

4. 데이터베이스

▼ 목차

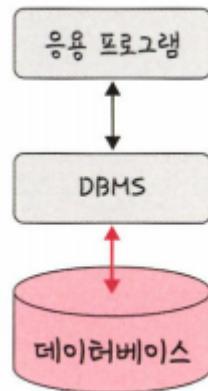
- 4-1. 데이터베이스의 기본
 - 4-1-1. 엔터티
 - 4-1-2. 릴레이션
 - 4-1-3. 속성
 - 4-1-4. 도메인
 - 4-1-5. 필드와 레코드
 - 4-1-6. 관계
 - 4-1-7. 키
 - 4-2. ERD와 정규화 과정
 - 4-2-1. ERD의 중요성
 - 4-2-2. 예제로 배우는 ERD
 - 4-2-3. 정규화 과정
 - 4-3. 트랜잭션과 무결성
 - 4-3-1. 트랜잭션
 - 4-3-2. 무결성
 - 4-4. 데이터베이스의 종류
 - 4-4-1. 관계형 데이터베이스
 - 4-4-2. NoSQL 데이터베이스
 - 4-5. 인덱스
 - 4-5-1. 인덱스의 필요성
 - 4-5-2. B-트리
 - 4-5-3. 인덱스 만드는 방법
 - 4-5-4. 인덱스 최적화 기법
 - 4-6. 조인의 종류
 - 4-6-1. 내부 조인
 - 4-6-2. 왼쪽 조인
 - 4-6-3. 오른쪽 조인
 - 4-6-4. 합집합 조인
 - 4-7. 조인의 원리
 - 4-7-1. 중첩 루프 조인
 - 4-7-2. 정렬 병합 조인
 - 4-7-3. 해시 조인
-

4-1. 데이터베이스의 기본

데이터베이스는 일정한 규칙 혹은 규약을 통해 구조화되어 저장되는 데이터의 모음.

해당 데이터베이스를 제어, 관리하는 통합 시스템을 DBMS(DataBase Management System)

데이터베이스 안에 있는 데이터들은 특정 DBMS마다 정의된 쿼리 언어를 통해 삽입, 삭제, 수정, 조회 등을 수행할 수 있다. 또한 데이터베이스는 실시간 접근과 동시 공유가 가능하다.



그림처럼 데이터베이스 위에 DBMS 가 있고

그 위에 응용 프로그램이 있으며 이러한 구조를 기반으로 데이터를 주고 받는다.

예를 들어 MySQL이라는 DBMS가 있고 그 위에 응용 프로그램에 속하는 node.js나 php에 해당 데이터베이스 안에 있는 데이터를 끄집어내 해당 데이터 관련 로직을 구축할 수 있는 것이다.

4-1-1. 엔터티

엔터티는 사람 장소 물건 사건 개념 등 여러 개의 속성을 지닌 명사를 의미

예를 들어 회원이라는 엔터티가 있다고 한다면 회원 이름 아이디 주소 전화번호 같은 속성이 있는 것이다.

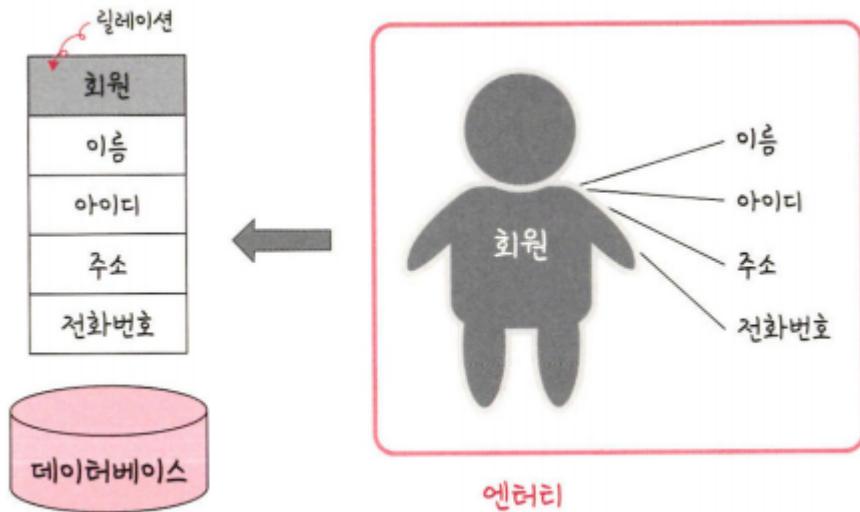
약한 엔터티와 강한 엔터티

A가 혼자서는 존재하지 못하고 B의 존재 여부에 따라 종속적이라면 A는 약한 엔터티이고 B는 강한 엔터티가 된다.

