데이터베이스

**파일시스템**

1. 파일(데이터의 모임)을 저장 장치에 저장하고 사용하기 일종의 규칙이나 체계
2. 파일과 관련된 기능을 수행하는 시스템
3. 데이터 정의가 응용프로그램에 내포되어 있다. 따라서 응용 프로그램과 데이터 간의 의존관계가 존재
4. 따라서 **데이터 구조, 접근 방법이 변경되면 기존의 프로그램과 데이터를 함께 변경해야 한다**.
5. 이로 인해 많은 단점들이 발생 🡺 DBMS의 장점을 반대로 하면 됨

**Database**

1. 여러 사람에 의해 공유되어 사용될 목적으로 통합하여 관리되는 데이터들의 집합
2. 자료항목의 중복을 없애고 자료를 구조화해서 저장하여 **자료 검색과 갱신의 효율성을 높임**
3. 특징
   * **실시간 접근성**: 실시간 처리에 의한 응답이 가능해야 한다.
   * **계속적인 변화**: 데이터의 삽입, 삭제, 갱신으로 항상 최신의 데이터를 유지
   * **동시공용**: 다수의 사용자가 동시에 같은 내용의 데이터 이용 가능해야 한다.
   * **내용에 의한 참조**: 데이터 레코드의 주소나 위치에 의한 참조가 아닌, 사용자가 요구하는 데이터 내용으로 데이터를 찾는다.

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

1. 데이터베이스를 관리하며 응용프로그램들이 데이터베이스를 공유하여 사용할 수 있는 환경을 제공하는 소프트웨어
2. 종류: Oracle, MySQL 등
3. 장점
   * **데이터 중복의 최소화**: 응용 프로그램마다 자신의 파일을 개별적으로 유지 관리하여 데이터 중복 발생. 데이터를 통합하여 구성함으로써 중복을 사전에 통제
   * **데이터의 일관성 유지**: 중복된 데이터 존재할 때 한가지만 변경된다면 데이터의 불일치 발생. DBMS는 데이터의 중복을 제어하고 데이터의 일관성을 유지할 수 있다.
   * **데이터의 무결성 유지**: 데이터베이스가 생성 조작될 때마다 제어 기능을 통해 유효성을 검사함으로써 데이터의 무결성을 유지(허용되지 않는 값, 부정확한 데이터 관리)
   * **데이터의 공용**: 다수 사용자를 위한 동시성 제어가 제공된다. (충돌 제어 가능)
   * **데이터의 보안 보장**: 데이터베이스를 중앙 집중식으로 총괄함으로써 데이터베이스의 관리 및 접근을 효율적으로 통제 가능.
4. Database system의 구조 (추상화 level)
   * Physical level: 디스크에 저장하는 방법을 묘사
   * Logical level(Conceptual level): DB에 저장된 데이터와 데이터간 관계를 묘사 (table 구조) 🡸 우리가 실질적으로 작성하는 단계
   * View level: 각 사용자에게 DB를 보여주는 단계로 보안 목적에 의해 전체 DB중 일부분만을 사용자에게 보여줌
   * **Physical data independence**: 프로그램은 logical schema에 의존하므로 physical schema(저장위치 or 파티션)이 변경에 영향을 받지 않는다. (level의 구분 때문)

**Data Language**

1. **DDL** (Data Definition Language): 데이터 정의어
   * DB를 정의하는 언어로 데이터를 생성, 수정, 삭제하는 등의 **데이터의 전체 골격을 결정**하는 역할을 한다. 데이터 베이스 관리자나 설계자가 사용
2. **DML** (Data Manipulation Language): 데이터 조작어
   * 정의된 데이터베이스에 입력된 레코드를 조회, 수정 또는 삭제하는 등의 역할 하는 언어로 사용자가 응용프로그램이나 질의어를 통해 **실질적으로 데이터를 처리하는데 사용됨**
   * Procedural(절차적언어, C++, Java가 포함됨)와 Declarative(선언적, SQL)유형이 존재
3. **DCL** (Data Control Language): 데이터 제어어
   * **데이터베이스에 접근하거나 객체에 권한을 주는 등의 역할**을 하는 언어
   * 데이터의 보안, 무결성, 회복, 병행 수행제어 등을 정의하는데 사용

