관계형 데이터베이스 2장

- 관계형 데이터 모델
- 관계형 데이터베이스
- 관계 대수

관계형 데이터 모델

- 데이터 모델
  - 모델링? - 단순하게 표현하는것, 추상화 시키는것
  - 물리적 혹은 추상적으로 존재하는 실생활(현실세계에 존재하는 어떤 명사? 같은것들)를 **단순화되고 정형화 된 형태로 표현하는 하나의 방식** 또는 규범 - 누가봐도 무언가라는 것을 알 수 있을 정도로 단순하게 표현, 추상화
  - 실제 데이터가 갖는 특성을 살리면서, **목적에 맞게 관심있는 정보만을 단순화** 하여 표현하는 방식 - 데이터에 대한 조작이 가능해야함
  - 장점
    - 훨씬 저 용량으로 데이터를 저장 가능
    - 데이터에 대한 조작이 가능해짐 - 데이터모델화 하면서 연산이 가능하다.

릴레이션(relation)의 개념

- 관계형 데이터 모델(relational data model)
  - **테이블 형식**(표)을 이용하여 데이터들을 정의하고 설명한 모델
  - 실세계의 데이터를 누구나 직관적으로 이해할 수 있는 형태로 기술할 수 있는 간단한 방식을 제공
  - 테이블을 **릴레이션**(relation)이라 부름
- 릴레이션(relation)
  - 수학적으로, 두 개 이상의 집합으로부터 각 집합을 구성하는 원소들의 순서쌍에 대한 집합을 의미

    이름 = {홍길동, 김광식, 박철수, 최용만}

    주소 = {서울, 대전, 대구, 부산}

    ⇒ 순서쌍 : {<홍길동, 서울>, <김광식, 대전>, <박철수, 서울>, <최용만, 부산>}

이 름	주소
홍길동	서울
김광식	대구
박철수	서울
최용만	광주

  - 순서쌍을 테이블로 표현한 예
- 속성(attribute) – 필드, 컬럼
  - 릴레이션을 구성하는 각 열(column)의 이름
  - 예) 주소록 릴레이션을 구성하는 속성
    - 이름, 전화번호, 주소, 생일
- 튜플(tuple) – 레코드, 행
  - 릴레이션의 각 행
  - 예) 주소록 릴레이션의 한 튜플
    - <홍길동, 880-1234, 서울, 3월 15일>
- 이 책에서는 테이블, 필드, 레코드란 용어를 사용함

    - 릴레이션 - **테이블**
    - 속성 - **필드**(field), 컬럼(column)
    - 튜플 - **레코드**(record), 행(row)
- 도메인(domain)
    - 각 필드에 입력 가능한 값들의 범위, 즉 각 필드가 가질 수 있는 모든 값들의 집합 혹은 범위
    - 원자값(atomic value, 더 이상 분리되지 않는 값)이어야 함 - 집합형태X, 여러개가 들어가면 안된다.
    - '나이' 필드 - 정수, 지나치게 큰수는 안됨, 이런느낌이 도메인이다.
    - 예) 주소록의 도메인
        - 이름: 개인 이름들로 구성된 문자열 집합
        - 전화번호: “ddd-ddd-dddd”의 형식으로 구성된 문자열의 집합 (d는 0부터9까지 의 숫자)
        - 주소: 도시를 나타내는 문자열의 집합
        - 생일: “dd월dd일”로 구성된 문자열의 집합
- 널(null)
    - 특정 필드에 대한 값을 **알지 못하거나 아직 정해지지 않**아 입력하지 못한 경우의 필드의 값 - unknown
    - 0이나 공백 문자와는 다름
    - **NULL은 확실하지 않으므로(모르니까) 어떠한 질문에도 거짓으로 취급됨 - 포함이 안된다. - 주소가 아직 정해지지 않음은 예외**

