

DB 구조 및 설계

[KEY](#)

[JOIN](#)

[정규화](#)

[RDB-NoSQL 차이](#)

KEY

- 데이터베이스에서 조건에 만족하는 행을 찾거나 순서대로 정렬할 때 다른 행들과 구별할 수 있는 유일한 기준이 되는 Attribute(속성)

학생 테이블

학번	주민번호	이름
1	971218-xxxxxxx	a
2	971218-xxxxxxx	b
3	971218-xxxxxxx	c

수강 테이블

학번	과목명
1	db
2	java
2	db
3	java

1. 후보키

- 테이블을 구성하는 속성들 중 행을 **유일하게 식별**할 수 있는 속성들의 부분집합
- 모든 테이블은 반드시 하나 이상의 후보키를 가져야 함
- 유일성 O, 최소성
 - 유일성 : 하나의 키 값으로 행을 유일하게 식별 가능
 - 최소성 : 키를 구성하는 속성들 중 꼭 필요한 최소한의 속성들로만 키를 구성
- 기본키가 될 수 있는 후보들
- ex) 학번, 주민번호

2. 기본키

- 한 테이블에서 특정 행을 유일하게 구별할 수 있는 속성
- Null값 불가
- 기본키로 정의된 속성에는 동일한 값이 중복되어 저장될 수 없음
- ex) 학생 : 학번 or 주민번호
수강 : 학번 + 과목명

3. 대체키

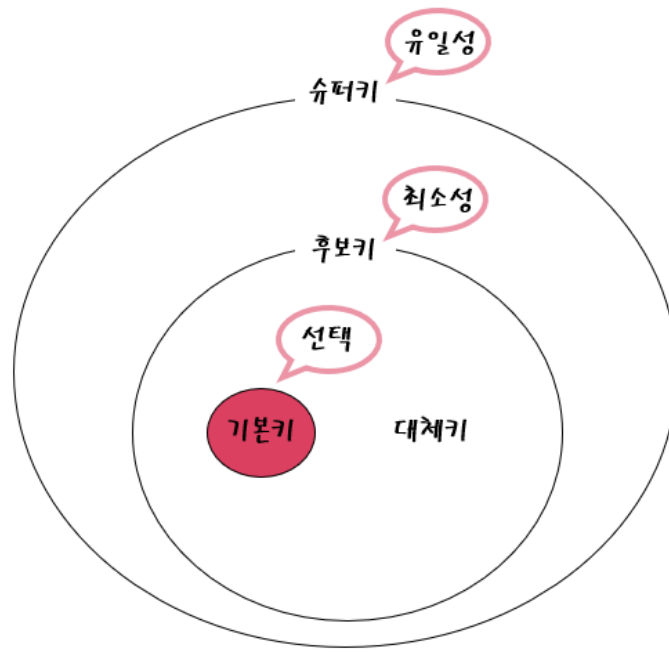
- 후보키가 둘 이상일 때 기본키를 제외한 나머지 후보키
- ex) 학생 테이블에서 학번이 기본키면 주민번호가 대체키

4. 슈퍼키

- 한 테이블 내에 있는 속성들의 집합으로 구성된 키
- 유일성 O, 최소성 X
- ex) 학생 : 학번, 주민번호, 학번+주민번호, 학번+이름, 주민번호+이름, 학번+주민번호+이름

5. 외래키

- 관계를 맺고 있는 테이블 1, 2가 있을 때 테이블 1이 참조하고 있는 테이블 2의 기본키와 같은 속성
- 외래키로 지정되면 참조 테이블의 기본키에 없는 값은 입력할 수 없다.



▼ 참고 (개체무결성, 참조무결성)

- **개체 무결성:** 기본키는 Null이 올 수 없으며, 기본키를 구성하는 어떠한 속성값이라도 중복값이나 Null값을 가질 수 없다
- **참조 무결성:** 외래키는 참조할 수 없는 값을 키로 가질 수 없다. 즉 외래키는 1. 기본키 값과 같거나, 2. 외래키가 자신을 포함하고 있는 테이블의 기본키를 구성하고 있지 않으면 Null값을 갖는다.