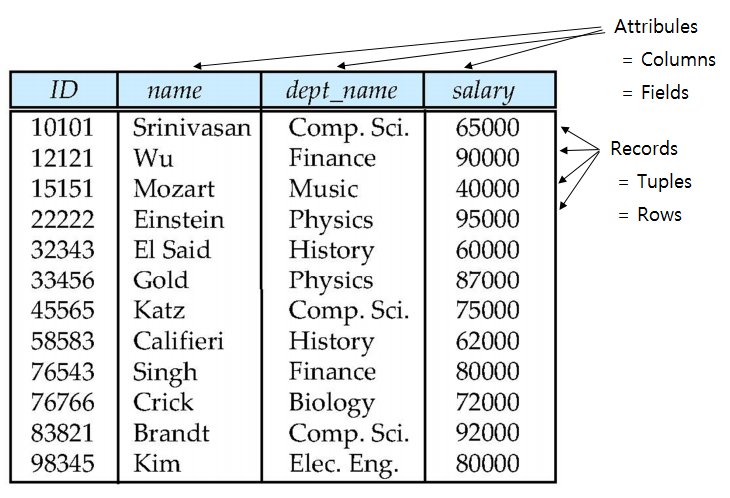
**Data Model(데이터 모델)**

1) 데이터를 저장하고 접근하는 방식을 개념적으로 정의한 것

1. **개념적 데이터 모델**
   * 속성들로 기술된 개체 타입과 개체 타입들 간의 관계를 이용하여 현실 세계를 표현
   * 대표적인 개념 데이터 모델: **E-R model**
2. **논리적 데이터 모델**
   * 개념적 모델링 과정에서 얻은 개념적 구조를 컴퓨터가 이해하고 처리할 수 있는 컴퓨터 세계의 환경에 맞도록 변환하는 과정
   * 데이터 간의 관계표현 방식에 따른 종류: **Relational model**, 계층 모델, 네트워크 모델

**Relational Model**

1. 개체 속성, 개체간 관계에 관한 정보를 **table 형식으로 저장**
2. Example of a Relation (instructor table)



1. Attribute Types
   * 각 속성은 domain이라고 하는 허가된 값의 집합을 가진다. (변수의 값의 범위, 글자 수 제한 등)
   * 속성 값들은 **atomic(원자성)**이 요구됨. (각 cell에 값이 하나씩 존재)
   * Null은 모든 도메인에 포함되고 unkown또는 not applicable의 의미를 가진다.
2. Relation Schema
   * 테이블의 청사진으로 relation의 이름과 속성들로 표현된다.
   * 위 그림 예제의 relation schema는 instructor = (ID, name, dep\_name, salary)
3. Relation Instance
   * 도메인 D1, D2, …, Dn이 있을 때, 하나의 relation r은 D1xD2x…Dn 조합의 subset으로 이루어져 있다.
   * A relation is a set of n-tuples (a1, a2, …, an) where each ai ∈ Di
   * Tuple: 요소의 순서에 의미가 있고 데이터 중복 가능
   * Set(집합): 요소의 순서에 의미가 없고 데이터 중복 불가능
   * **relation에 속 tuple들의 순서는 의미가 없고 tuple의 속성들 순서는 의미 있음**
4. **Key: 중복없이 데이터를 다른 데이터들과 구별해주는 식별자**
   * **Super key(슈퍼키)**: 중복없이 데이터를 다른 데이터들과 구별해줄 수 있는 모든 가능한 속성 정보들의 집합
   * **Candidate kay(후보키)**: superkey 중 조합 개수가 가장 작은 조합들의 집합
   * **Primary key(기본키)**: 실제 table에서 사용되는 key를 의미하며 후보키 중 하나를 선택하여 사용한다. 기본키가 되면 null값을 가질 수 없다.
   * Super key 🡪 candidate key 🡪 primary key 순서로 key를 찾음
   * **Foreign key(외래키)**: 다른 relation의 primary key를 참조하는 속성 (나에게는 외래키, 참조 대상에게는 primary key)

**관계 데이터 연산 – 관계 대수와 관계 해석**

1. **Relational algebra(관계 대수)**: 절차식 언어로 데이터를 얻기까지 처리과정과 방법을 표현
2. 기본 연산
   * **select(σ)**: sql에서 where절과 동일