테이블 스키마와 테이블 인스턴스 *

- 테이블 스키마(table schema, 스키마)
    - 테이블 정의에 따라 만들어진 데이터 구조
        - R(A1 , A2 , ..., An) R : 테이블의 이름 A1 , A2 , ..., An : 필드들의 이름
    - 예)
        - 신입생(학번, 주민등록번호, 이름, 주소, 학과명) - 차수가 5
- 차수(degree) - 많이는 안씀.
    - 테이블 스키마에 정의된 필드의 수
    - 차수 = 1 : 단항 테이블(unary relation)
    - 차수 = 2 : 이항 테이블(binary relation)
    - 차수 = n : n 항 테이블(n-ary relation)
- 테이블 인스턴스(table instance, 인스턴스)
    - 테이블 스키마에 현실 세계의 데이터를 레코드로 저장한 형태
    - 스키마는 한번 정의하면 거의 변함이 없지만 인스턴스는 수시로 바뀔 수 있음 – 레코드의 삽입, 삭제, 수정 등
    - 보통 스키마(데이터 구조)를 정해놓으면 잘 안바꿈 - 자주바뀐다? 프로그램 설계가 잘못된것임임
    - 데이터를 넣을때마다의 인스턴스가 변함 - 레코드의 수정,삭제등
    - 즉 인스턴스는 수시로 바뀔수 있지만 스키마는 잘 변함이 없음
- 기수(cardinality) - 잘안
    - 테이블 인스턴스의 레코드의 수

테이블의 특성 *

- 중복된 레코드가 존재하지 않음 - 이론상으로는 그렇지만 존재하기도 함
    - 테이블 인스턴스는 레코드들의 “집합”임
- 레코드간의 순서는 의미가 없음 - 순서가 바뀌어도 모양은 다르지만 같은 인스턴스임
    - 테이블 인스턴스는 레코드들의 “집합”임
    - ‘첫번째 레코드’, ‘두번째 레코드’란 표현은 의미 없음
- 레코드 내에서 필드의 순서는 의미가 없음 - DBS마다 다른 방식으로 정렬
    - 테이블 스키마는 필드들의 집합으로 표현됨

    - '첫번째 필드', '두번째 필드'란 표현은 의미 없음
- 모든 필드는 원자값을 가짐
- 이것들은 이론적인 특징이다. -> dbms에서 아주 강하게 이런 특성들을 규제하지 않는다. 즉 실제로는 조금 다르다.

키 (Key)

- 키는 왜 필요한가?
    - 레코드간의 **순서가 의미가 없으므로** 레코드를 구분하기 위해서는 각 레코드의 **값**을 이용함
    - 키(key) - 딱히 KEY 될만한게 없으면 본인이 KEY를 만들어도 됨 - 중복만 안되면 된다.
        - 필드들의 일부로 **각 레코드들을 유일하게 식별해낼 수 있는 식별자** (identifier)
        - 구분을 할 수 있는 레코드를 key로 해야
        - 일반적으로 하나의 필드를 지정하여 키로 지정하나, 여러 개의 필드들로 키를 구성할 수 도 있음
        - 두 개 이상의 필드로 구성된 키를 복합키(composite key)라고 함
        - 예를 들어 신입생 테이블의 학번 또는 주민등록번호 필드는 각 레코 드간에 유일하므로 키가 될 수 있음
            - 그러나 학과명은 키가 될 수 없음
    - **관계형 데이터 모델에서 특정 레코드를 구별하거나 탐색하기 위한 유일한 방법**

수퍼키, 후보키, 기본키의 개념 *

- 수퍼키(super key)
    - 아무런 제약 조건 없이 레코드들을 식별할 수 있는 필드의 집합, 복합키도 가능
    - 예) (주민등록번호) (학번, 주민등록번호) (주민등록번호, 이름) (이름, 주소) 등
- 후보키(candidate key)
    - 최소한의 필드만으로 구성된 키
    - 예) (학번) (주민등록번호)(이름, 주소)(이름, 학과명)
- 기본키(primary key)
    - 후보키 중에서 식별자로 정의한 하나의 키
    - 키는 여러개를 정의할 수 없음
    - 선택하는건 데이터베이스 관리자의 몫
    - 되도록 하나의 필드로 구성된 후보키를 선정하는 것이 유리함
    - 예) (학번)
    - 나머지의 필드들을 가지고 후보키로 설정하는것은 현재 인스턴스에서는 가능할지 모르지만 앞으로의 변화할 모든 상황에 대해서는 키가 될수 있다고 보장할수 없다. 즉 **진정한 후보키는 학번과 주민등록번호밖에 없다.**
    - **그렇다면, (주민등록번호)(이름, 주소)(이름, 학과명) 모두 후보키 자격 이 있는가?**
        - 중복된 레코드가 들어올 가능성이 있는 절대로 key가 될 수 없다.
        - **즉 모든 가능한 인스턴스에 대해서도 키가 될 수 있어야 함** - 나중에도 식별이 가능해야 한다.
        - 지금은 후보키이다! 는 하나도 안중요함

키가 널(null)이 될 수 있나?