4-1-2. 릴레이션

데이터베이스에서 정보를 구분하여 저장하는 기본 단위

엔터티에 관한 데이터를 데이터베이스는 릴레이션 하나에 담아서 관리



그림처럼 회원이라는 엔터티가 데이터베이스에서 관리될 때 릴레이션으로 변화된 것을 볼 수 있다.

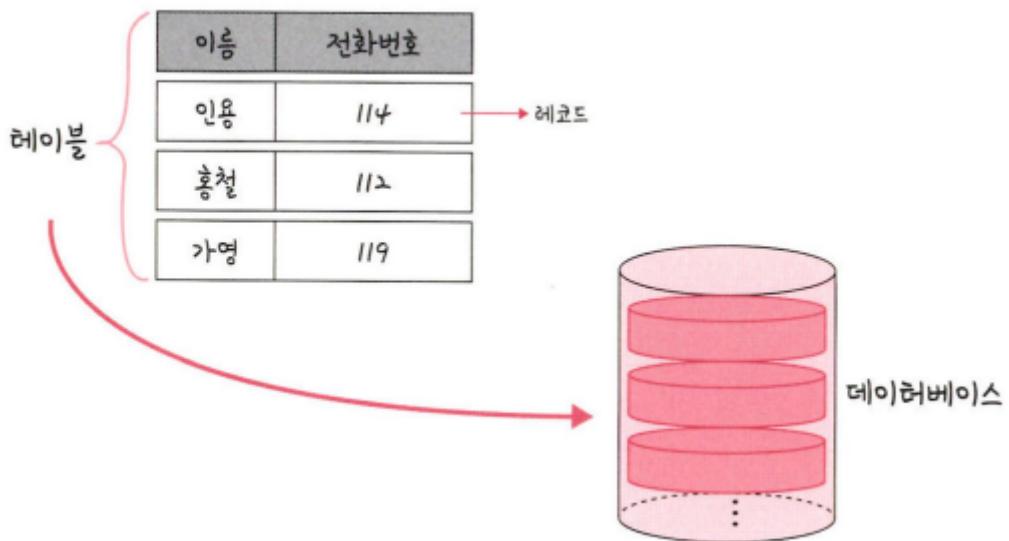
릴레이션은 관계형 데이터베이스에서는 테이블이라고 하며, NoSQL 데이터베이스는 컬렉션이라고 한다

테이블과 컬렉션

데이터베이스의 종류는 크게 관계형 데이터베이스와 NoSQL 비관계형 데이터베이스로 나눔

이 중 대표적인 관계형 데이터베이스인 MySQL과
대표적인 NoSQL 데이터베이스인 MongoDB를 예로 들면,

MySQL의 구조는 레코드-테이블-데이터베이스로 이루어져 있고
NoSQL 데이터베이스의 구조는 도큐먼트-컬렉션-데이터베이스로 이루어져 있다.



그림처럼 레코드가 쌓여서 테이블이 되고 테이블이 쌓여서 데이터베이스가 되는 것이다.

4-1-3. 속성

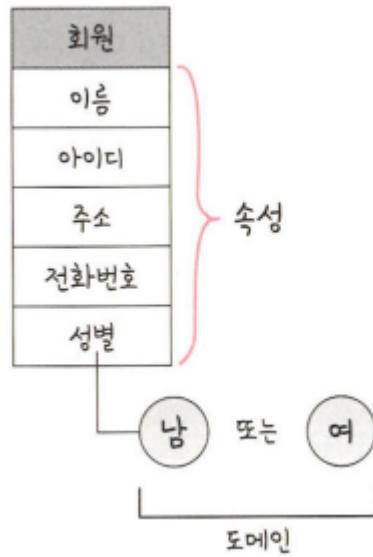
속성은 릴레이션에서 관리하는 구체적이며 고유한 이름을 갖는 정보

예를 들어 '차'라는 엔터티의 속성을 뽑아보자. 차 넘버, 바퀴 수, 차 색깔, 차종 등이 있겠다. 이 중에서 서비스의 요구 사항을 기반으로 관리해야 할 필요가 있는 속성들만 엔터티의 속성이 된다.