JOIN

컴공교수 테이블

ID	이름
1	a
2	b
3	c
4	d

강의 테이블

강의번호	교수ID	과목명
1	1	db
2	2	java
3	3	python
4	7	회계원론

• 내부조인(INNER JOIN)

- 양쪽 테이블 데이터 집합에서 공통적으로 존재하는 데이터만 조인해서 결과 데이터 집합으로 추출
- 교집합

```
SELECT *
FROM 교수, 강의
WHERE 교수.ID = 강의.교수ID;
```

교수.ID	교수.이름	강의.강의번호	강의.교수ID	강의.과목명
1	a	1	1	db
2	b	2	2	java
3	c	3	3	python

• 외부조인(OUTER JOIN)

- 조인 대상 테이블에서 특정 테이블의 데이터가 모두 필요한 상황에서 활용한다.

1. 왼쪽 외부 조인(LEFT OUTER JOIN)

- 우측 테이블에 조인할 컬럼의 값이 없는 경우 사용한다.
- 좌측 테이블의 모든 데이터를 포함하는 결과 집합을 생성한다.

```
SELECT *
FROM 교수 left join 강의
on 교수.ID = 강의.교수ID;
```

교수.ID	교수.이름	강의.강의번호	강의.교수ID	강의.과목명
1	a	1	1	db

교수.ID	교수.이름	강의.강의번호	강의.교수ID	강의.과목명
2	b	2	2	java
3	c	3	3	python
4	d	null	null	null

2. 오른쪽 외부 조인(RIGHT OUTER JOIN)

- 좌측 테이블에 조인할 컬럼의 값이 없는 경우 사용한다.
- 우측 테이블의 모든 데이터를 포함하는 결과 집합을 생성한다.

```
SELECT *
FROM 교수 right join 강의
on 교수.ID = 강의.교수ID;
```

교수.ID	교수.이름	강의.강의번호	강의.교수ID	강의.과목명
1	a	1	1	db
2	b	2	2	java
3	c	3	3	python
null	null	4	7	회계원론

3. 완전 외부 조인(FULL OUTER JOIN)

- 양쪽 테이블 모두 OUTER JOIN이 필요할 때 사용한다.

```
SELECT *
FROM 교수 full outer join 강의
on 교수.ID = 강의.교수ID;
```

교수.ID	교수.이름	강의.강의번호	강의.교수ID	강의.과목명
1	a	1	1	db
2	b	2	2	java
3	c	3	3	python
4	d	null	null	null
null	null	4	7	회계원론

- **셀프조인**
 - 동일 테이블 사이의 조인을 말한다.
 - 같은 테이블끼리 조인하는 것이므로 별칭(ALIAS)을 꼭 사용해야 한다.



정규화

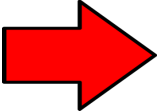
- 테이블 간에 중복된 데이터를 허용하지 않는 것
- 무결성(Integrity) 유지, DB의 저장 용량 확보

1. 제 1 정규화

- 테이블의 컬럼이 원자값(Atomic Value, 하나의 값)을 갖도록 테이블을 분해

고객취미들(이름, 취미들)

이름	취미들
김연아	인터넷
추신수	영화, 음악
박세리	음악, 쇼핑
장미란	음악
박지성	게임



고객취미(이름, 취미)

이름	취미
김연아	인터넷
추신수	영화
추신수	음악
박세리	음악
박세리	쇼핑
장미란	음악
박지성	게임

2. 제 2 정규화

- 제1 정규화를 진행한 테이블에 대해 완전 함수 종속을 만족하도록 테이블을 분해
- 완전 함수 종속 : 기본키의 부분집합이 결정자가 되어선 안된다

