DB구조 & 설계

▼ 키(key)

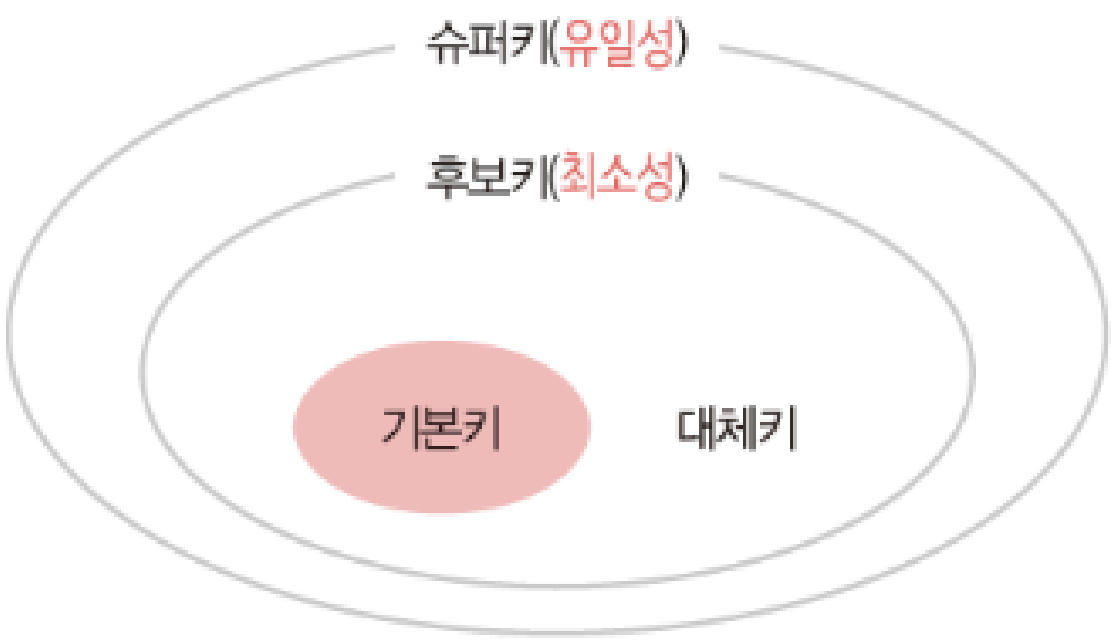


그림 5-8 키의 관계

1. 후보키 (Candidate Key)

- Tuple을 유일하게 식별하기 위해 사용하는 속성들의 부분 집합. (기본키로 사용할 수 있는 속성들)
- 2가지 조건 만족
  - 유일성 : Key로 하나의 Tuple을 유일하게 식별할 수 있음
  - 최소성 : 꼭 필요한 속성으로만 구성
- ex) (학번), (주민번호)

2. 기본키 (Primary Key)

- 후보키 중 선택된 **Main Key**
- 특징
  - Null 값을 가질 수 없음
  - 동일한 값이 중복될 수 없음
  - 오직 1개만 지정할 수 있음

3. 슈퍼키 (Super Key)

- **유일성을 만족**하지만, 최소성은 만족하지 못하는 키
- ex) (학번 + 이름) : 동명이인이 있어도 튜플 구분을 할 수 있지만(유일성 만족), 학번 하나만으로도 구분할 수 있음(최소성 만족 x)

4. 대체키 (Alternate Key)

- 후보키 중 기본키로 선택되지 않은 키 = 보조키

5. 외래키 (Foreign Key)

- 다른 테이블(Relation)의 기본키를 그대로 참조하는 속성의 집합
- 테이블 간의 관계를 나타내기 위해서 사용
- **참조 무결성**을 보장
  - 일반적으로 부모테이블의 기본키 값과 동일하므로 데이터 무결성 보장

    - 설정에 따라 NULL을 허용할 수 있음

6. 복합키 (Composite Key)