- 기본키는 식별자의 기능을 함
- 기본키로 정의된 필드가 널을 갖게 되면 이러한 식별 기능 을 상실
    - 예를 들어 두 개의 레코드에 대한 기본키 값이 동시에 널이면 그 들은 서로 구별할 수 없음
    - 따라서 **기본키는 널이 될 수 없음**

외래 키(foreign key) - 테이블간의 관계계

학생 (학번, 주민등록번호, 이름, 주소, 학년, 학과번호)
학과 (학과번호, 학과명, 과사무실)

학번	주민등록번호	이름	학년	학과번호
1292001	900424-1825409	김광식	2	920
1292002	900305-1730021	김정현	2	920
1292003	891021-2308302	김현정	2	920
1292301	890902-2704012	김현정	2	923
1292303	910715-1524390	박광수	1	923
1292305	921011-1809003	김우주	2	923
1292501	900825-1506390	박철수	1	925
1292502	911011-1809003	백태성	2	925

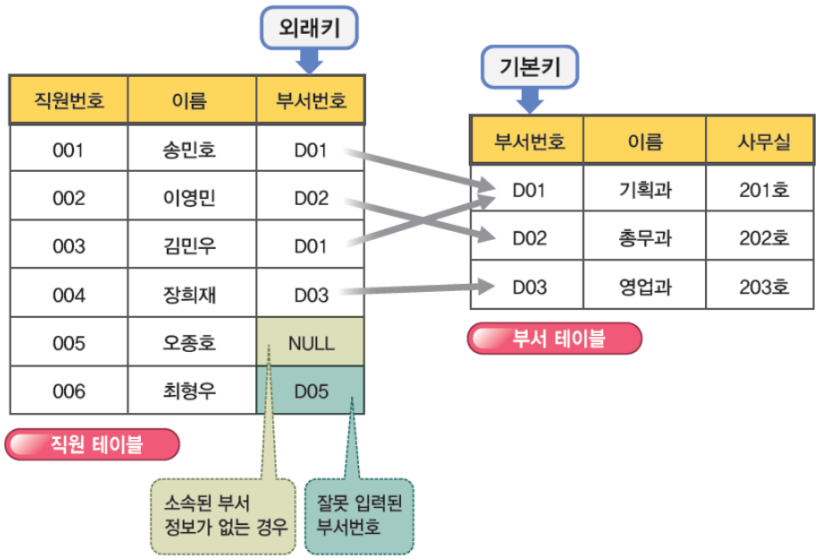
(a) 학생 테이블 인스턴스

학과번호	학과명	과사무실
920	컴퓨터공학과	201호
923	산업공학과	207호
925	전자공학과	308호

(b) 학과 테이블 인스턴스

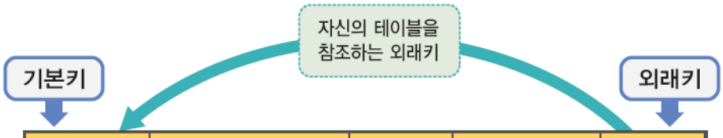
- 하나의 테이블에서만 쓰는 개념 아니라 2개 이상의 테이블 간의 관계를 맺을때 쓰는 개념
- 무결성을 위해서 - 이 필드에는 이것만 들어가야 한다! 를 위해 설정한다고도 할 수 있따. - NULL 들어갈수 있음. 참조하는 테이블에서는 key가 아니기 때문에
- 외래키 (참조한다.) - 외래키가 되는 필드(참조 된다.)
- a 테이블의 학과번호는 b 테이블의 학과번호를 참조하고 b는 참조된다.
- 다른 테이블의 기본 키를 참조하는 필드집합
- 두 테이블 스키마 R1 , R2에 대해,
  - R1의 어떤 필드집합 FK가 다음 두 조건을 만족하면, FK는 R2 의 기본키인 PK를 **참조하는** R1의 외래키임 - 위의 표에선 학과번호
  - FK의 필드들은 테이블 스키마 R2 의 기본 키 PK와 동일한 도메인을 가짐
  - R1의 각 레코드의 FK값은 R2 의 레코드 중 하나의 PK값과 일치하거나 널이 됨 - R2의 레코드 값이 아닌것은 R1의 레코드로 넣을수 없음
  - 여기서 R1 레코드의 FK값이 널이 된다는 것은 알지 못하거나 아직 결정 되지 않았다는 것을 의미함
  - 이때,
    - R1 : 참조하는 테이블(referencing table)
    - R2 : 참조되는 테이블(referenced table)
- 왜래키 예시