*σ* *p*(*r*) = {t| t ∈ r and p(t)} : 해당 조건(p)에 맞는 tuple들을 결과로 출력

* + **project(П)**: 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

* + **set difference(-)**

*r – s* = {*t* | *t* ∈ *r* **and** t ∉ *s*}: 차집합을 결과로 출력

* + **Cartesian product(x)**

*r* x *s* = {*t q* | *t* ∈ *r* **and** *q* ∈ *s*}: 두 relation의 모든 조합을 결과로 출력

* + **rename(ρ)**

 E 테이블의 이름을 x로 바꾼다는 것으로 x(a1, a2, …, an)으로 왼쪽 수식과 같이 표현할 경우 해당 테이블의 속성들의 이름도 변경 가능

1. 추가 연산 (기본 연산자들을 이용해 표시 가능)
   * **set intersection(∩)**

*r* ∩ *s* = { *t* | *t* ∈ *r* **and** *t* ∈ *s* }: 두 relation의 교집합을 결과로 출력

*r* ∩ *s* = *r* – (*r* – *s*)

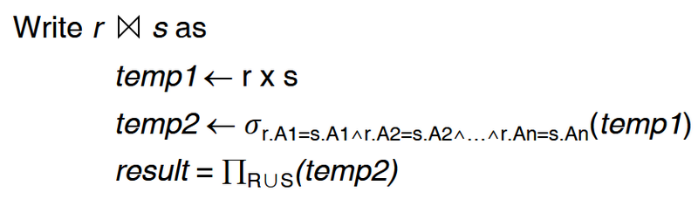
* + **Natural join**

: 두 relation의 공통 속성들의 값이 같은 tuple만 매칭해 결과로 출력

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

* + **Assignment**

연산(🡨)은 복잡한 쿼리문을 표현하기 위한 편리한 방법을 제공하며, SQL에서 with문과 같이 임시 table만드는 것



* + **Outer join (밑에 SQL설명에 설명 있음)**

정보의 손실을 피하기 위한 natural join연산의 확장 버전으로 null을 이용

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

* + **Division**

연산(÷) 사용. r ÷ s는 r table에서 s를 모두 포함하는 row들을 출력하는 것이다.



예) SC/C2: 학과목에서 과목2에 존재하는 모든 과목을 들은 사람을 결과로 표시

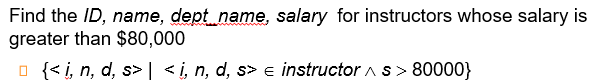


1. **Relational calculus(관계 해석)**: 선언적 언어로 어떤 데이터를 원하는지 선언한다.
   1. **Tuple 관계해석**: {t | P(t) }와 같은 형태를 갖는 query

t는 tuple을 의미하며 조건 P에 대해 참인 모든 tuple들의 집합을 의미

* 1. **Domain 관계 해석**: 이며 { < *x*1*, x*2*, …, xn* > | *P* (*x*1, *x*2*, …, xn*)} 와 같은 형태를 갖는 query

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



**SQL**

관계 대수와 관계 해석에 기초한 **데이터베이스 언어**

|  |  |  |
| --- | --- | --- |
| **DDL** | 테이블 생성, 변경, 삭제 | CREATE, ALTER, DROP |
| **DML** | 데이터 선택, 삽입, 변경, 삭제 | SELECT, INSERT, UPDATE, DELET |
| **DCL** | 권한부여, 권한해제 | GRANT, REVOKE |

1. **테이블 생성 CREATE**

**create table** *instructor* (  
 *ID* **char**(5),  
 *name* **varchar**(20) **not null, 🡸 null값 삽입 시 오류 발생***dept\_name* **varchar**(20),  
 *salary* **numeric**(8,2),  
 **primary key** (*ID*),  
 **foreign key** *(dept\_name*) **references** *department)*

* + 생성 시 무결성 제약조건:

Not null, primary key, foreign key (Am, …, An) references r

1. **테이블 삭제 DROP**

* Drop table 테이블명 [cascade constraint];
* cascade constraint 옵션: 테이블과 관계 있는 참조되는 제약조건에 대해서도 삭제

1. **테이블 변경 ALTER**
   * 컬럼을 추가/삭제 또는 제약조건을 추가/삭제하는 작업
   * 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
2. **데이터 선택 SELECT**

**[With 절]**

**Select [distinct]** A1, A2, …, An

**From** r1, r2, …, rm

**Where** P

**[Order by** A1, A2, … [desc | asc]] 🡨 asc가 기본 정렬방식

**[group by** Am]

[**having** P2(그룹의 조건)]