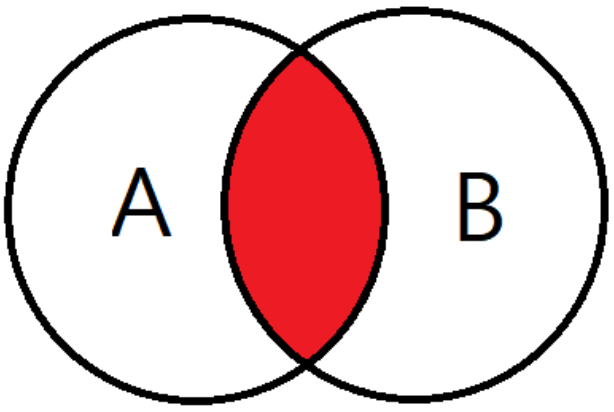
- **2개 이상**의 속성(attribute)을 사용한 키

▼ JOIN

- **두 개 이상**의 테이블이나 데이터베이스를 연결하여 **데이터를 검색**하는 방법
- 테이블을 연결하려면, 적어도 하나의 칼럼을 서로 공유하고 있어야 하므로 이를 이용하여 데이터 검색에 활용한다.

Join 종류

1. INNER JOIN (내부조인)

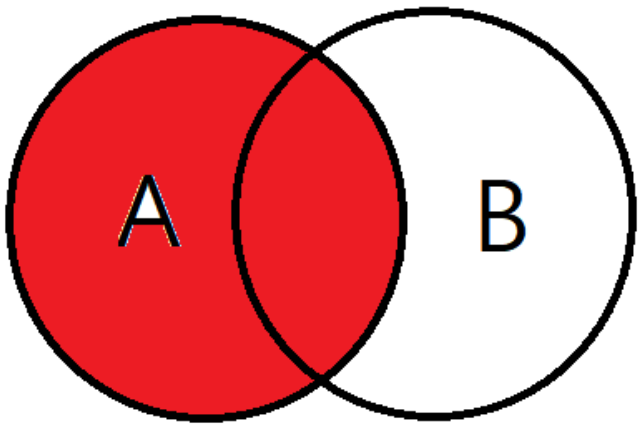


- **교집합**으로, 기준 테이블과 join 테이블의 중복된 값을 보여준다.
- 여러 애플리케이션에서 사용되는 **가장 흔한 결합 방식**이며, **기본 조인 형식**으로 간주된다.

```
# 명시적 표현
SELECT A.NAME, B.AGE
FROM EX_TABLE A
INNER JOIN (JOIN) JOIN_TABLE B ON A.NO_EMP = B.NO_EMP

#암시적 표현
SELECT A.NAME, B.AGE
FROM EX_TABLE A, JOIN_TABLE B
WHERE A.NO_EMP = B.NO_EMP
```

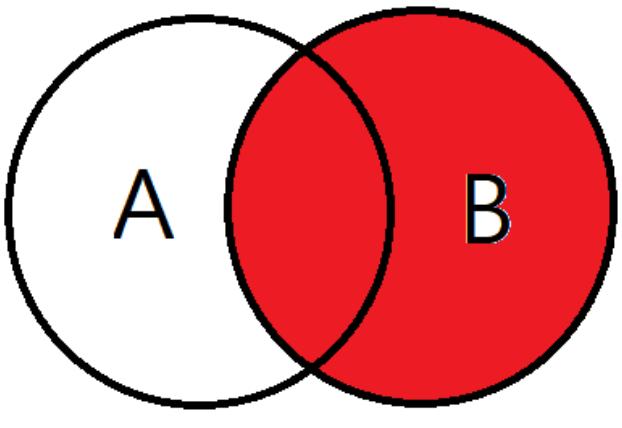
2. LEFT OUTER JOIN (왼쪽 외부 조인)



- **좌측 테이블의 모든 데이터**를 포함하는 결과 집합을 생성한다. (왼쪽테이블 기준으로 JOIN)
- 기준테이블값과 조인테이블과 중복된 값을 보여준다.

```
SELECT A.NAME, B.AGE
FROM EX_TABLE A
LEFT OUTER JOIN (LEFT JOIN) JOIN_TABLE B ON A.NO_EMP = B.NO_EMP
```

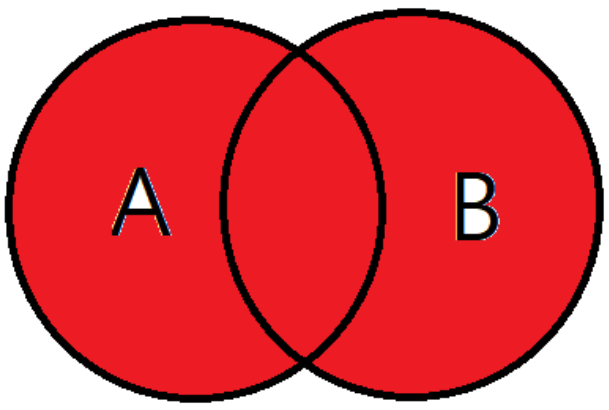
3. RIGHT OUTER JOIN (오른쪽 외부 조인)



