 identifying entity set의 primary key와 weak entity set의 partial key로 이루어져 있다. Section의 primary key = (course\_id, sec\_id, semester, year)으로 relation schema로 변환 시 테이블은 section(course\_id, sec\_id, semester, year)이 된다.



**Relationship Set**

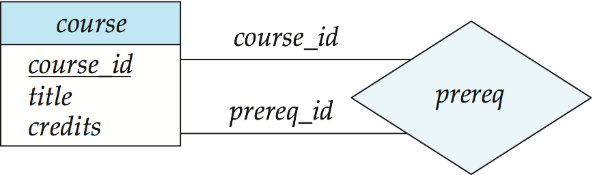
Relationship: entity간의 관계를 의미. 예를 들어 학생 entity와 강사 entity 사이에는 advisor라는 relationship이 존재한다.

Relationship set: 두 개 이상의 entity간의 수학적인 relation이다. 또한 고유 속성을 가질 수도 있다. (오른쪽 그림 경우 instructor과 student entity sets사이에 advisor relationship set이 date라는 고유 속성을 가진다.)

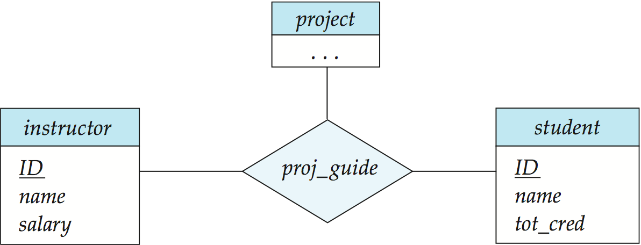


Degree of Relationship set: 관련된 개체의 개수에 따라 분류된다.

* Binary relationship: 두개의 entity set을 포함한다. 데이터베이스 시스템에서 대부분의 relationship set은 binary이다.
* Unary relationship: 하나의 entity set내부 attribute끼리 relationship이 있는 경우이다. 같은 entity set의 내부 속성끼리 관계가 있을 경우 역할(role)을 정해줘야 한다.



* Ternary/N-ary relationship: 여러 개의 entity set을 포함한다.



**Attributes**

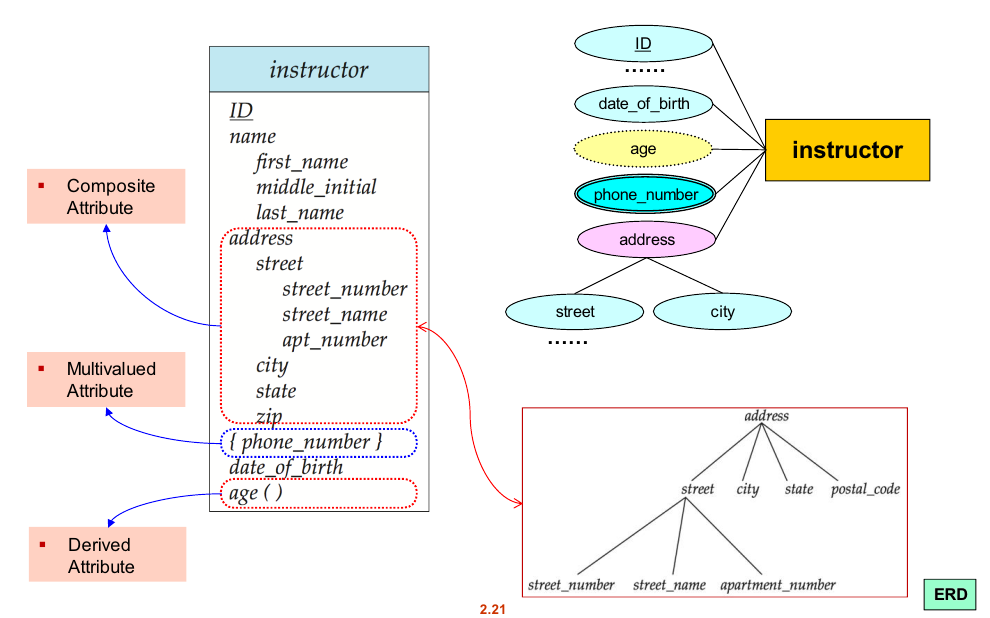
하나의 entity는 속성들의 set으로 표현된다. (attribute는 entity의 특징들을 나타낸다.)

Ex) instructor = (ID, name, street, city, salary)

Domain: 속성이 가질 수 있는 모든 값의 집합

속성의 종류

* Simple / composite attributes
* Single-valued / multivalued attributes (값의 개수)
* Derived attributes (다른 속성들에 의해 계산될 수 있다. 예를 들어, age경우 생년월일이 주어지면 구할 수 있다.)