직원(직원번호, 이름, 부서번호)
부서(부서번호, 부서명, 사무실)



- 외래키는 자기 테이블도 참조 가능

교수 (교수번호, 주민등록번호, 이름, 학과명, 학과장)



교수번호	주민등록번호	이름	학과명	학과장
92001	590327-1839240	이태규	컴퓨터공학과	92001
92002	690702-1350026	고희석	컴퓨터공학과	92001
92301	741011-2765501	최성희	산업공학과	92302
92302	750728-1102458	김태석	산업공학과	92302
92501	620505-1200546	박철재	전자공학과	NULL
92502	681006-1023456	강만희	전자공학과	NULL

관계형 데이터베이스(relational database)

- 정의
  - **관계형 데이터 모델**에 기반하여 **하나 이상의 테이블**로 실세계를 표현한 데이터베이스
    - 실세계를 관계형 데이터 모델이라는 추상적인 도구를 이용하여 표현 한 것 - 이론적인 모델이다.
    - 테이블들을 컴퓨터의 기억 장치에 어떠한 방법으로 저장할 것인가에 대한 **물리적인 구조까지 정의한 것은 아님**
- 관계형 데이터베이스가 하나 이상의 테이블로 구성되어 있을 때
  - 데이터베이스 스키마(database schema) : 테이블 스키마의 집합
  - 데이터베이스 인스턴스(database instance) : 테이블 스키마들에 대한 테이블 인스턴스의 집합

데이터베이스 : 하나이상의 테이블로 구성된걸 의미

관계 대수 (Relational Algebra)

- 질의어(query language)
  - 삽입, 삭제, 수정, 검색 등의 데이터 조작을 위한 연산들을 표현하기 위한 언어
  - 절차적 언어인것도 있고 비 절차적 언어인것도 있음
- 절차적 언어(procedural language)
  - 사용자가 원하는 결과를 얻기 위해 수행되어야 할 일련의 절차를 명 시해야 하는 언어
  - 예: C, C++와 같은 대부분의 프로그래밍 언어
- 비절차적 언어(non-procedural language) , 혹은 선언적 언어
  - 수행 절차는 기술하지 않고 사용자가 원하는 결과만을 형식적으로 명시하는 언어
  - 실질적 수행절차는 시스템 내부적으로 결정해야 함
- 관계 대수 : 관계형 데이터베이스에서의 대표적 질의어, 절차적 언어
  - 관계형 데이터베이스의 테이블들을 조작하기 위한 언어
  - 관계 대수(relational algebra) - 이론적인 언어, 실무에선 잘 안씀
    - 절차적 언어
    - 수학에서의 수식구조와 유사

        - 피연산자(operand) : 테이블
        - 연산자(operator)
            - 단항 연산자(unary operator)
            - 이항 연산자(binary operator)
    - 관계 해석(relational calculus)
        - 비절차적 언어
        - 이 책에서는 다루지 않음

관계대수의 연산 종류

- 기본 연산
    - 선택 연산
    - 추출 연산
    - 재명명 연산
    - 집합 연산
    - 카티션 프로덕트
- 추가연산
    - 조인
    - 자연 조인
    - 외부 조인
    - 지정 연산
    - (나누기 연산) - 굳이 안함

선택(selection) - 가로로 자른다.