- **우측 테이블의 모든 데이터**를 포함하는 결과 집합을 생성한다. (오른쪽 테이블 기준으로 JOIN)

```
SELECT A.NAME, B.AGE
FROM EX_TABLE A
RIGHT OUTER JOIN JOIN_TABLE B ON A.NO_EMP = B.NO_EMP
```

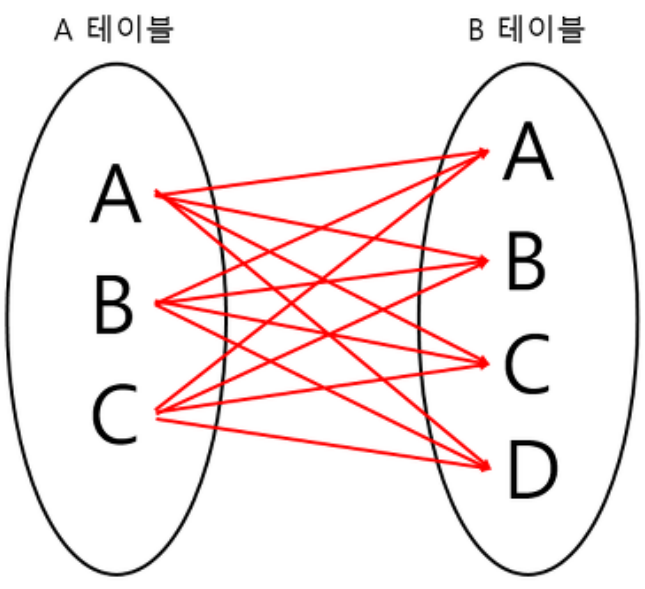
4. FULL OUTER JOIN (완전 외부 조인)



- **합집합**으로, **양 테이블의 모든 데이터**가 검색된다.

```
SELECT A.NAME, B.AGE
FROM EX_TABLE A
FULL OUTER JOIN JOIN_TABLE B ON A.NO_EMP = B.NO_EMP
```

5. CROSS JOIN (교차 조인)

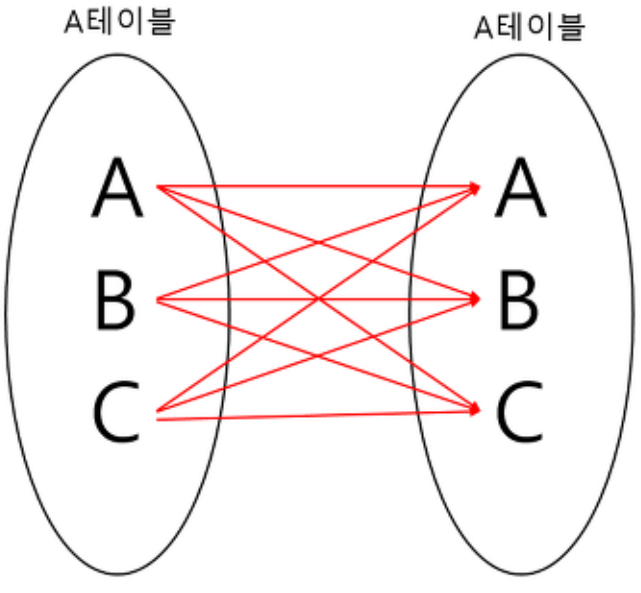


- **모든 경우의 수**를 전부 표현해주는 방식이다. (두 번째 테이블로부터 각 행과 첫 번째 테이블에서 각 행이 한번씩 결합된 열을 만듦)
- m행을 가진 테이블과 n행을 가진 테이블이 교차 조인되면 m*n 개의 행을 생성한다.

```
SELECT A.NAME, B.AGE
FROM EX_TABLE A
```

```
CROSS JOIN JOIN_TABLE B
```

6. SELF JOIN



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

```
SELECT A.NAME, B.AGE
FROM EX_TABLE A, EX_TABLE B
```