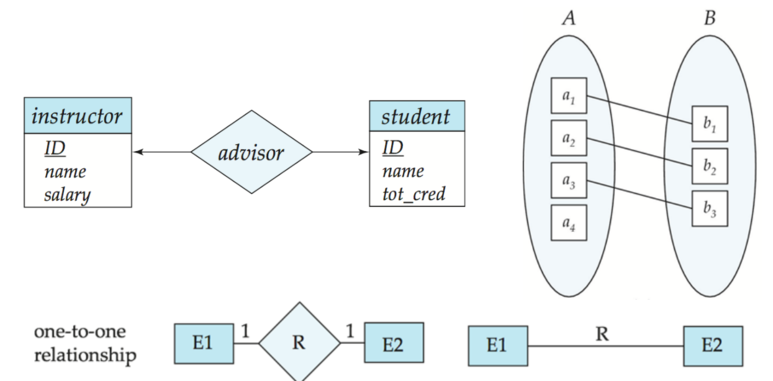


**Mapping Cardinality Constraints (mapping 관계유형)**

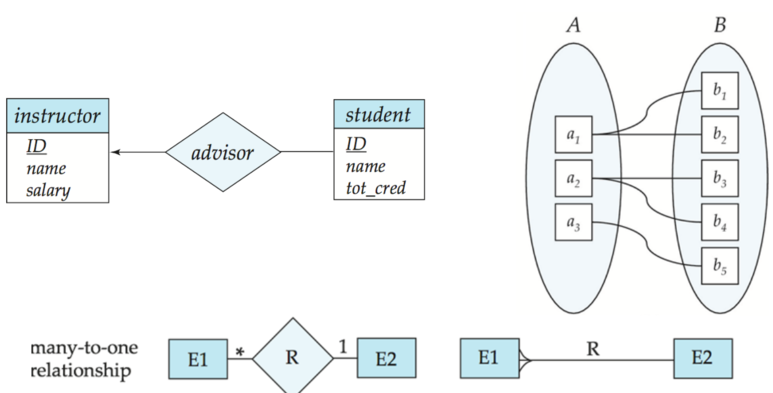
한 entity에서 하나의 relationship set으로 연결할 수 있는 다른 entity의 개수를 의미하며 binary relationship에서 주로 사용된다.

Binary relationship set에서 mapping cardinality는 4가지 종류가 있다. (각 entity set속 요소들은 연결되지 않을 수도 있다) one일 경우 →로 표시하며, many일 경우 ㅡ로 표시한다.

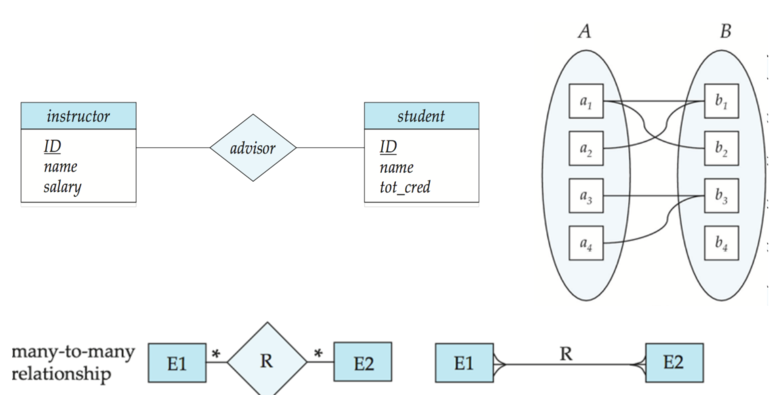
* One to one: relationship set에서 두 entity set으로 화살표 방향이 있다.



* One to many: 일대다 관계로 일을 나타내는 entity set에게 화살표가 있다.



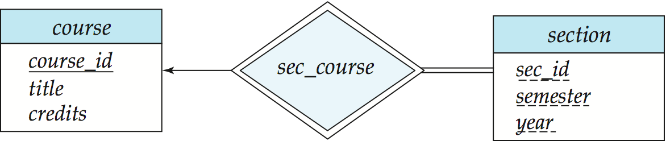
* Many to one: 다대일 관계로 one to many와 동일
* Many to many: 다대다 관계로 relationship set에서 두 entity set으로 화살표가 없다.



* Participation of an Entity Set in a Relationship Set: entity들의 relationship 참여를 의미하며 두가지가 존재한다.

Total participation: entity set에 있는 모든 entity들은 최소 한 개의 relationship에 참가해야 한다. E-R diagram 표시 시 =으로 entity set과 relationship set을 연결한다.

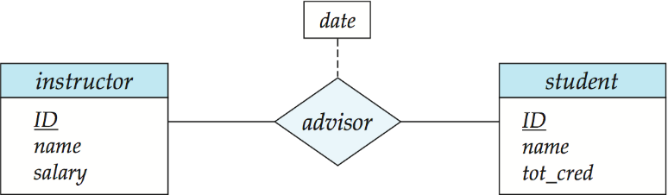
Partial participation: entity set에 있는 entity들 중 몇몇 entity는 관계에 참여하지 않아도 된다.