수강강좌

학생번호	강좌이름	강의실	성적
501	데이터베이스	공학관 110	3.5
401	데이터베이스	공학관 110	4.0
402	스포츠경영학	체육관 103	3.5
502	자료구조	공학관 111	4.0
501	자료구조	공학관 111	3.5

학생번호

강좌이름

성적

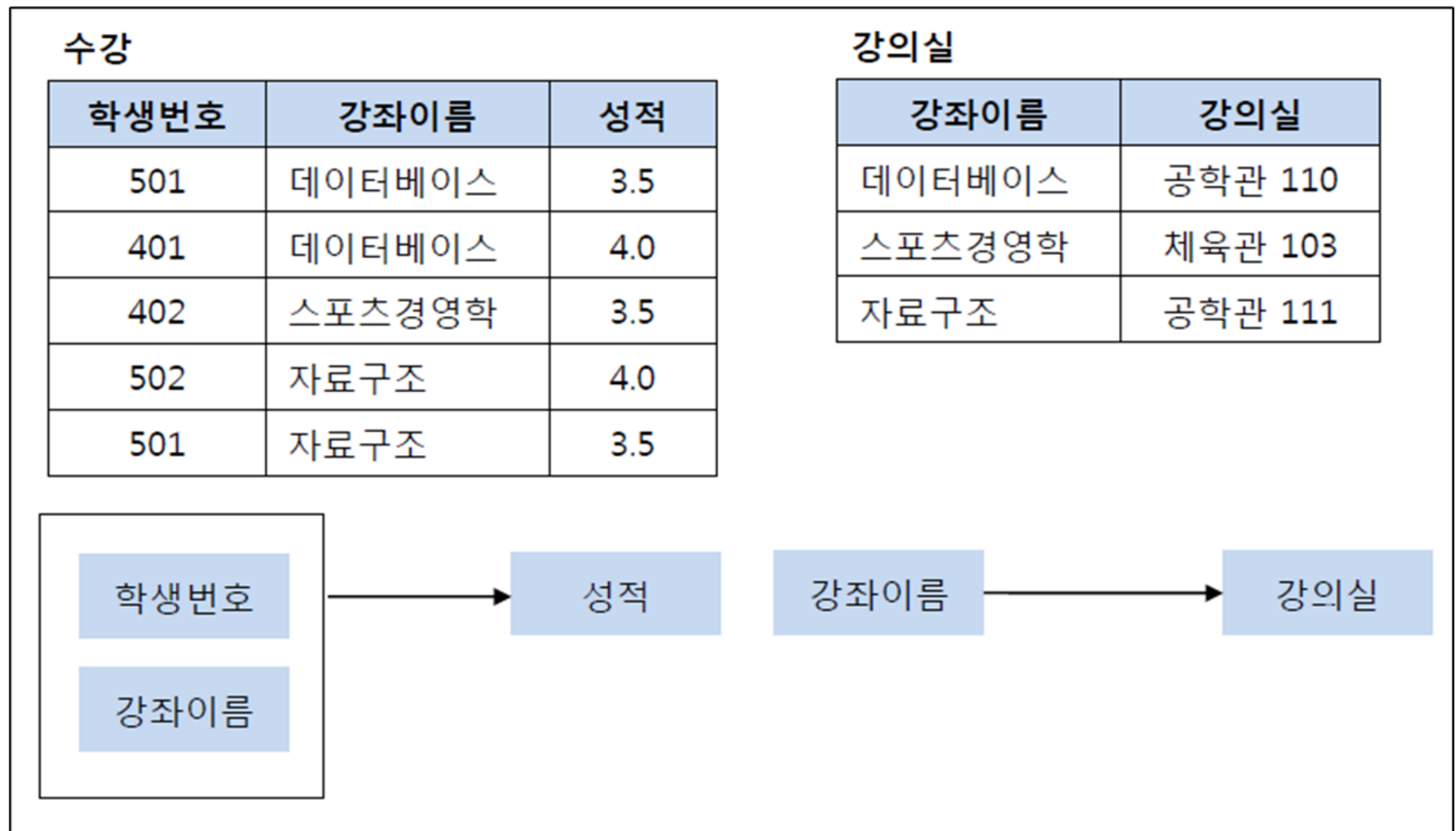
강의실

→

→

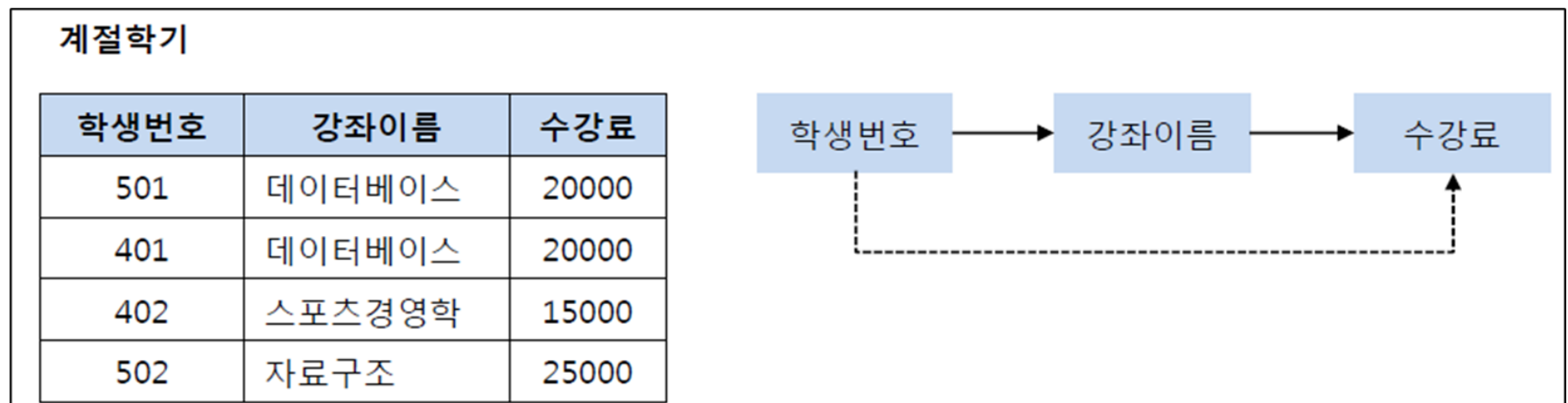
DB 구조 및 설계

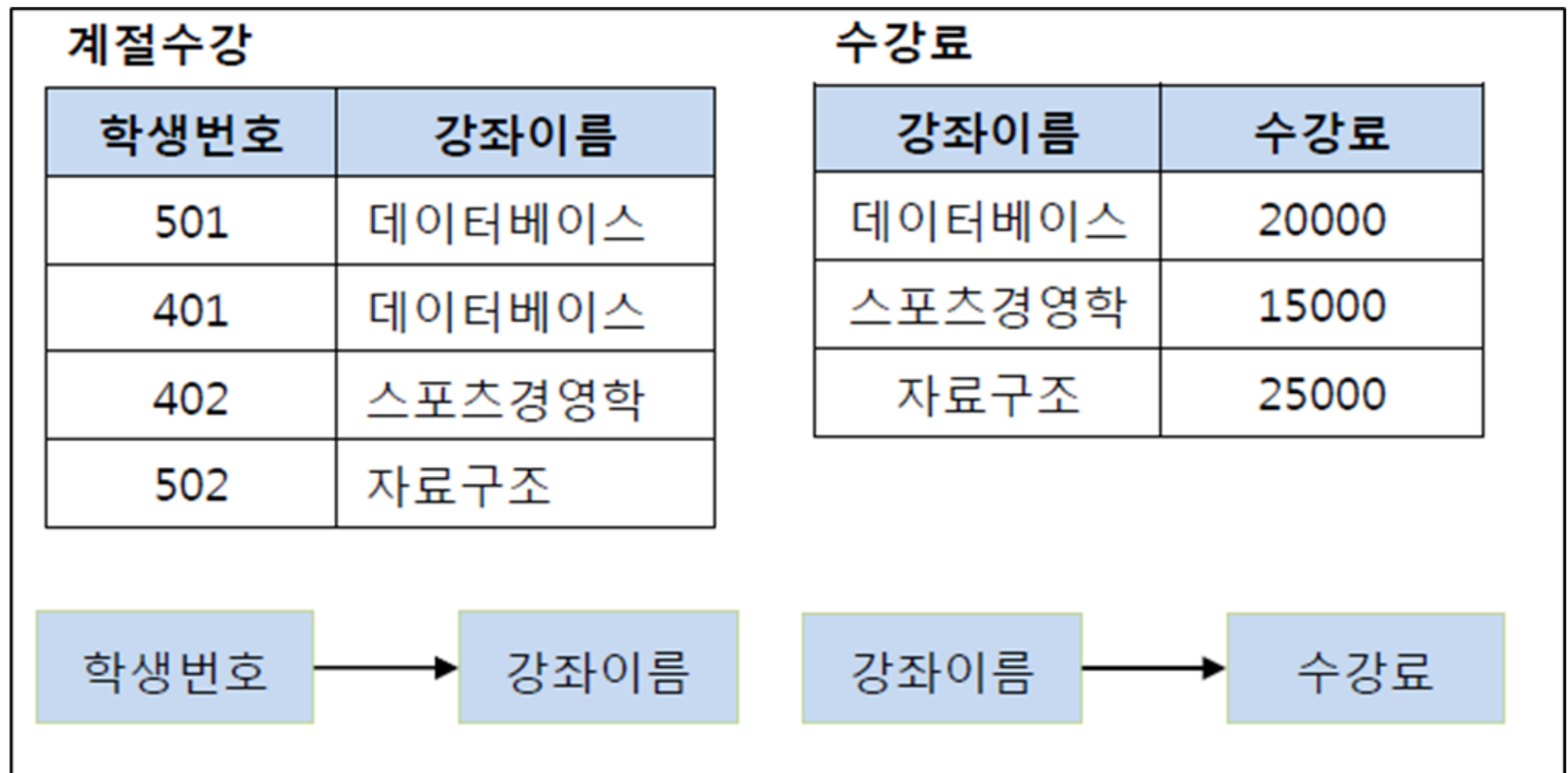
4



3. 제 3 정규화