4-1-4. 도메인

릴레이션에 포함된 각각의 속성들이 가질 수 있는 값의 집합을 말한다

예를 들어 성별이라는 속성이 있다면 이 속성이 가질 수 있는 값은 (남, 여)라는 집합이 된다



그림처럼 회원이라는 릴레이션에 이름 아이디 주소 전화번호 성별이라는 속성이 있고 성별은 (남, 여)라는 도메인을 가지는 것을 알 수 있다.

4-1-5. 필드와 레코드

앞에서 설명한 것들을 기반으로 데이터베이스에서 필드와 레코드로 구성된 테이블을 만들 수 있다

member			
name	ID	address	phonenumber
큰돌	kundol	서울	112
가영	key	대전	114
빅뱅	b1g	카이루	119
:	:	:	:

Annotations: The 'phonenumber' column is highlighted with a red border and has an arrow pointing to it labeled '필드'. Two rows ('큰돌' and '가영') are also highlighted with a red border and have arrows pointing to them labeled '레코드'.

회원이란 엔터티는 member라는 테이블로 속성인 이름, 아이디 등을 가지고 있으며 name id address 등의 필드를 가진다. 그리고 이 테이블에 쌓이는 행 단위의 데이터 레코드라고 한다. 또한 레코드를 튜플이라고도 한다.

필드 타입

필드는 타입을 갖는다. 예를 들어 이름은 문자열이고 전화번호는 숫자다. 이러한 타입들은 DBMS마다 다르다. 아래는 MySQL 을 기준으로 설명한 것이다.

숫자 타입

숫자 타입으로는 TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT 등이 있습니다.

▼ 표 4-1 MySQL 숫자 타입

타입	용량(바이트)	최솟값(부호 있음)	최솟값(부호 없음)	최댓값(부호 없음)	최댓값(부호 있음)
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-263	0	263-1	264-1

날짜 타입

날짜 타입으로는 DATE, DATETIME, TIMESTAMP 등이 있습니다.

DATE

날짜 부분은 있지만 시간 부분은 없는 값에 사용됩니다. 지원되는 범위는 1000-01-01~9999-12-31입니다. 3바이트의 용량을 가집니다.

DATETIME

날짜 및 시간 부분을 모두 포함하는 값에 사용됩니다. 지원되는 범위는 1000-01-01 00:00:00에서 9999-12-31 23:59:59입니다. 8바이트의 용량을 가집니다.

TIMESTAMP

날짜 및 시간 부분을 모두 포함하는 값에 사용됩니다. 1970-01-01 00:00:01에서 2038-01-19 03:14:07까지 지원합니다. 4바이트의 용량을 가집니다.

문자 타입

문자 타입으로는 CHAR, VARCHAR, TEXT, BLOB, ENUM, SET이 있습니다.

CHAR와 VARCHAR

CHAR 또는 VARCHAR 모두 그 안에 수를 입력해서 몇 자까지 입력할지 정합니다. 예를 들어 CHAR(30)이라면 최대 30글자까지 입력할 수 있습니다.

CHAR는 테이블을 생성할 때 선언한 길이로 고정되며 길이는 0에서 255 사이의 값을 가집니다. 레코드를 저장할 때 무조건 선언한 길이 값으로 '고정'해서 저장됩니다.

VARCHAR는 가변 길이 문자열입니다. 길이는 0에서 65,535 사이의 값으로 지정할 수 있으며, 입력된 데이터에 따라 용량을 가변시켜 저장합니다. 예를 들어 10글자의 이메일을 저장할 경우 10글자에 해당하는 바이트 + 길이기록용 1바이트로 저장하게 됩니다. VARCHAR(10000)으로 선언했음에도 말이죠.