▼ 정규화 & 이상

정규화 하는 이유

- 한 릴레이션에 여러 엔티티의 애트리뷰트들을 혼합하게 되면 **정보가 중복 저장** ⇒ **저장 공간 낭비**, **이상현상** 발생
- 이를 해결하기 위해 정규화 과정

이상

1. 삽입 이상 (Insertion Anomaly)

- 불필요한 데이터를 추가해야지, 삽입할 수 있는 상황 (불필요한 정보를 함께 저장하지 않고는 어떤 정보를 저장하는 것이 불가능)

학번	학생명	학과 코드	학과명	학과장 코드	학과장명
1	도우너	101	경영학과	1000	워런 버핏
2	고길동	101	경영학과	1000	워런 버핏
3	또치	102	물리학과	2000	아인슈타인
4	마이콜	102	물리학과	2000	아인슈타인
5	둘리	103	컴퓨터공학과	3000	빌 게이츠
		104	수학과	4000	피타고라스

↳ 학생이 없으므로 추가할 수 없음.

2. 갱신 이상 (Update Anomaly)

- 반복된 데이터 중에 일부만 수정하면 데이터의 불일치가 발생 (일부만 변경하여, 데이터가 불일치 하는 모순의 문제)

이름	회원번호	스터디명	스터디번호	스터디 생성일
흰둥이	1	리팩토링	1	10월9일
신짱구 ?	1	짱구의 일기	2	1월16일
신짱구 ?	1	디우 괴롭히기	3	11월15일
김디우	2	리팩토링	1	10월9일
김디우	2	짱구의 일기	2	1월16일

- 신짱구가 흰둥이로 수정됐는데 하나만 수정되고 나머지는 수정되지 않아 데이터 모순 발생

3. 삭제 이상 (Deletion Anomaly)

- 유용한 정보를 함께 삭제하지 않고는 어떤 정보를 삭제하는 것이 불가능 (튜플 삭제로 인해 꼭 필요한 데이터까지 함께 삭제되는 문제)

학번	학생명	학과 코드	학과명	학과장 코드	학과장명
1	도우너	101	경영학과	1000	워런 버핏
2	고길동	101	경영학과	1000	워런 버핏
3	또치	102	물리학과	2000	아인슈타인
4	마이콜	102	물리학과	2000	아인슈타인
5	둘리	103	컴퓨터공학과	3000	빌 게이즈

둘리 학생이 존재하는 행을 삭제하면, 컴퓨터공학과 관련 정보도 모두 삭제되는 문제가 발생.

정규화

관계형 데이터베이스 설계 시 **중복과 이상 현상을 최소화**하기 위해 **데이터를 구조화**하는 작업

목적

- 데이터의 **중복을 없애**면서 불필요한 데이터를 최소화시킨다.
- **무결성**을 지키고, **이상 현상을 방지**한다.
- 테이블 구성을 논리적이고 직관적으로 할 수 있다.
- 데이터베이스 구조를 **확장에 용이**해진다.

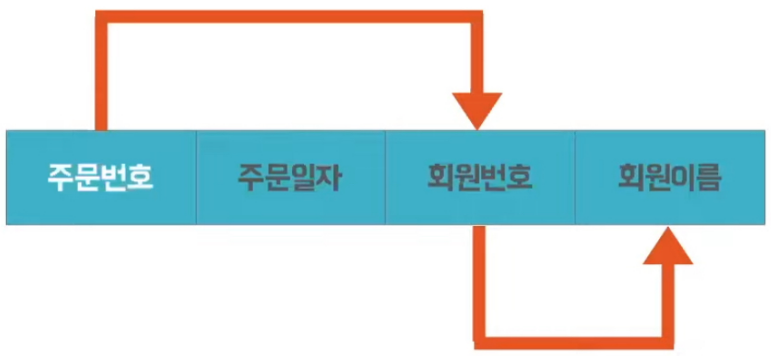
함수의 종속성

- 애트리뷰트 X값이 애트리뷰트 Y의 값을 **유일하게 결정**한다면 Y가 X에 함수적으로 종속
- **완전 함수 종속**
  - 어떤 애트리뷰트가 기본키에 대해 완전히 종속적일 때
  - 직책은 (회원번호, 스터디번호) 두개가 있어야 결정

- **부분 함수 종속**

- 어떤 애트리뷰트가 기본키에 대해 부분적으로 종속적일 때
- 이름은 (회원번호)로만 결정

- **이행 함수 종속**
  - 릴레이션에서 X, Y, Z라는 3개의 속성이 있고 X→Y, Y→Z란 종속 관계가 있을 때, X→Z가 성립하는 경우



제 1정규화(1NF)

- 테이블 컬럼이 **원자값(하나의 값)**을 갖도록 테이블을 분리시키는 것
- 만족해야 할 조건
  - 어떤 릴레이션에 속한 모든 도메인이 **원자값만**으로 되어 있어야한다.
  - 모든 속성에 **반복**되는 그룹이 나타나지 **않는다**.
  - **기본키를 사용**하여 관련 데이터의 각 집합을 **고유하게 식별**할 수 있어야 한다.