**Keys**

Super key, candidate key 그리고 primary key가 있다.

**E-R Diagrams**



Rectangle: entity set을 의미

Diamond: relationship set

Attribute: entity rectangle내부에 존재

Primary key: attributes 중 밑줄 그어진 속성

Relationship set에서 점선으로 이어진 rectangle은 relationship set의 고유 속성을 의미한다. (없을 수도 있다.)

**Reduction to Relation Schemas (relation schema로 변환)**

Entity set과 relationship set은 데이터베이스의 내용을 나타내는 relation schema로 표현할 수 있다.

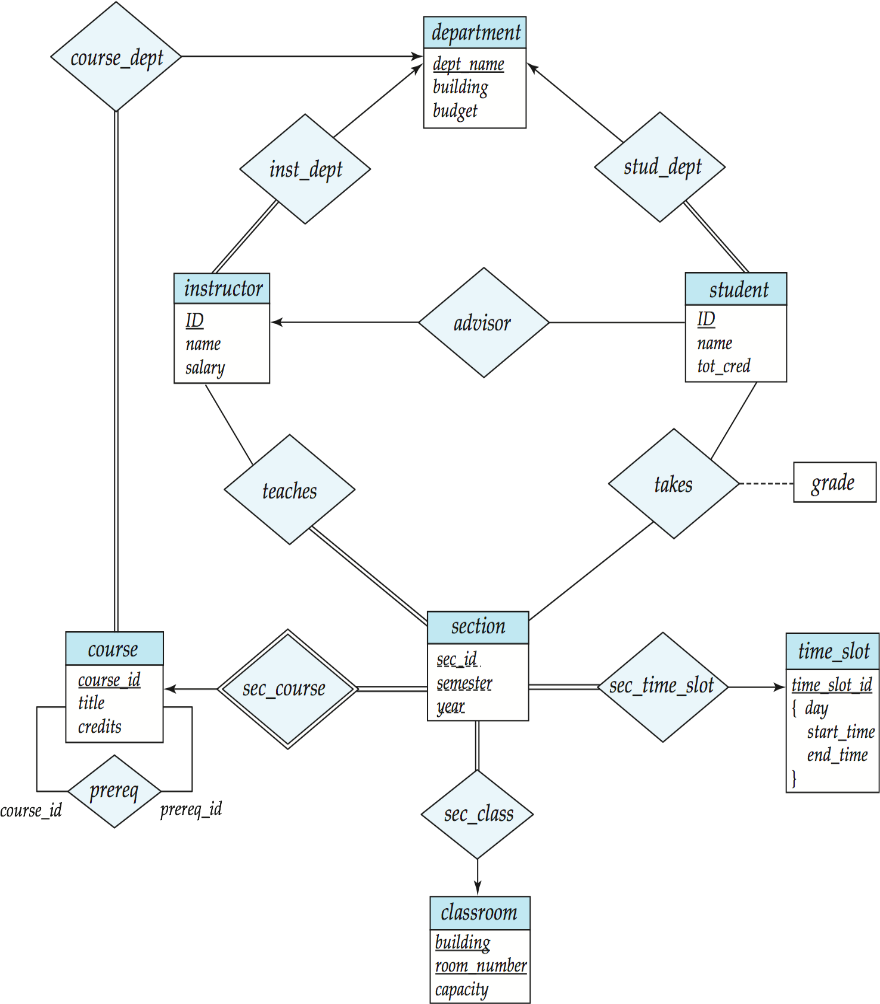
Entity set

Strong entity set은 변환 시 그대로 table이 형성되며, weak entity set은 partial key + 식별해주는 키(primary key of the identifying strong entity set)로 테이블이 형성된다.

Relationship set

One to many relationship: relationship set에 대한 schema를 따로 만들지 않는다. “one” side의 entity set은 변환 시 그대로 table을 형성하며, “many” side의 entity set은 존재하는 속성들에 “one” side의 primary key를 포함하여 table을 형성한다.

*🡨 이 그림에 대한 schema 형성*

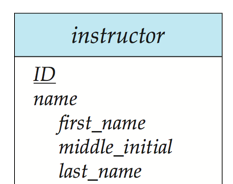


*instructor = (ID, name, salary, dept\_name)  
student = (ID, name, tot\_cred, advisor, dept)*

(속성 이름을 바꿔서 적어도 되며 student 속 advisor는 instructor.ID, dep는 department.dep\_name을 의미한다.)

Many to many relationship: relationship set에 대한 table을 만든다. 이때 many to many관계를 형성하는 두 entity set의 primary key를 가지고 테이블을 형성하며 이 2개의 속성이 이 테이블의 primary key가 된다.