그렇기 때문에 지정된 형태에 따라 저장된 CHAR의 경우 검색에 유리하며, 검색을 별로 하지 않고 유동적인 길이를 가진 데이터는 VARCHAR로 저장하는 것이 좋습니다.

TEXT와 BLOB

두 개의 타입 모두 큰 데이터를 저장할 때 쓰는 타입입니다.

TEXT은 큰 문자열 저장에 쓰며 주로 게시판의 본문을 저장할 때 씁니다.

BLOB은 이미지, 동영상 등 큰 데이터 저장에 씁니다. 그러나 보통은 아마존의 이미지 호스팅 서비스인 S3를 이용하는 등 서버에 파일을 올리고 파일에 관한 경로를 VARCHAR로 저장합니다.

▼ 그림 4-8 아마존의 S3



ENUM과 SET

ENUM과 SET 모두 문자열을 열거한 타입입니다.

ENUM은 ENUM('x-small', 'small', 'medium', 'large', 'x-large') 형태로 쓰이며, 이 중에서 하나만 선택하는 단일 선택만 가능하고 ENUM 리스트에 없는 잘못된 값을 삽입하면 빈 문자열이 대신 삽입됩니다. ENUM을 이용하면 x-small 등이 0, 1 등으로 매핑되어 메모리를 적게 사용하는 이점을 얻습니다. ENUM은 최대 65,535개의 요소들을 넣을 수 있습니다.

SET은 ENUM과 비슷하지만 여러 개의 데이터를 선택할 수 있고 비트 단위의 연산을 할 수 있으며 최대 64개의 요소를 집어넣을 수 있다는 점이 다릅니다.

참고로 ENUM이나 SET을 쓸 경우 공간적으로 이점을 볼 수 있지만 애플리케이션의 수 정에 따라 데이터베이스의 ENUM이나 SET에서 정의한 목록을 수정해야 한다는 단점이 있습니다.

4-1-6. 관계

데이터베이스에 테이블은 하나만 있는 것이 아니다

여러 개의 테이블이 있고 이러한 테이블은 서로의 관계가 정의되어 있다. 이러한 관계를 관계화살표로 나타낸다

하나의 A는 하나의 B로 구성되어 있다:



하나의 A는 하나 이상의 B로 구성되어 있다:



하나의 A는 하나 이하의 B로 구성되어 있다:

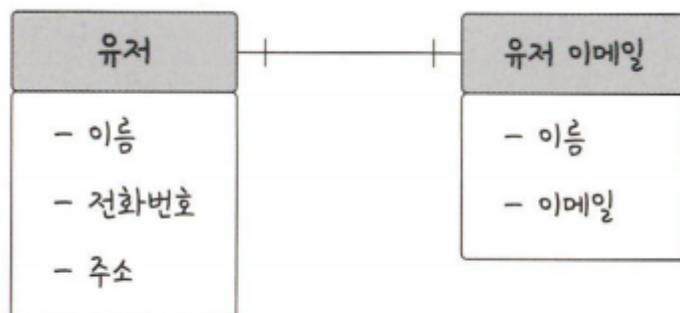


하나의 A는 0 또는 하나 이상의 B로 구성되어 있다:



1:1 관계

예를 들어 유저당 유저 이메일은 한 개씩 있을 것. 이 경우 1:1 관계가 된다

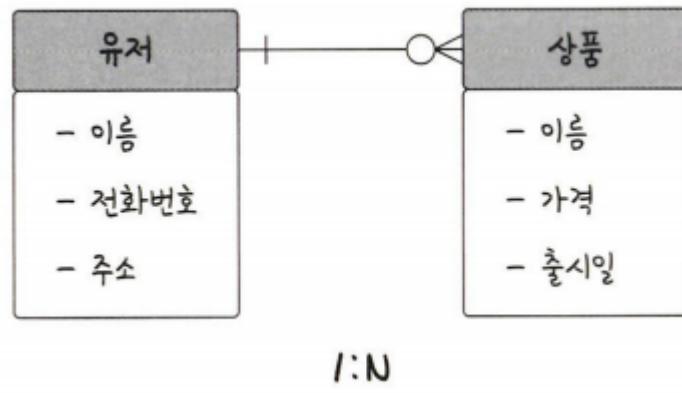


1:1

1:1 관계는 테이블을 두 개의 테이블로 나눠 테이블의 구조를 더 이해하기 쉽게 만들어 준다

1:N 관계

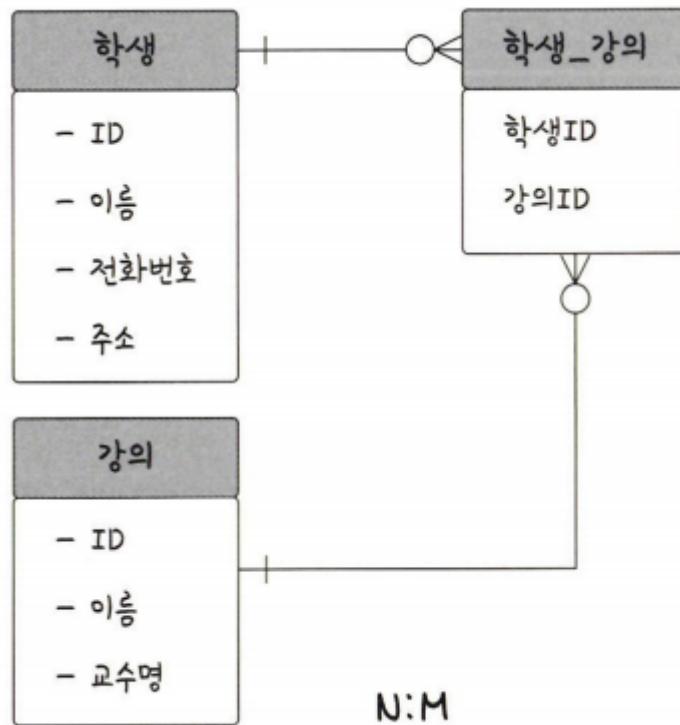
쇼핑몰을 운영한다고 하면 한 유저당 여러 개의 상품을 장바구니에 넣을 수 있다
이 경우가 1:N 관계가 된다. 한 개체가 다른 많은 개체를 포함하는 관계를 말한다.



1:N

N:M 관계

학생과 강의의 관계를 정의한다면 학생도 강의를 많이 들을 수 있고 강의도 여러 명의 학생을 포함할 수 있다. 이러한 관계가 N:M 관계이다

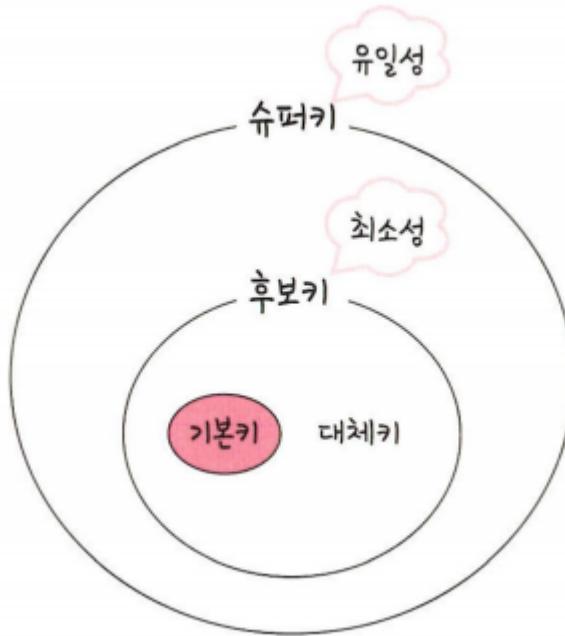


N:M

중간에 학생_강의 라는 테이블이 끼어 있다. N:M은 테이블 두 개를 직접적으로 연결해서 구축하지 않고 1:N, 1:M 이라는 관계를 갖는 테이블 두 개로 나눠서 설정한다

4-1-7. 키

테이블 간의 관계를 조금 더 명확하게 하고 테이블 자체의 인덱스를 위해 설정된 장치로 기본키, 외래키, 후보키, 슈퍼키, 대체키가 있다



키들은 앞의 그림과 같은 관계를 가진다.