Customer

Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

- 현재 테이블은 전화번호를 여러개 가지고 있어 원자값이 아님. 따라서 1NF에 맞추기 위해서는 아래와 같이 분리

Customer

Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025
123	Pooja	Singh	192-122-1111
456	San	Zhang	182-929-2929
456	San	Zhang	(555) 403-1659 Ext. 53
789	John	Doe	555-808-9633

제 2정규화(2NF)

- **제 1정규형을 만족**하고, 테이블의 기본키가 아닌 모든 컬럼이 **기본키에 완전 함수적 종속**을 만족하도록 테이블 분리시키는 것
- 기본키가 두 개 이상의 컬럼으로 구성되었을 경우에만 제 2정규형을 만족하는가를 고려할 필요가 있다.
    - 테이블에서 기본키가 복합키(키1, 키2)로 묶여있을 때, 두 키 중 하나의 키만으로 다른 컬럼을 결정지을 수 있으면 안된다.

Electric Toothbrush Models

Manufacturer	Model	Model Full Name	Manufacturer Country
Forte	X-Prime	Forte X-Prime	Italy
Forte	Ultraclean	Forte Ultraclean	Italy
Dent-o-Fresh	EZbrush	Dent-o-Fresh EZbrush	USA
Kobayashi	ST-60	Kobayashi ST-60	Japan
Hoch	Toothmaster	Hoch Toothmaster	Germany
Hoch	X-Prime	Hoch X-Prime	Germany

- `Manufacture`과 `Model`이 키가 되어 `Model Full Name`을 알 수 있음
- `Manufacturer Country`는 `Manufacturer`로 인해 결정. (부분 함수 종속)
- 결국 완전 함수적 종속을 충족시키지 못하고 있는 테이블. 부분 함수 종속을 해결하기 위해 테이블을 아래와 같이 나눠서 2NF를 만족

Electric Toothbrush Models

Manufacturer	Model	Model Full Name
Forte	X-Prime	Forte X-Prime
Forte	Ultraclean	Forte Ultraclean
Dent-o-Fresh	EZbrush	Dent-o-Fresh EZbrush
Kobayashi	ST-60	Kobayashi ST-60
Hoch	Toothmaster	Hoch Toothmaster
Hoch	X-Prime	Hoch X-Prime

Electric Toothbrush Manufacturers

Manufacturer	Manufacturer Country
Forte	Italy
Dent-o-Fresh	USA
Kobayashi	Japan
Hoch	Germany

제 3정규화(3NF)

- **제 2정규형을 만족**하고, 기본키가 아닌 모든 컬럼이 **기본키에 이행 함수 종속되지 않도록** 테이블을 분리시키는 것
- 기본키가 아닌 속성들은 **기본키에 의존**

Tournament Winners

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

- 현재 테이블에서는 `Tournament`와 `Year`이 기본키
- `Winner`는 이 두 복합키를 통해 결정
- 하지만 `Winner Date of Birth`는 기본키가 아닌 `Winner`에 의해 결정되고 있음
- 따라서 이는 3NF를 위반하고 있으므로 아래와 같이 분리

Tournament Winners

Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson
Cleveland Open	1999	Bob Albertson
Des Moines Masters	1999	Al Fredrickson
Indiana Invitational	1999	Chip Masterson