One to one relationship: 두 entity set 중 한 곳에 한꺼번에 저장하면 된다. 즉 2개의 entity set에 대해 table이 한 개 형성된다. (두 entity set에 대해 테이블을 따로 만들 경우 정보를 찾기에 좀 더 좋지만 같은 정보가 2번씩 존재하고 한쪽의 정보를 지울 경우 다른 쪽 table의 매칭되는 정보를 지우지 않으면 일관성이 떨어지는 문제가 생긴다)

Composite Attributes

Composite 속성 경우 해당 attribute의 component attribute들에 대해 schema를 만든다.

Ex) instructor entity set에 composite 속성 name이 존재하고 name에는 component 속성 3가지(first, middle, last)가 있을 때, instructor의 schema는 instructor(ID, first\_name, middle\_name, last\_name, …)으로 표현된다.

Multivalued Attributes

Entity set E의 multivalued attribute M은 분리된 schema EM으로 표현된다. 즉 E의 schema와 분리

Schema EM은 E의 primary key와 M을 속성으로 가지며 EM의 primary key는 (E의 primary key, M)이다.

만약 entity set E의 속성이 primary key와 multivalue 두가지만 존재할 경우 따로 테이블 만들 필요 없이 E에 대한 테이블만 만들면 된다.

**Extended E-R Features**

Specialization(특수화): top-down design process, 세분화하며 super class에서 sub class로 내려가기

Generalizaion(일반화): bottom-up design process, 같은 특징을 공유하는 entity set들을 결합하며 올라가기

두개 모두 방향만 다르고 결과는 같다.

8장 – relational database design (정규화)

Normalization(정규화)란 관계형 데이터베이스의 설계에서 중복을 최소화하게 데이터를 구조화하는 프로세스를 말한다. 정규화의 목표는 이상이 있는 관계를 재구성하여 작고 잘 조직된 관계를 생성하는 것에 있다.

정규화의 목적 (주로 상위 2가지를 목적으로 한다)

* 불필요한 데이터(data redundancy)를 제거
* 각종 이상현상(Anomaly)을 방지하기 위해, 테이블의 구성을 논리적이고 직관적으로 한다.
* 데이터베이스 구조 확장 시 재디자인을 최소화
* 다양한 관점에서의 query를 지원하기 위해
* 무결성 제약조건의 시행을 간단하게 하기 위해

이상 현상은 갱신 이상(Modification Anomaly), 삽입 이상(Insertion Anomaly), 삭제 이상(deletion Anomaly)로 구성된다.

* 갱신 이상: 반복된 데이터 중에 일부를 갱신할 시 데이터의 불일치가 발생 가능
* 삽입 이상: 불필요한 정보를 함께 저장하지 않고는 어떤 정보를 저장하는 것이 불가능
* 삭제 이상: 필요한 정보를 함께 삭제하지 않고는 어떤 정보를 삭제하는 것이 불가능

정규화의 법칙은 1차정규화(1NF), 2차정규화(2NF), 3차정규화(3NF), BCNF, 4차정규화(4NF), 5차정규화(5NF)로 나눌 수 있으며, 실무적으로는 4차, 5차 정규화까지 가능 경우는 드물다.

용어설명: 함수적 종속성, 무손실 분해

함수적 종속성(Functional Dependency): 좋은 릴레이션 설계의 정형적 기준으로 사용되며 속성들의 의미와 속성들 간의 상호관계로부터 유도되는 제약조건의 일종이다. 함수적 종속성과 키는 릴레이션의 정규형을 정의하기 위해 사용된다.

* 정의: X와 Y를 임의의 속성 집합이라고 할 때, X의 값이 Y의 값을 유일하게 결정한다면 “X는 Y를 함수적으로 결정한다”고 한다.
* 표기: X 🡪 Y이며 “Y는 X에 함수적으로 종속된다.”를 의미한다. (x를 결정자, y를 종속자)
* Trivial: 종속성 중 당연한 성질(증명할 필요가 없다)을 뜻하며, ID, NAME 🡪 ID와 같은 경우 함수적 종속성이 trivial 하다고 한다. 일반적으로 b ⊆ a이면, a → b 은 Trivial이다. (a가 b를 결정하는데 b가 a의 subset)
* 완전 함수적 종속: 종속자가 기본키에만 종속되며, 기본키가 여러 속성으로 구성되어 있을 경우 기본키를 구성하는 모든 속성이 포함된 기본키의 부분집합에 종속된 경우
* 부분 함수적 종속: 종속자가 기본키가 아닌 다른 속성에 종속되거나, 기본키를 구성하는 속성 중 일부만 종속되는 경우
* 이행적 함수 종속: X, Y, Z라는 3개의 속성이 있을 때, X🡪Y, Y🡪Z란 종속 관계가 있을 경우 X🡪Z가 성립될 때를 의미함. 즉, X를 알면 Y를 알고 그를 통해 Z를 알 수 있는 경우
* 암스트롱의 추론 규칙: 재귀성 규칙(b ⊆ a이면, a → b이다.), 부가성 규칙(a→b이면, aγ→bγ이다.(aγ은 a∪γ 을 의미)), 이행성 규칙(a→b 이고, b→γ이면, a→γ 이다.)
* 함수적 종속성의 집합 F의 Closure: (F로부터 추론할 수 있는 모든 가능한 함수적 종속성들의 집합)
* F에서 속성 집합 X의 closure: (함수적 종속성 집합 F를 사용하여 α에 의해 함수적으로 결정되는 모든 속성의 집합)
* Canonical Cover: 함수적 종속성들 중 추론을 통해 얻어질 수 있는 함수 종속을 없앨 수 있다. (간략화)