슈퍼키는 유일성이 있고, 그 안에 포함된 후보 키는 최소성까지 갖춘 키이다. 후보키 중에서 기본키로 선택되지 못한 키는 대체키가 된다. 유일성은 중복되는 값은 없으며, 최소성은 필드를 조합하지 않고 최소 필드만 써서 키를 형성할 수 있는 것을 말한다.

기본키

기본키는 줄여 PK 또는 프라이머리키라고 많이 부르며, 유일성과 최소성을 만족하는 키이다
기본적으로 중복되는 값은 기본키 값이 될 수 없다

ID	name
1	주홍철
2	주홍철
3	최범석
4	양기영

아이디와 이름이라는 복합키를 기본키로 설정할 수 있지만 그렇게 되면 최소성을 만족하지 않는다

기본키는 자연키 또는 인조키 중에 골라 설정한다.

자연키

유저 테이블을 만든하고 가정하면 주민번호, 이름, 성별 등의 속성이 있다. 이 중 이름, 성별 등은 중복된 값이 들어올 수 있으므로 부적절하고 남는 것은 주민번호이다. 이런 식으로 중복된 값을 제외하며 중복되지 않는 것을 자연스레 뽑다가 나오는 키를 자연키라고 한다. 자연키는 언젠가는 변하는 속성을 가진다.

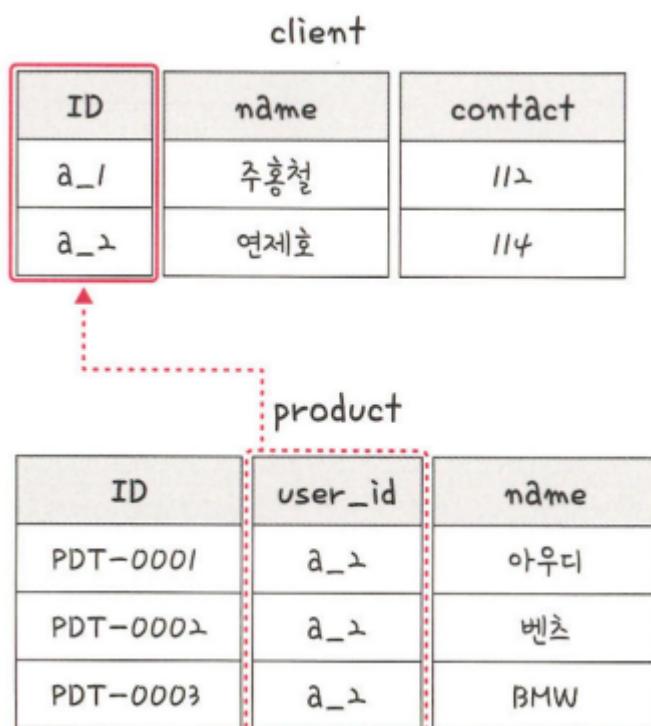
인조키

유저 테이블을 만든다고 했을 때 회원 테이블을 생성한다고 가정하면 주민 번호 이름 성별 등의 속성이 있다. 여기에 인위적으로 유저 아이디를 부여한다. 이를 통해 고유 식별자가 생겨난다. 오라클은 sequence mysql은 auto increment 등으로 설정한다. 이렇게 인위적으로 생성한 키를 인조키라고 한다.

자연키와는 대조적으로 변하지 않는다. 따라서 보통 기본키는 인조키로 설정한다.

외래키

외래키는 FK라고도 하며, 다른 테이블의 기본키를 그대로 참조하는 값으로 개체와의 관계를 식별하는데 사용한다.



외래키는 중복되어도 괜찮다. 앞의 그림을 보면 client라는 테이블의 기본키인 id가 product라는 테이블의 user_id라는 외래키로 설정될 수 있음을 보여준다. 또한 user_id는 a_2라는 값이 중복되는 것을 볼 수 있다.

후보키

기본키가 될 수 있는 후보들이며 유일성과 최소성을 동시에 만족하는 키

대체키

후보키가 두 개 이상일 경우 어느 하나를 기본키로 지정하고 남은 후보키들을 말한다

슈퍼키

각 레코드를 유일하게 식별할 수 있는 유일성을 갖춘 키

4-2. ERD와 정규화 과정

ERD (Entity Relationship Diagram)는 데이터베이스를 구축할 때 가장 기초적인 뼈대 역할을 하며, 릴레이션 간의 관계들을 정의한 것이다.