* + - Select절
      * 결과로 표시할 속성들 선택, 모든 속성 표시할 경우 \*로 대체
      * SQL은 relation에서 중복을 허용하므로 중복 제거하고 싶으면 Distinct추가
    - From절
      * 데이터 선택할 relation 작성, 여러 relation작성시 join된다.
      * From A, B 작성 시 Cartesian Product 형태 출력
      * From A **natural join** B 작성 시 A,B에서 같은 이름 가진 속성들의 같은 값 가진 tuple끼리 매칭
    - Outer Join: 정보 손실을 방지하기 위한 natural join연산의 확장된 연산
      * 서로 매칭 안되는 정보 가지고 있을 때 null을 이용해 그 값을 채운다.
      * Natural Left Outer Join: 왼쪽 table정보 모두 적기
      * Natural Right Outer Join: 오른쪽 table정보 모두 적기
      * Natural Full outer join: 양쪽 table정보 모두 적기
    - Where절: 조건문으로 식들을 and, or, not을 이용해 연결
    - Order by: 결과 relation의 tuple들을 특정 속성(속성들)에 대해 정렬
    - Group by: 기본 SQL문에 의해 나타난 결과 table을 특정 속성에 대해 그룹별로 결과를 표시한다.
    - Having Clause: 그룹의 조건으로 group by밑에 작성
    - Rename Operation(별칭 만들기): relation과 속성에 대해 별칭을 만들 수 있으며 한 query문에 같은 relation을 여러 번 사용하게 될 경우 별칭으로 구분할 수 있다.

🡺Old-name **as** new-name

* Aggregate Functions (집단 함수): 특정 속성값에 해당하는 column값을 모두 해당 함수에 대해 계산해서 하나의 결과값 출력
  + - Avg, min, max, sum, count 함수가 존재
    - Count(\*)을 제외한 모든 Aggregate 연산은 null값을 무시한다. 단, non-null값이 존재하지 않을 경우 결과에 null을 출력 (count(표현식)경우 non-null 없으면 0출력)

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

Ex) Select count(\*): tuple의 개수를 결과로 출력

* + - With절
      * 임시테이블을 만든 것을 view와 쓰임새가 비슷하지만 한번 만들면 drop할 때까지 사용가능한 view와 달리 with절은 한번 실행할 query문 내에 정의되며 해당 쿼리문에서만 실행된다. (이후 재사용 불가)

1. **데이터 삭제** (table 속 tuple을 삭제)
   * Delete from r [where ~]
   * 모든 tuple을 삭제해도 table은 유지됨 (drop과 구분)
2. **데이터 삽입** (새로운 tuple 삽입)
   * Insert into r values (속성1, … , 속성n)
3. **데이터 변경** (속성값 변경)

Update r set 변경할 속성 값 where 조건

1. **권한 부여 Grant**

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

1. **권한 회수 Revoke**

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

1. **View**: 사용자에게 접근이 허용된 자료만을 제한적으로 보여주기 위해 하나 이상의 기본 테이블로부터 유도된, 이름을 가지는 가상 테이블

[표현] CREATE VIEW v as <query expression> (v: view name)

* + 저장장치 내에 물리적으로 존재하지 않음.
  + 정의한 식만 저장되며 data는 복사되지 않는다. 따라서 사용자가 query문 작성시 view를 사용할 경우 view에 저장된 식으로 대체되어 수행됨
  + 여러 번 사용되는 질의를 view로 만들면 사용시 편리하다.
  + 기본 테이블의 기본키를 포함한 속성 집합으로 뷰를 구성해야 삽입, 삭제, 갱신 연산이 가능
  + 단점
    - 독립적인 인덱스를 가질 수 없다
    - 뷰의 정의를 변경할 수 없고 뷰로 구성된 내용에 대한 삽입, 삭제, 갱신, 연산에 제약이 따른다.