- 하나의 테이블에서 주어진 조건을 만족하는 **레코드**들을 검색하는 기능

    형식 $\sigma_{\langle 조건식 \rangle}(\langle 테이블이름 \rangle)$

    - <테이블이름>
        - <테이블이름>은 연산의 대상이 되는 테이블의 이름
    - <조건식>
        - 비교연산자(<, >, <=, >=, <>)와 부울 연산자($\vee$, $\wedge$, NOT)의 조합
-
- 테이블에 대한 연산 결과는 또다른 테이블이 됨
- 선택 연산에서의 NULL 처리 교제 48p

- Professor 테이블에 null이 입력된 예

prof_id	resident_id	name	dept_id	position	year_emp
92001	590327-1839240	이태규	920	교수	1997
92002	690702-1350026	고희석	NULL	부교수	2003
92301	741011-2765501	최성희	NULL	부교수	2005
92302	750728-1102458	김태석	923	교수	1999

$\sigma_{dept\_id='920'}(professor)$

연산 결과

prof_id	resident_id	name	dept_id	position	year_emp
92001	590327-1839240	이태규	920	교수	1997

$\sigma_{dept\_id\neq'920'}(professor)$

연산 결과

prof_id	resident_id	name	dept_id	position	year_emp
92302	750728-1102458	김태석	923	교수	1999

- **dept_id가 '920'인지 그렇지 않은지 알지 못함 따라서 검색 결과에서 배제해야 함**
- null 은 unknown(아직 알지 못한다. 정해지지 않았다)

추출(project) - 세로로 자른다.

- 테이블에서 사용자가 원하는 **필드**만을 결과로 출력하는 연산

형식 $\pi_{\langle 필드리스트\rangle}(\langle 테이블이름\rangle)$

  - <테이블이름>
    - <테이블이름>은 연산의 대상이 되는 테이블의 이름
  - <필드리스트>
    - 테이블에서 추출하고자 하는 필드들의 리스트
- 필드만 추출하기 때문에 필드를 ,로 연결해서 씀
- 중복을 제거해야함 - 관계형 모델은 중복된 레코드들을 허용하지 않음(테이블의 연산결과는 또다른 테이블)
  - 참고: 관계 대수에서는 중복 레코드를 허용하지 않으나 실제 실무 DBMS에서는 대부분 허용함

연산자들의 조합

- 관계 대수 연산자들은 **상호 중첩**하여 사용 가능
  - 2000년 이후에 임용된 '부교수'들의 레코드를 검색

질의 6 $\sigma_{position='부교수'}(\sigma_{year\_emp>=2000}(professor))$

중간 결과

prof_id	resident_id	name	dept_id	position	year_emp
92002	690702-1350026	고희석	920	부교수	2003
92301	741011-2765501	최성희	923	부교수	2005
92501	620505-1200546	박철재	925	조교수	2007
92502	740101-1830264	장민석	925	부교수	2005

최종 결과

prof_id	resident_id	name	dept_id	position	year_emp
92002	690702-1350026	고희석	920	부교수	2003
92301	741011-2765501	최성희	923	부교수	2005
92502	740101-1830264	장민석	925	부교수	2005

- 이렇게도 할수 있고, and와 or 연산자를 이용해도 됨
- 위의 질의6은 관계 대수이고, 식을 이용해 절차를 표시함으로 절차적 언어이다
- 2000년 이후의 어쩌구는 내가 바라는 결과만 클로 표시함으로 비절차적 언어이

- **선택 연산**은 **교환 법칙**이 성립

$$\sigma_{\langle 조건식1\rangle}(\sigma_{\langle 조건식2\rangle}(\langle 테이블이름\rangle)) \equiv \sigma_{\langle 조건식2\rangle}(\sigma_{\langle 조건식1\rangle}(\langle 테이블이름\rangle))$$

예)

$$\sigma_{year\_emp\rangle=2000}(\sigma_{position='부교수'}(professor)) \equiv \sigma_{position='부교수'}(\sigma_{year\_emp\rangle=2000}(professor))$$

- **추출 연산**에 대해서 다음의 두 질의는 동일한 결과

$$\pi_{name,\ position}(\pi_{prof\_id,\ name,\ position}(professor))$$

$$\pi_{name,\ position}(professor)$$

- 다음은 잘못된 질의임 - 한번 추출된것에서 없는걸 추출하는 연

$$\pi_{prof\_id,\ name,\ position}(\pi_{name,\ position}(professor))$$

- 다음의 두 질의가 동일하기 위한 조건은?