무손실 분해(Lossy decomposition): 하나의 릴레이션을 분해하고 다시 조인연산(natural join)을 했을 때 데이터 손실이 없는 것을 의미한다. (분해 전과 분해 후 조인 결과 값을 비교 시 결과가 같다는 뜻)

Relation R가 좋은 형태가 아닌 경우 각각이 좋은 relation이고 무손실 분해를 만족하는 여러 relation으로 나눠야 한다.

정규화 법칙

1차정규형 (1NF, First Normal Form)

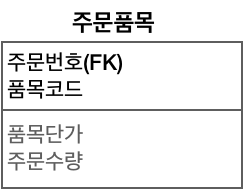
Domain의 값이 atomic해야한다.(atomic하지 않은 domain: set on names(집합), composite attribute) 만약 릴레이션 R의 모든 속성에 대한 도메인이 atomic하다면, 릴레이션 R은 1차 정규형을 만족한다.

atomic하지 않은 domain이 사용될 경우 질의어가 복잡해지고, 저장 장치가 복잡 그리고 데이터의 중복된 저장공간 사용을 유도한다.

2차정규형 (2NF, Second Normal Form)

1차정규형을 만족하면서 릴레이션에 존재하는 부분 함수적 종속을 제거하여, 모든 속성이 기본키에 완전 함수 종속이 되도록 만들어진 정규형이다.

부분적 함수 의존이란 기본 키 중에 특정 컬럼에만 종속된 컬럼이 존재함을 의미한다.

예를 들어, 다음과 같은 relation이 주어질 때 주문수량에 경우 이 relation의 primary key (주문번호, 품목코드)에 종속되어 둘 중하나 또는 두개가 모두 변경될 경우 수량 또한 바뀌게 된다. 하지만 품목단가 속성에 경우, primary key에 존재하는 속성 중 품목코드에만 종속한다. 즉, 주문번호가 변경되더라도 품목단가는 변경되지 않는다. 이를 부분적 함수 종속 관계라고 한다.

부분적 함수 종속이 발생하게 되면 데이터의 중복이 발생하게 된다.

해결방안: 부분적 함수 종속 관계를 형성하는 속성들에 대한 테이블을 따로 만든 뒤, 기존 테이블에서 부분적 함수 종속 관계를 형성하는 속성 중 primary key가 아닌 속성(원인이 되는 속성)을 제거한다. 즉 테이블을 쪼개어 제2정규화를 만족하는 relation들을 만든다.

위 예시 경우, 주문품목(주문번호, 품목코드, 주문수량), 품목(품목코드, 품목단가)로 나눈다.

3차정규형 (3NF, Third Normal Form)

2차 정규형을 만족하면서 릴레이션을 구성하는 속성들 간의 이행적 종속관계를 분해하여 속성들간 이행적 함수 종속이 없도록(기본키 이외의 다른 컬럼이 그외 다른 컬럼을 결정할 수 없는 것) 만들어진 정규형이다.

이행적 함수 종속이 발생할 경우 데이터 중복이 생기게 된다. 데이터 중복을 줄이기 위해 제2정규화와 마찬가지로 테이블을 나눈다.

해결법: 원인이 되는 속성을 제거하여 다른 엔티티의 속성으로 뺀다. 즉, 기본 테이블에서 A(primary key), B, C, D가 존재하고 B🡪C라는 이행적 함수 종속이 존재할 경우, 테이블을 (A,B,D)와 (B,C)로 나눈다.

BCNF (Boyce-Codd Normal Form)

3차 정규형을 만족하면서 모든 결정자가 super key인 경우(a🡪b, a 는 R의 슈퍼키) 해당 relation은 BCNF를 만족한다. 이 또한 BCNF를 만족하지 않은 경우 table을 나눔으로써 해결한다.

3차 정규형을 만족하면서 BCNF를 만족하지 못하는 경우: 일반 컬럼이 후보키를 결정하는 경우

해결방안: BCNF에 위반한 dependency a🡪b에 대해 두 테이블로 나눈다. (a∪b와 R-(b-a))

4차정규화 (4NF, fourth normal form)

BCNF를 만족하면서 릴레이션에서 다치 종속 관계를 제거한 정규형이다.