**Transaction(트랜잭션)**

1. 데이터베이스의 상태를 변환시키는 **하나의 논리적 기능을 수행하기 위한 작업의 단위** 또는 한꺼번에 모두 수행되어야 할 일련의 연산들을 의미
2. 특징
   * 데이터베이스 시스템에서 병행 제어 및 회복 작업 시 처리되는 작업의 논리적 단위
   * 사용자가 시스템에 대한 서비스 요구 시 시스템이 응답하기 위한 상태변환 과정의 작업 단위
   * 하나의 트랜잭션은 원자성을 가진다 (commit되거나 Rollback된다)
3. 성질
4. Atomicity(원자성): 트랜잭션 내의 모든 명령은 반드시 완벽히 수행되어야 하며, 어느 부분 하나라도 오류가 발생하면 한 트랜잭션 내의 모든 명령이 전부 취소되어야 한다.
5. Consistency(일관성): 시스템이 가지고 있는 고정요소는 트랜잭션 수행전과 완료 후의 상태가 같아야 한다. 즉 실행이 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 변환한다.
6. 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.
7. Durability(영속성, 지속성): 성공적으로 완료된 트랜잭션의 결과는 시스템이 고장나도 영구적으로 반영되야 한다.
8. Commit 연산: 트랜잭션에 의한 갱신연산이 완료된 것을 관리자에게 알려주는 연산
9. Rollback 연산: 트랜잭션이 비정상적으로 종료되어 DB의 일관성을 깨뜨렸을 때, 트랜잭션의 원자성을 위해 모든 연산을 취소(Undo)하는 연산. 트랜잭션을 재시작 or 폐기

**Integrity constraints(무결성 제약조건)**

1. 무결성: 데이터베이스에 저장된 데이터 값과 그것이 표현하는 현실 세계의 실제 값이 일치하는 정확성을 의미
2. **무결성 제약조건**: 데이터베이스에 들어있는 데이터의 정확성과 일관성을 보장하기 위해 부정확한 자료가 데이터베이스 내에 저장되는 것을 방지하기 위한 제약 조건
3. 무결성의 종류: 개체 무결성, 참조 무결성, 도메인 무결성
4. **도메인 무결성**: 속성이 정의되어 있는 영역을 벗어나지 않도록 규정

* **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 입력 받을 때 NULL을 허용하지 않는다.

1. **참조 무결성**: 참조 관계에 있는 두 테이블의 데이터가 항상 일관된 값을 가지는 규정

* **Foreign key**: 다른 테이블의 데이터를 참조할 때 없는 값을 참조할 수 없도록 제약을 주는 것으로 참조될 테이블이 먼저 만들어지고 값이 입력되어야 한다.
* **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

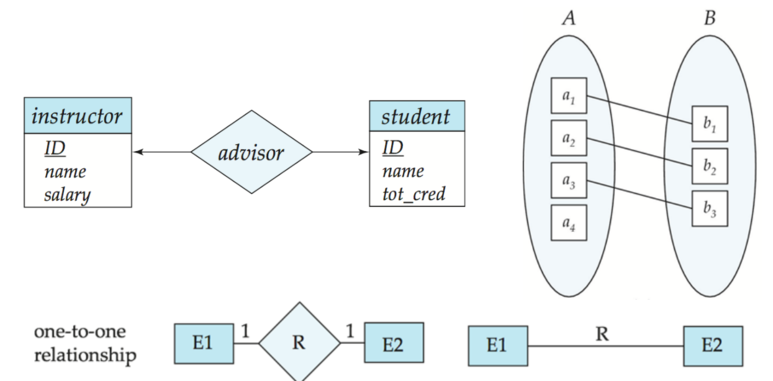
1. **개체 무결성**: 하나의 테이블에 중복된 행이 존재하지 않도록 규정 (기본키 제약)

* **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

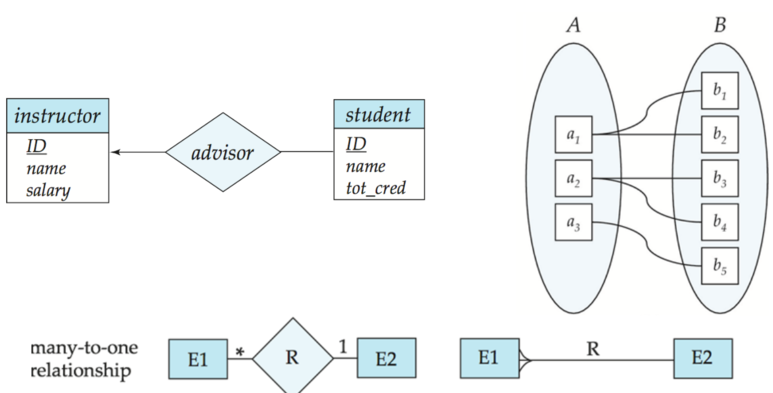
**E-R model (Entity Relationship model)**