Winner Dates of Birth

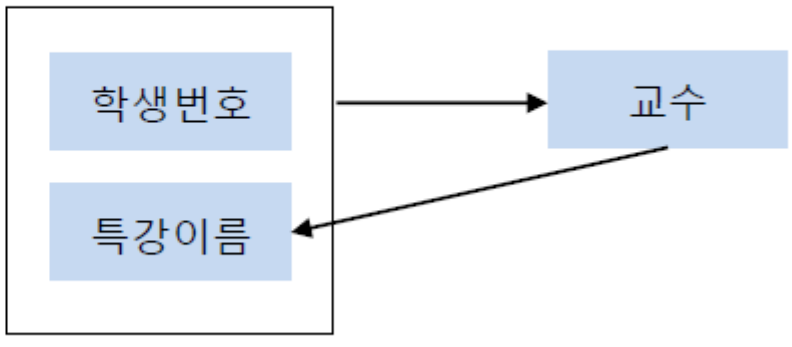
Winner	Date of Birth
Chip Masterson	14 March 1977
Al Fredrickson	21 July 1975
Bob Albertson	28 September 1968

BCNF(Boyce-Codd) 정규형

- **제3 정규화**를 진행한 테이블에 대해 **모든 결정자가 후보키가 되도록** 테이블을 분리하는 것

특강수강

학생번호	특강이름	교수
501	소셜네트워크	김교수
401	소셜네트워크	김교수
402	인간과 동물	승교수
502	창업전략	박교수
501	창업전략	홍교수



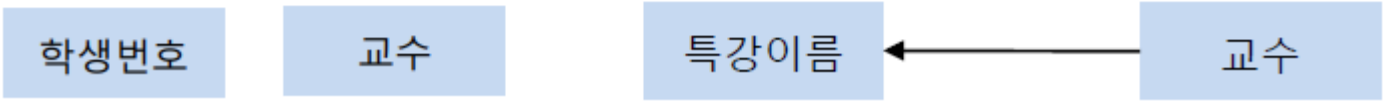
- 테이블에서 기본키는 (`학생번호` , `특강이름`)
- 기본키는 `교수` 를 결정.
- `교수` 는 `특강이름` 을 결정
- 교수가 특강이름을 결정하는 결정자이지만, 후보키가 아님

특강신청

학생번호	교수
501	김교수
401	김교수
402	승교수
502	박교수
501	홍교수

특강교수

특강이름	교수
소셜네트워크	김교수
인간과 동물	승교수
창업전략	박교수
창업전략	홍교수



- 그렇기 때문에 BCNF 정규화를 만족시키기 위해서 위의 테이블을 분해해야 하는데, 다음과 같이 특강신청 테이블과 특강교수 테이블로 분해할 수 있음

장점

- 데이터 구조의 **안정성 유지**
- 중복 값 & 널 값이 줄어듦
- 데이터베이스 **변경 시 무결성을 지키고, 이상 현상(Anomaly) 제거**
- 저장공간 최소화
- 데이터베이스 **구조 확장 시 재 디자인 최소화** (새로운 데이터 형의 추가로 인한 확장 시, 그 구조를 변경하지 않아도 되거나 일부만 변경)

단점

- 릴레이션의 분해로 인해 **릴레이션 간의 연산(JOIN 연산)이 많아짐**. 이로 인해 질의에 대한 **응답 시간이 느려질 수 있음**.

반정규화(De-normalization, 비정규화)

- 시스템의 성능 향상, 개발 및 운영의 편의성 등을 위해 정규화된 데이터 모델을 통합, 중복, 분리하는 과정으로 **의도적으로 정규화 원칙을 위배하는 행위**

▼ RDB - NoSQL

관계형 데이터베이스 (RDB)

- 사전에 엄격하게 **정의된 DB 스키마**를 요구하는 **table 기반 데이터 구조**를 갖음
- 데이터를 **테이블끼리 관계**를 통해 정의 (데이터는 관계를 통해 여러 테이블에 분산)
- **SQL**을 사용하여 데이터 조작, 관리

특징

- 고정된 스키마를 가지고 있어 데이터 구조가 미리 정의되어 있어, **스키마를 준수하지 않은 레코드는 테이블에 추가할 수 없음**
- **트랜잭션**을 사용 하므로 **ACID**한 성질을 갖음

장점

- 정해진 스키마에 따라 데이터를 저장하여야 하므로 **명확한 데이터 구조를 보장**
- 관계를 통해 각 데이터를 **중복없이** 한 번만 저장
- 복잡한 쿼리 및 JOIN 연산이 가능

단점

- 테이블 간 관계를 맺고 있어 **시스템이 커질 경우 JOIN문이 많은 복잡한 쿼리** 발생 가능
- 성능 향상을 위해서 **Scale-up만 지원.** (이로 인해 비용이 기하급수적으로 늘어날 수 있음)
- 스키마로 인해 데이터가 **유연하지 못함** (나중에 수정하기 힘듬)