다치 종속(multival dependency): 두 속성이 1대다 대응이 되는 경우이다. 예를 들어, ‘학번’과 ‘수강과목’ 속성이 있을 때 한 학생이 여러 과목을 수강할 수 있는 경우 하나의 ‘학번’은 여러 개의 ‘수강과목’에 대응하므로 다치 종속 관계이다.

5차정규화

후보키를 통하지 않은 조인종속(Join Dependency)을 제거한 정규형이다. (잘 모름)

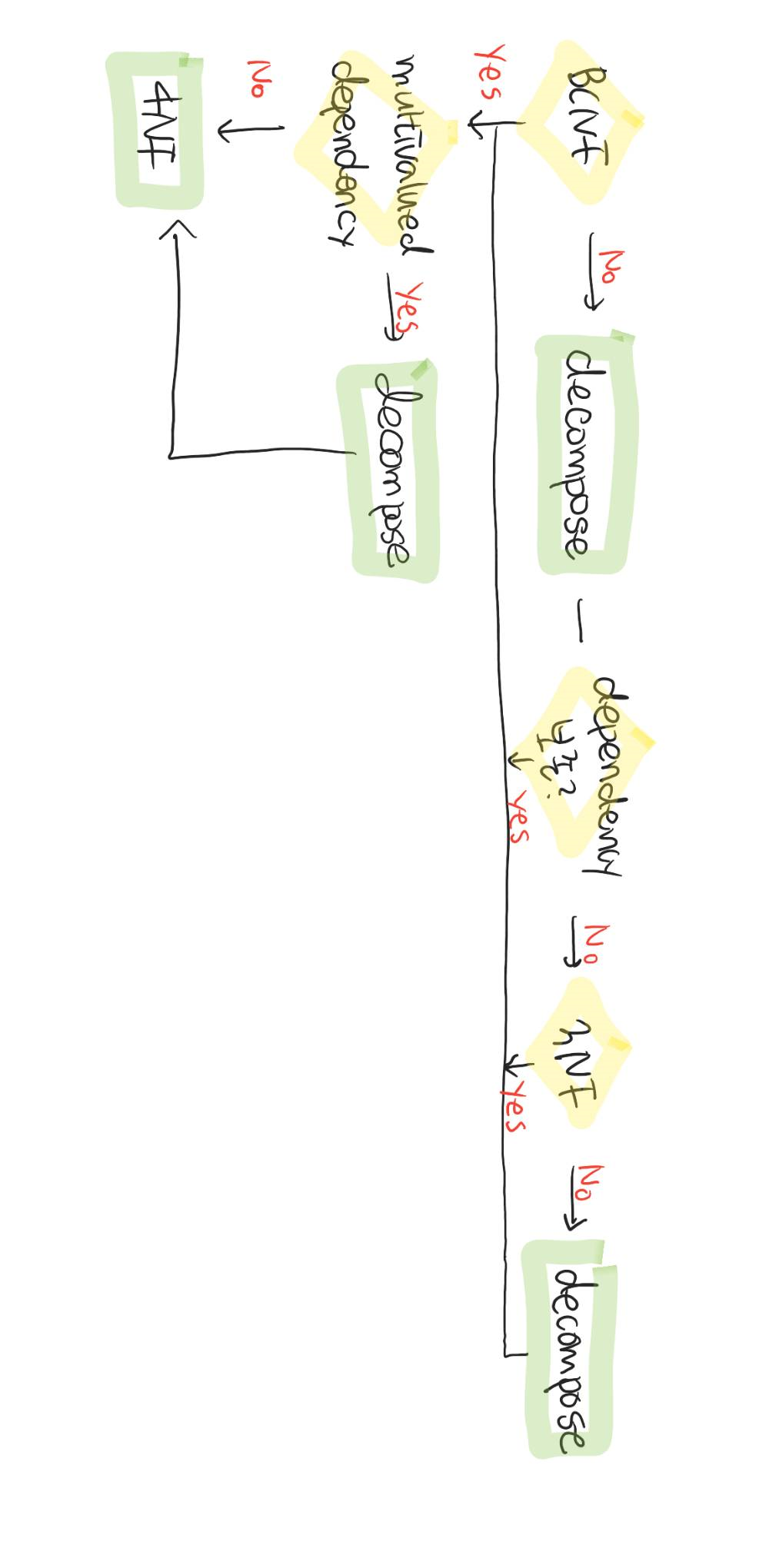
**3NF와 BCNF의 비교**

3NF는 무손실 조인 분해와 종속 보존 분해가 가능하지만 BCNF 경우 무손실 조인분해는 가능하나 종속 보존 분해가 불가능한 경우가 존재한다.

예를 들어, R(A, B, C, D) dependency: C🡪D, AD🡪B가 있을 때 C는 슈퍼키가 아니므로 BCNF를 만족하지 않는다. 따라서 분리시켜야 하며 R(A,B,C), T(C,D)로 분리된다. 이때, 나눠진 테이블은 AD🡪B를 만족하기 위해서는 두 테이블을 합쳐야 한다. 즉 나눠진 테이블에 대해서 dependency가 만족하지 않아 join할 상황이 생기게 된다.