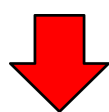
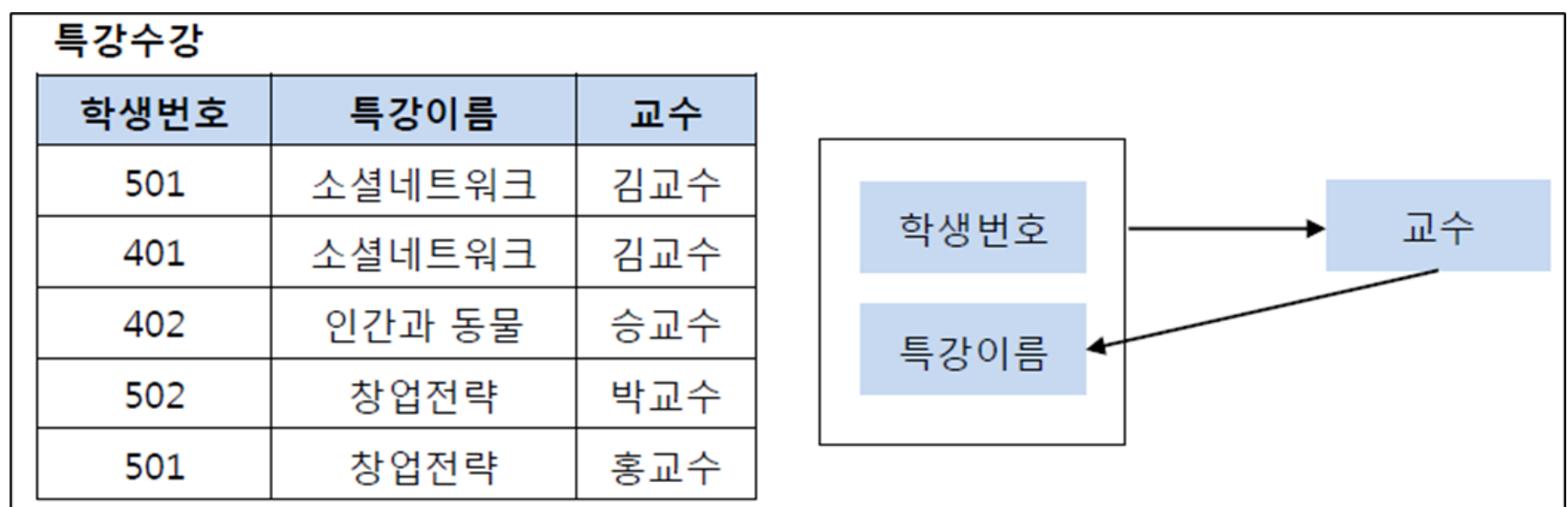
- 제2 정규화를 진행한 테이블에 대해 **이행적 종속을 없애도록** 테이블을 분해하는 것
- 이행적 종속 : A -> B, B -> C가 성립할 때 A -> C가 성립되는 것