* + 개념적 데이터 모델로 Entity(개체)와 entity간 관계를 나타낸다.
  + 이 모델을 통해 relational model을 구현

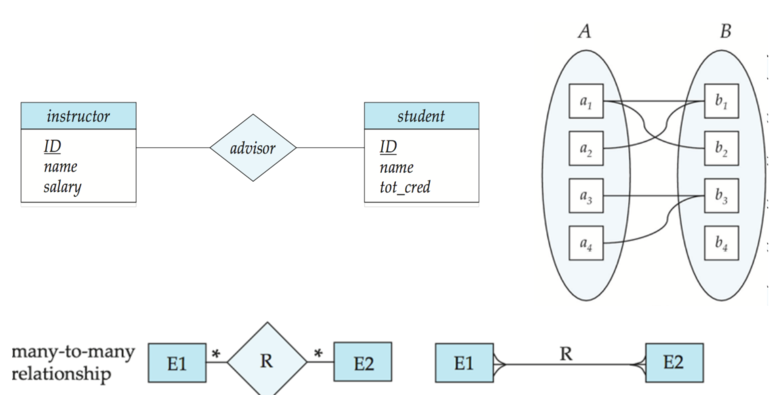
1. Entity set
   * Entity: 실제로 존재하는 object(물체)이고 다른 물체와 구별되어야 한다. 주로, 명사로 특정한 인물, 특정 회사, 특정 신물 등이 해당됨 (entity는 속성을 가진다. 예를 들어, 인물의 속성으로 이름과 주소 등이 있다.)
   * Entity set: entity의 집합으로 같은 성질을 가지고 있는 entity들을 의미 (예를 들어, 사람들의 집합, 회사들의 집합 등)
2. Relationship set
   * Relationship: entity간의 관계를 의미. 예를 들어, 학생 entity와 강사 entity사이에는 담당자 advisor라는 relationship이 존재한다.
3. Mapping 관계 유형
   * One to one: relationship set에서 두 entity set으로 화살표 방향이 있다.



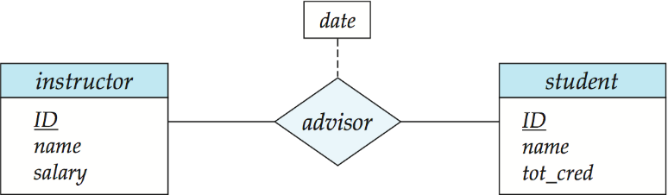
* + One to many: 일대다 관계로 일을 나타내는 entity set에게 화살표가 있다.



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



* + E-R Diagrams



* + 1. Rectangle: entity set을 의미
    2. Diamond: relationship set
    3. Attribute: entity rectangle내부에 존재
    4. Primary key: attributes 중 밑줄 그어진 속성
    5. Relationship set에서 점선으로 이어진 rectangle은 relationship set의 고유 속성을 의미한다. (없을 수도 있다.)

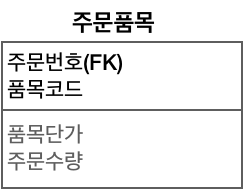
**정규화(Normalization)**

1. 관계형 데이터베이스의 설계에서 최소화하게 데이터를 구조화하는 프로세스를 말한다.
2. 목표: 이상이 있는 관계를 재구성하여 작고 잘 조직된 관계를 생성하는 것
3. 목적 (주로 상위 두가지를 목적으로 함)
   1. 불필요한 데이터(data redundancy)를 제거
   2. 각종 이상현상(Anomaly)을 방지하기 위해, 테이블의 구성을 논리적이고 직관적으로 한다.
   3. 데이터베이스 구조 확장 시 재디자인을 최소화
   4. 다양한 관점에서의 query를 지원하기 위해
   5. 무결성 제약조건의 시행을 간단하게 하기 위해
4. 이상현상의 종류
   1. 갱신 이상: 반복된 데이터 중 일부를 갱신할 시 데이터의 불일치 발생 가능
   2. 삽입 이상: 불필요한 정보를 함께 저장하지 않고는 어떤 정보를 저장하는 것이 불가능
   3. 삭제 이상: 필요한 정보를 함께 삭제하지 않고는 어떤 정보를 삭제하는 것이 불가능
5. 정규화 법칙은 총 6개(1NF, 2NF, 3NF, BCNF, 4NF, 5NF)로 나뉘며 실무적으로는 4차, 5차 정규화까지 가는 경우가 드물다.
6. **1차 정규형 (1NF, First Normal Form)**: 릴레이션 **모든 속성에 대한 domain의 값이 atomic**(하나의 cell에 값이 한 개)하도록 만들어진 정규형