**Multivalued Dependencies**: multivalued 속성은 한 table 씩 1개 존재하게 해야 한다.

정규화 과정



**Overall Database Design Process**

Schema R은 크게 3가지로 형성된다. 1. E-R diagram을 통해 테이블 R 생성. 2. 관계된 모든 속성을 가진 하나의 table 생성(Universal relation) 3. Ad hoc design (주먹구구식)

이후 생성된 테이블 R을 정규화 과정을 수행한다.

**Denormalization (역정규화)**

정규화를 통해 분리되었던 릴레이션에서 중복을 허용하고 다시 통합하거나 분할하여 구조를 재조정하는 과정이다. 데이터베이스의 물리적 설계 과정에서 성능을 향상시키기 위해 역정규화를 실행한다.

역정규화 시 찾을 때시간을 아끼는 장점과 업데이트시 추가 공간과 시간이 필요하고 추가적인 코딩 작업과 추가적인 코드에 대한 에러의 가능성이 존재할 수 있다는 단점이 있다.

뷰를 사용할 경우 코딩작업에 관한 단점이 사라진다. 나머지 장단점은 유지

10장 – 저장공간과 파일 구조 (Storage and file structure)

Physical Storage Media의 분류: 데이터 접근 속도, data 단위당 cost, 신뢰성(데이터 손실 유무, 고장), 휘발성과 비휘발성

**Physical Storage Media**

Cache: cpu속에 존재하며 가장 빠르고 가장 비싼 storage이다. 휘발성이고 컴퓨터 시스템 하드웨어에 의해 관리된다.

Main memory: cpu가 접근하는 저장공간이며 cache다음으로 빠르다. 휘발성

Flash memory: 비휘발성 메모리로 정보의 입출력이 자유로워 디지털 카메라, 핸드폰과 같은 임베디드 장치에 많이 사용된다. 단, 메모리에 data를 지우고 쓰는 것에 제한된 횟수가 존재. 메인 메모리처럼 읽기는 빠르지만 지우고 쓰는 것은 속도가 느리다.

* NOR flash와 NAND flash가 존재
* Wear leveling: 사용할수록 닳기 때문에 골고루 사용하도록 하여 더 오래 쓸 수 있도록 함.

Magnetic-disk (HDD): cpu가 접근하려면 disk에서 main memory로 정보를 이동시켜야 한다.

* 수행시간 측정 Access time = seek time(정확한 track위치 찾는 시간) + rotational latency(sector에 접근하는 시간) + data transfer time(disk에 데이터 저장 or 전송되는 시간)
* 데이터는 Block(한 track에서 인접한 연속된 sector)단위로 이동되며, block의 최소 크기는 512 bytes로 크기가 작으면 이동횟수가 많아지고 크기가 크면 공간의 낭비가 커진다.
* Disk-arm-scheduling: track에 대기중인 엑세스 순서를 지정해서 disk arm의 이동을 최소화한다.
* File organization: 데이터 접근 방법에 따라 block을 조직함으로써 block 접근 시간을 최적화한다.

Optical Storage(광학): 비휘발성 메모리이고 디스크에서 데이터를 읽을 때 레이저를 사용한다. Magnetic disk보다 읽고 쓰는 시간이 느리며 CD-ROM, DVD가 존재.

Tape Storage: 비휘발성 메모리이며 백업용이나 보관용으로 사용한다. 디스크보다 속도가 느리며 데이터를 읽을 때 차례대로 읽는다.

위에서 아래로 저장 가능한 데이터 양이 많아진다.

**Storage 계층**

Primary storage: 주기억장치 (cache, main memory) 빠르고 휘발성

Second storage: 보조기억장치 (flash memory, magnetic disk) 그 다음으로 빠르고 비휘발성

Tertiary storage: 백업용 기억장치 (magnetic tape, optical storage) 느리고 비휘발성

**RAID (Redundant Arrays of Independent Disks)**

여러 디스크를 병렬적으로 사용함으로써 더 높은 공간과 속도를 가진다. 또한, 데이터를 중복 저장함으로써 신뢰성을 높인다. (정보 손실을 막음)

Mirroring: 디스크 내용을 복제하여 같은 내용이 2개의 디스크에 존재한다.

Bit-level striping: 디스크 병렬 방식 중 하나로, 한 bit씩 저장 디스크를 나눈다. 읽는 시간은 빨라지지만 접근시간이 느려진다.

Block-level striping: block단위로 저장 디스크를 나눈다.

LEVEL

RAID 0: block striping, 복제 없음 / 데이터 손실 확률이 높아진다.

RAID 1: block striping, mirrored disks / n개의 디스크에 대해 n개만큼 내용 복제한 디스크 존재

RAID 2: bit striping, ECC(에러 감지하는 코드) / ECC는 데이터 통신에서 주로 쓰인다.