$$\pi_{\langle 필드리스트1\rangle}(\pi_{\langle 필드리스트2\rangle}(\langle 테이블이름\rangle)) \equiv \pi_{\langle 필드리스트1\rangle}(\langle 테이블이름\rangle)$$

  - 필드리스트 1 이 필드리스트 2에 포함이 되어야함

- 일반적으로는 선택과 추출 연산의 조합으로 질의를 표현
  - 순서가 중요하다.
    - 2000년 이후에 임용된 '부교수'들의 이름을 검색

질의 10 $\pi_{name}(\sigma_{year\_emp\rangle=2000\ \wedge\ position='부교수'}(professor))$

연산 결과

name
고희석
최성희
장민석

    - 위의 연산에서 선택과 추출 연산의 순서를 바꿀 수 있는가?

- 안된다. 세로로 name만 먼저 짜르면 다음 연산의 조건을 줄수가 없어짐
- 그래서 일반적으로 테이블에 대헤 selection과 projection를 같이 쓰는 대부분의 경우에 selection를 먼저 하고 projection을 한다.

재명명 연산

- 테이블에 이름을 부여하거나 변경하는 연산
  - 테이블에 이름을 부여하거나 변경하는 연산

    형식 $\rho_{\langle 테이블명1\rangle}(\langle 테이블명2\rangle)$

    - <테이블명2>의 이름을 <테이블명1>로 변경하라는 의미

    형식 $\rho_{\langle 테이블명1\rangle(\langle 필드리스트\rangle)}(\langle 테이블명2\rangle)$

    - <테이블명2>의 이름을 <테이블명1>로 변경하는 동시에 <테이블명2>에 정의된 필드명들을 모두 <필드리스트>로 변경
  - professor 테이블에서 dept_id가 '920'인 교수들의 이름을 검색

    질의 13 $\pi_{com\_dept.name}(\rho_{com\_dept}(\sigma_{dept\_id='920'}(professor)))$

  - 강의실이 '301호'인 class의 prof_id와 enroll을 검색

    질의 14 $\rho_{class301(id,\ number)}(\pi_{prof\_id,\ enroll}\ (\sigma_{classroom='301'}(class)))$

    연산 결과

id	number
92301	40
92301	30
92502	30

- 질의 14는 테이블의 이름을 class301, 레코드의 이름을 id와 number로 바꾼다.
- 테이블이름.필드이름 - 필드의 소속이 어딘지 밝히기 위해
- 주의
  - 재명명 연산은 중간 결과나 최종 결과에 대한 테이블명이 변경됨
  - 본래 데이터베이스에 저장된 테이블명까지 변경되는 것은 아님

집합 연산

- 수학적 집합 이론에서 정의된 연산
  - 합집합(union)
  - 차집합(minus)
  - 카티션 프로덕트(Cartesian product) - 두 개의 테이블에서 각각의 레코드들을 서로 결합하여 하나의 레코드 로 구성하면서 가능한 모든 조합의 레코드들로 테이블을 생성
  - 교집합(intersection) → 차집합으로 정의할 수 있음
- 호환 가능한 테이블들(compatible relations) - 하나의 테이블로 만들기 위해서
  - 합집합, 차집합, 교집합 연산에서 두 피연산자의 **차수와 필드 이름들 이 동일**해야 함
  - 같은 이름의 필드들이라 하더라도 **도메인이 일치**해야 함

추가 연산 - 실무에서 많이 사용

- 조인(join) - 세타조인(theta join) (theta: Θ) 이라고 하기도
- 자연 조인 (natural join)
- 외부 조인 (outer join)
- 지정 연산

조인(join)

- 두 테이블로 부터 **특정 조건을 만족하는 레코드들**을 **하나의 레코드로 결합**하는 연산
- 결합을 하는 단계에서 어떤 조건을 만족하는 레코드들만 조합을 시키고 나머지는 결합할때 배제하자 - 선택적으로 결합을 하니까 중간결과테이블이 어마무시하게 크지 않음
- 카티션 프로덕트는 모든 가능한 조합에 의해 레코드들을 생성하지만 **조인은 특정 조건은 만족하는 레코드만**을 선택
- 세타(theta: Θ) 조인이라고도 함