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

1. **2차 정규형 (2NF, Second Normal From)**: 1NF를 만족하면서 릴레이션에 존재하는 **부분 함수적 종속을 제거**하여, **모든 속성이 기본키에 완전 함수 종속**이 되도록 만들어진 정규형

🡪부분적 함수 의존: 기본 키를 이루는 속성들 중 한 속성에만 종속된 다른 컬럼이 존재함을 의미 (부분적 함수 종속이 발생하면 데이터의 중복 발생)

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.

🡪해결: 하나의 테이블을 쪼개어 제2정규화를 만족하는 테이블들을 만든다. (부분적 함수 종속을 형성하는 속성들에 대한 테이블 따로 만들기)

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

1. **3차 정규형 (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.

🡪해결방안: 2NF와 마찬가지로 중복이 발생하게 된다. 따라서 테이블 나누기

1. **BCNF**: 3차 정규형을 만족하면서 **모든 결정자가 super key인 경우**(a🡪b, a 는 R의 슈퍼키) 해당 relation은 BCNF를 만족한다.

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

1. **4차 정규형** **(4NF, Fourth Normal Form)**: BCNF를 만족하면서 릴레이션에서 **다치 종속 관계를 제거**한 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.

1. **5차 정규형 (5NF, Fifth Normal Form)**: 후보키를 통하지 않은 조인종속(Join Dependency)을 제거한 정규형이다.
2. 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.

**역정규화(Denormalization)**

* 1. 정규화를 통해 분리되었던 릴레이션에서 중복을 허용하고 다시 통합하거나 분할하여 구조를 재조정하는 과정이다. 데이터베이스의 물리적 설계 과정에서 성능을 향상시키기 위해 역정규화를 실행한다
  2. 장점: 원하는 값을 찾을 때 시간을 아낄 수 있다. Join을 안하기 때문
  3. 단점: 업데이트 시 불필요한 추가 공간과 시간이 필요, 추가적인 코딩 작업과 추가 코드에 대한 에러의 가능성이 존재(코딩작업 관련 단점은 view를 사용하면 사라진다)

---------------------------------------------------------------------------------

여기부터는 os에 내용 존재

**Physical 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)** 🡨 OS에서 설명

1. 여러 디스크를 병렬적으로 사용함으로써 더 높은 공간과 속도를 가진다. 또한, 데이터를 중복 저장함으로써 신뢰성을 높인다. (정보 손실을 막음)
2. Mirroring: 디스크 내용을 복제하여 같은 내용이 2개의 디스크에 존재한다.
3. Bit-level striping: 디스크 병렬 방식 중 하나로, 한 bit씩 저장 디스크를 나눈다. 읽는 시간은 빨라지지만 접근시간이 느려진다.
4. Block-level striping: block단위로 저장 디스크를 나눈다.
5. LEVEL
   1. RAID 0: block striping, 복제 없음 / 데이터 손실 확률이 높아진다.
   2. RAID 1: block striping, mirrored disks / n개의 디스크에 대해 n개만큼 내용 복제한 디스크 존재
   3. RAID 2: bit striping, ECC(에러 감지하는 코드) / ECC는 데이터 통신에서 주로 쓰인다.
   4. RAID 3: bit-interleaved Parity(parity bit를 사용하는 방식 🡪 데이터 손실 시 이를 이용해 복구)
   5. 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
   6. RAID 5: block-interleaved distributed parity / parity를 모든 disk에 분배해서 저장한다. 병목현상을 피할 수 있다.
   7. RAID 6: P+Q redundancy scheme / 5와 유사하며 데이터 복제도 같이 한다. Parity가 2배 늘어남 (별로 사용 안한다. 비용이 비쌈)
   8. 대부분을 1 또는 5 중 선택하며 성능은 1이 더 좋고 용량은 5가 더 좋다. 1을 가장 많이 씀