RDB를 사용해야할 때

- 데이터베이스의 ACID 성질을 준수해야 하는 소프트웨어를 개발하는 경우
- 관계를 맺고 있는 데이터가 자주 변경되는 애플리케이션의 경우
- 변경될 여지가 없고 명확한 스키마가 사용자와 데이터에게 중요한 경우

NoSQL (비관계형 데이터베이스)

- 등장배경
  - 데이터가 폭발적으로 증가하면서 단일 서버에 모든 데이터를 넣을 수 없어졌음
  - 따라서, 서버의 확장이 불가피해짐
  - RDB는 단일 서버에서 돌아갈거라는 가정을 가지고 만들었기 때문에 확장에 한계
  - 이러한 RDBMS의 한계를 극복하기 위해 NoSQL 등장

특징

- **유연한 데이터 모델**을 가짐 (고정된 스키마 X)
- 테이블 간 **관계 정의 X**
- **다양한 방식으로 데이터를 표현**
  - ex) documment, key-value, graph
- 저렴한 비용
- **분산 시스템**을 사용
  - 데이터를 여러대의 컴퓨터 노드에 분산하여 저장하고 처리하는 시스템
  - **확장성 용이(Scale-out)**
  - 노드 중 하나가 다운되어도 시스템 전체가 중단되지 않도록 하여 시스템의 **가용성**을 높인다
  - 각 노드는 자기가 가진 데이터만 처리하고 여러 노드가 **병렬 처리**를 하여 **빠른 처리**가 가능하다

장점

- 스키마가 없기 때문에 **유연하며 자유로운 데이터 구조** 가능. 언제든 저장된 데이터를 조정하고 새로운 필드를 추가 가능
- 데이터는 애플리케이션이 **필요로 하는 형식**으로 저장됨. 데이터 읽어오는 **속도 빨라짐**

- 데이터 분산이 용이하며 Saclue-up과 **Scale-out 가능**해서 애플리케이션이 발생시키는 모든 읽기/쓰기 요청 처리 가능

단점

- **데이터 중복이 발생**할 수 있으며 중복된 데이터가 변경 될 경우 **수정을 모든 컬렉션에서** 수행을 해야 함
- 스키마가 존재하지 않기에 명확한 데이터 구조를 보장하지 않으며 데이터 구조 결정 어려움

NoSQL을 사용해야할 때

- 정확한 데이터의 구조를 알 수 없거나 변경, 확장될 가능성이 있는 경우
- 읽기는 자주 해도 데이터 변경은 자주 없는 경우
- 막대한 양의 데이터를 다뤄야 해서 데이터베이스를 수평으로 확장해야 하는 경우

RDB vs NoSQL

	RDB (SQL)	NoSQL
데이터 저장 모델	table	json document / key-value / 그래프 등
개발 목적	데이터 중복 감소	애자일 / 확장가능성 / 수정가능성
예시	Oracle, MySQL, PostgreSQL 등	MongoDB, DynamoDB 등
Schema	엄격한 데이터 구조	유연한 데이터 구조
⭐**장점**⭐	- 명확한 데이터구조 보장 - 데이터 중복 없이 한 번만 저장 (무결성) - 데이터 중복이 없어서 **데이터 update 용이**	- 유연하고 자유로운 데이터 구조 - 새로운 필드 추가 자유로움 - 수평적 확장(scale out) 용이
⭐**단점**⭐	- 시스템이 커지면 Join문이 많은 복잡한 query가 필요 - 수평적 확장이 까다로워 비용이 큰 수직적 확장(Scale up)이 주로 사용됨. - 데이터 구조가 유연하지 못함	- 데이터 중복 발생 가능 - **중복 데이터가 많기 때문에 데이터 변경 시 모든 컬렉션에서 수정이 필요함** - 명확한 데이터구조 보장 X
⭐**사용**⭐	- **데이터 구조가 변경될 여지가 없이 명확**한 경우 - **데이터 update가 잦은** 시스템 (중복 데이터가 없으므로 변경에 유리)	- 정확한 데이터 구조가 정해지지 않은 경우 - Update가 자주 이루어지지 않는 경우 (조회가 많은 경우) - 데이터 양이 매우 많은 경우 (scale out 가능)