RAID 3: bit-interleaved Parity(parity bit를 사용하는 방식 🡪 데이터 손실 시 이를 이용해 복구)

RAID 4: block-interleaved parity, block-level striping / parity 저장되는 특정 disk 존재(내용 수정이나 입력 시 항상 계산해야 하므로 병목현상이 일어날 수 있다. 성능 떨어짐) 🡪 이는 3도 동일

RAID 5: block-interleaved distributed parity / parity를 모든 disk에 분배해서 저장한다. 병목현상을 피할 수 있다.

RAID 6: P+Q redundancy scheme / 5와 유사하며 데이터 복제도 같이 한다. Parity가 2배 늘어남 (별로 사용 안한다. 비용이 비쌈)

대부분을 1 또는 5 중 선택하며 성능은 1이 더 좋고 용량은 5가 더 좋다. 1을 가장 많이 씀

**Hardware Issues**

Latent failures(잠재적 실패): data 손실을 데이터 읽을 때 발견한다.

Data scrubbing: 평소 data를 scan해서 잘못된 data를 점검한다. (latent failures 문제에 대한 해결)

Hot swapping: 시스템 작동 중에 전원 끄는 것 없이 disk를 교체한다. (몇몇 RAID system에서 제공하며 회복시간의 감소와 이용가능성 향상의 장점이 있다)

**File Organization**

파일이 디스크에 저장되어 있을 때 레코드를 파일에 배치하는 방법이다.

Fixed-Length records: record의 길이를 고정한다. 각 record의 시작 위치를 쉽게 알 수 있다.

고정된 길이의 레코드 삭제 시 배치 방법

* Compacting: 레코드 제거 후 뒤에 레코드들을 앞으로 이동시킨다 (data 많을 때 오래 걸린다.)
* Moving last record: 제거된 레코드 위치에 마지막 레코드를 이동시킨다. (순서가 정렬된 경우에 좋지 않다)
* Free lists: 삭제된 record들 \*\*로 표시하고 비워둔다. Linked list로 빈 레코드를 연결해 재사용(디스크의 용량낭비가 있을 수 있다.)

Variable-Length records: 가변 길이 레코드 방식으로 고정길이보다 디스크 공간낭비가 적다. 하지만 전체길이가 변하므로 각 레코드의 시작부분을 알 수 없다. 데이터 저장 시 고정된 길이에 해당 레코드의 속성(offset, length)을 저장하고 그 이후 가변길이 부분에 실제 데이터를 저장한다.

Slotted page structure: 가변 길이 레코드 방식에서는 slotted page를 사용하며, 이 페이지의 header에는 전체 레코드 개수와 각 레코드의 위치와 크기가 포함된다.

**Organization of Records in Files (레코드 저장 법)**

Heap: 공간 있으면 아무 곳에나 저장

Sequential: 연속적 순서로 기록 (전체 file에 대한 연속적인 처리가 필요한 경우 좋음), search-key에 의해 정렬됨, pointer를이용해 레코드 연결하며 공간 없을 경우만 다른 곳에 저장

Hashing: hash함수 이용해 저장위치 결정

Multitable clustering file organization: 하나의 파일에 여러 relation들을 같이 저장한다. join연산시에는 좋으며 single table 처리에는 좋지 않다. 하지만 pointer로 특정 relation의 레코드들을 연결하면 single table 처리시 좀 더 쉽게 할 수 있다.

**Data Dictionary Storage**

Data dictionary = system catalog는 metadata를 저장한다.

Metadata: 테이블에 대한 정보(이름, 속성정보, view, 무결성 제약조건), 사용자, 계정, 권한, 패스워드, 통계자료(최적화시 사용됨), file 구조 를 저장하며 metadata도 table형식으로 저장된다.

**Storage Access**

Disk에 있는 file을 처리하기 위해서는 main memory에 정보가 존재해야 한다. Memory속 buffer에 block단위로 이동됨. Memory 공간이 충분할 경우 디스크 접근 수를 최소화 하기위해 main memory 속 buffer에 모든 db내용이 존재한다.

Buffer 교체 정책 (관리)

* LRU(latest recently used): 대부분의 OS가 사용하는 전략이다. 테이블 두개를 join한 경우처럼 데이터를 반복적으로 scan해야하는 경우 좋지 않는 전략이다.
* MRU(most recently used): join과 같은 작업 시 MRU가 더 좋다.
* Pinned block: disk로 돌려보내지말고 buffer에 계속 고정
* Toss-immediate storage: block을 사용한 후 즉시 공간을 free한다.
* Forced output: recovery의 목적으로 강제로 buffer를 disk로 이동하는 것 (다시 쓴 겨우 disk에 내용 알려줘야한다. 이용안하고 계속 buffer에 있을 시 잘못되면 recovery 불가능)
* 즉, 수행하는 작업을 잘 고려해서(통계정보, 확률) buffer를 관리해야 한다.