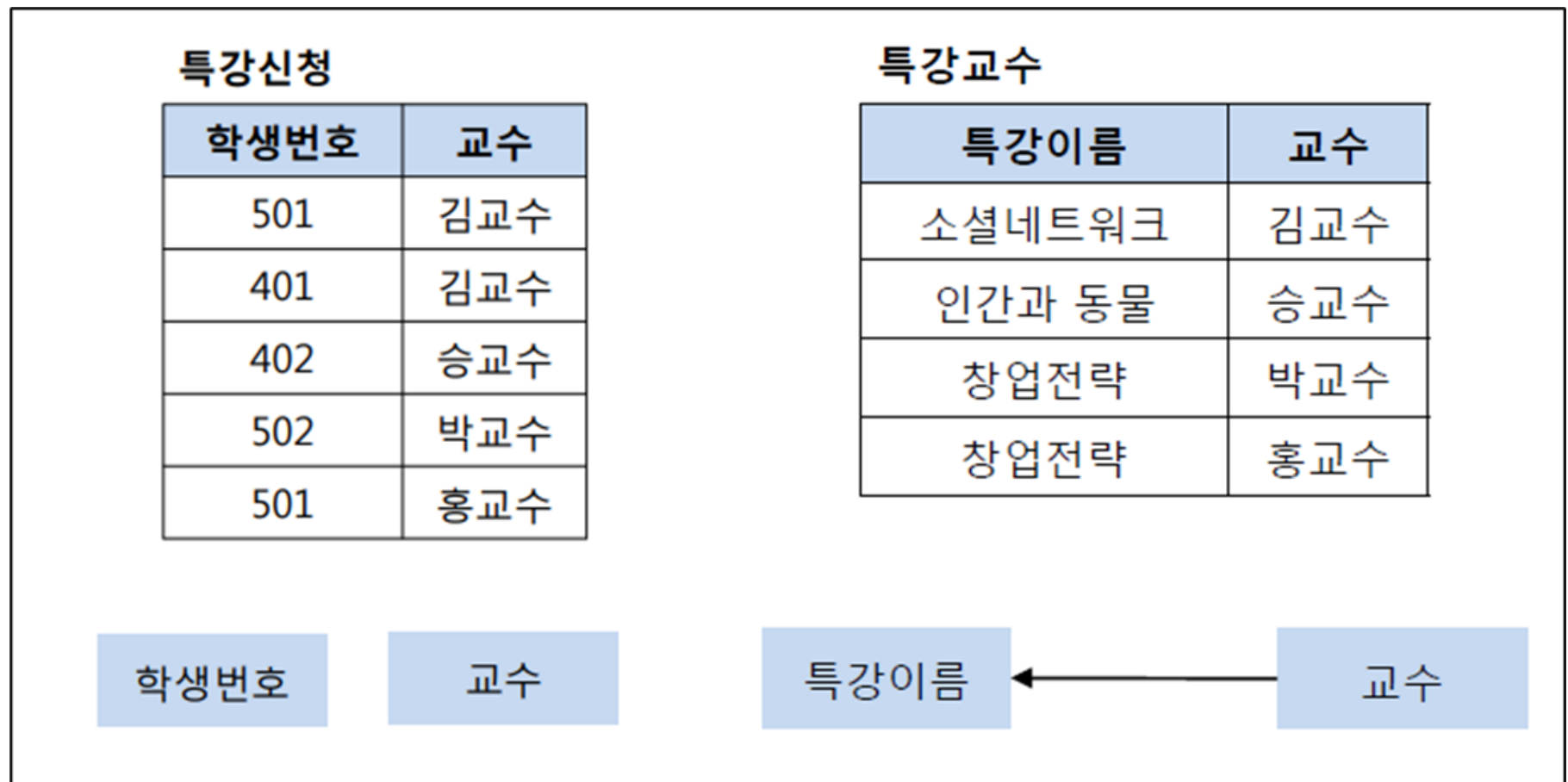




4. BCNF 정규화

- 제3 정규화를 진행한 테이블에 대해 모든 결정자가 후보키가 되도록 테이블을 분해하는 것





- **반정규화**
 - 정규화로 테이블을 쪼개면 빈번한 join이 발생해서 성능이 저하될 수 있다.
 - 시스템의 성능 향상, 개발 및 운영의 편의성 등을 위해 정규화된 데이터 모델을 통합, 중복, 분리하는 과정으로 **의도적으로 정규화 원칙을 위배하는 행위**

RDB-NoSQL 차이

	RDB	NoSQL
데이터 저장 모델	table	json / key-value / 그래프 등
개발 목적	데이터 중복 감소	애자일 / 확장가능성 / 수정가능성
schema	엄격한 데이터 구조	유연한 데이터 구조
장점	<ul style="list-style-type: none"> - 명확한 데이터 구조 보장 - 데이터 중복 없이 한 번만 저장(무결성) - 데이터 중복이 없어서 데이터 update 용이 	<ul style="list-style-type: none"> - 유연하고 자유로운 데이터 구조 - 새로운 필드 추가 자유로움 - 수평적 확장(scale-out) 용이
단점	<ul style="list-style-type: none"> - 시스템이 커지면 join문이 많은 복잡한 쿼리가 필요 - 수평적 확장이 까다로워 비용이 큰 수직적 확장(scale-up)이 주로 사용됨 - 데이터 구조가 유연하지 못함 	<ul style="list-style-type: none"> - 데이터 중복 발생 가능 - 중복 데이터가 많기 때문에 데이터 변경 시 모든 컬렉션에서 수정이 필요함 - 명확한 데이터 구조 보장 X
사용	<ul style="list-style-type: none"> - 데이터 구조가 변경될 여지 없이 명확한 경우 - 데이터 update가 잦은 시스템(중복 데이터가 없으므로 변경에 유리) 	<ul style="list-style-type: none"> - 정확한 데이터 구조가 정해지지 않은 경우 - update가 자주 이루어지지 않는 경우(조회가 많은 경우) - 데이터 양이 매우 많은 경우(scale-out 가능)

- ▼ 참고 (수평적 확장, 수직적 확장)
- 수직적 확장 : 단일 서버의 스펙을 단순히 더 좋은 것으로 업그레이드 하는 것
 - 수평적 확장 : 여러 대의 서버를 추가로 설치