  형식 〈테이블이름1〉 ⋈〈조건식〉〈테이블이름2〉

  - <조건식>
  - 조인조건(join condition)이라 부름
    - 조인 조건은 필드간의 동등비교(=)가 대부분이며 이를 동등조인(equijoin)이라 함
  - 다음과 같이 카티션 프로덕트로 표현가능

    $$\langle \text{테이블이름1} \rangle \bowtie_{\langle \text{조건식} \rangle} \langle \text{테이블이름2} \rangle \equiv \sigma_{\langle \text{조건식} \rangle} (\langle \text{테이블이름1} \rangle \times \langle \text{테이블이름2} \rangle)$$

자연 조인(natural join)

- 서로 다른 테이블에서 같은 이름을 갖는 두 필드를 찾고 그 필에 대한 동등 조인 중 하나의 필드를 제거하여 단순히 표현한 연산
- 필드의 이름이 같기때문에 필드를 2개 유지할 필요가 없음
- 세타조인의 조인 조건은 대부분의 경우에는 field 이름이 같다. 보통은 키와 외래키의 관계
- 조인조건을 스스로 유추. 두개의 테이블에 필드 이름이 같은게 있다. 이걸 조인을 해라.

  형식 〈테이블이름1〉 ⋈〈테이블이름2〉

  - 다음이 성립함

    $$R_1 \bowtie R_2 \equiv \pi_{R_1 \cup R_2} (\sigma_{R_1.A_1 = R_2.A_1 \wedge R_1.A_2 = R_2.A_2 \wedge \cdots \wedge R_1.A_n = R_2.A_n} (R_1 \times R_2))$$

    - $R_1 \cup R_2$: 필드들의 합집합
    - $A_1, A_2, \ldots, A_n$:공통 필드

  - 공통되는 필드가 없으면 카티션 프로덕트와 같음

외부 조인(outer join)

- 조인의 결과에 들어가지 않는 레코드도 넣기 위해

- 자연 조인의 예

name	address
김광식	서울
김현정	대전
조영수	대전

⋈

name	dept_name
김광식	컴퓨터공학과
김현정	산업공학과
이진영	전자공학과

=

name	address	dept_name
김광식	서울	컴퓨터공학과
김현정	대전	산업공학과

  - '조영수'와 '이진영'은 서로 일치되는 레코드가 없어 검색 결과에서 배제됨

- 외부 조인
  - 조인 조건에 만족되지 않은 레코드까지 검색 결과에 포함시키기 위한 방법
  - 서로 매치되지 않는 필드에 대해서는 NULL을 입력함
  - 종류
    - 왼쪽 외부조인(left outer join) 
    - 오른쪽 외부조인(right outer join) 
    - 완전 외부조인(full outer join) 

연산 결과

name	address	dept_name
김광식	서울	컴퓨터공학과
김현정	대전	산업공학과
조영수	대전	NULL

name	address	dept_name
김광식	서울	컴퓨터공학과
김현정	대전	산업공학과
이진영	NULL	전자공학과

name	address	dept_name
김광식	서울	컴퓨터공학과
김현정	대전	산업공학과
조영수	대전	NULL
이진영	NULL	전자공학과

지정(assignment) 연산

- 복잡한 질의를 여러 개의 질의로 분리하거나 중간 결과에 이름을 부여
- 최종 질의를 결과에 이름을 부여
- 연산 기호로는 ←를 사용
  - 예) student 테이블에서 3학년인 학생을 선택해서 그 결과 테이블을 junior이라는 이름으로 지정

질의 33 $junior \leftarrow \sigma_{year=3}(student)$

  - 다음의 두 질의는 동일

질의 29 $\pi_{year,\ semester,\ division}(\sigma_{title='데이터베이스'}(course \bowtie_{course.course\_id=class.course\_id} class))$

질의 34

$temp1 \leftarrow course \bowtie_{course.course\_id=class.course\_id} class$

$temp2 \leftarrow \sigma_{title='데이터베이스'}(temp1)$

$\pi_{year,\ semester,\ division}(temp2)$