만약 서비스를 구축한다면 가장 먼저 신경 써야 할 부분이며 이부분을 신경 쓰지 않고 서비스를 구축한다면 단단하지 않은 골조로 건물을 짓는 것이나 다름 없다

4-2-1. ERD의 중요성

ERD는 시스템의 요구 상황을 기반으로 작성되며 이 ERD를 기반으로 데이터베이스를 구축한다.

데이터베이스를 구축한 이후에도 디버깅 또는 비즈니스 프로세스 재설계가 필요한 경우에 설계도 역할을 담당하기도 한다.

하지만 ERD는 관계형 구조로 표현할 수 있는 데이터를 구성하는 데 유용할 수 있지만 **비정형 데이터를 충분히 표현할 수 없다는 단점**이 있다

- **비정형 데이터**

비구조화 데이터를 말하며, 미리 정의된 데이터 모델이 없거나 미리 정의된 방식으로 정리되지 않은 정보를 말한다.

4-2-2. 예제로 배우는 ERD

다음은 예제의 서비스 요구 사항과 답을 기반으로 ERD를 작성하며 공부해 보자
참고로 정답 ERD의 테이블 필드 타입은 생략

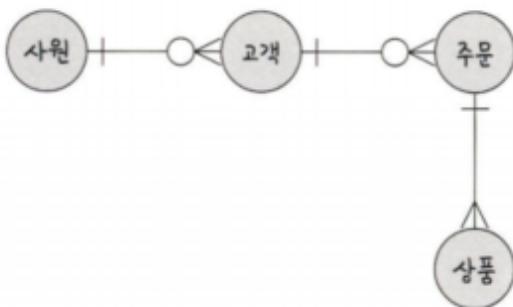
승원 영업부서의 ERD

요구 사항

- 영업사원은 0~n명의 고객을 관리한다.
- 고객은 0~n개의 주문을 넣을 수 있다.
- 주문에는 1~n개의 상품이 들어간다.

정답

▼ 그림 4-17 승원 영업부서의 ERD



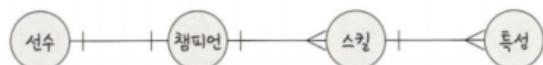
무무오브레전드의 ERD

요구 사항

- 선수들은 1명의 챔피언을 고를 수 있다.
- 챔피언은 한 개 이상의 스킬을 갖는다.
- 스킬은 한 개 이상의 특성을 갖는다.

정답

▼ 그림 4-18 무무오브레전드의 ERD



4-2-3. 정규화 과정

정규화 과정은 릴레이션 간의 잘못된 종속 관계로 인해 데이터베이스 이상 현상이 일어나서 이를 해결하거나, 저장 공간을 효율적으로 사용하기 위해 릴레이션을 여러 개로 분리하는 과정

데이터베이스 이상 현상이란 회원이 한 개의 등급을 가져야 하는데 세 개의 등급을 갖거나 삭제할 때 필요한 데이터가 같이 삭제되고, 데이터를 삽입해야 하는데 하나의 필드 값이 NULL이 되면 안 되어서 삽입하기 어려운 현상을 말한다.

정규화 과정은 정규형 원칙을 기반으로 정규형을 만들어가는 과정이며 정규화된 정도는 정규형으로 표현한다. 기본 정규형인 제1정규형 제2정규형 제3정규형. 보이스/코드 정규형이 있고 고급 정규형인 제4정규형 제5정규형이 있다. 이 중 기본 정규형인 제 123 정규형 보이스/코드 정규형을 알아보자

정규형 원칙

정규형의 원칙이란 같은 의미를 표현하는 릴레이션이지만 좀 더 좋은 구조로 만들어야 하고 자료의 중복성은 감소해야 하고, 독립적인 관계는 별개의 릴레이션으로 표현해야 하며 각각의 릴레이션은 독립적인 표현이 가능해야 하는 것을 말한다.

제 1 정규형

릴레이션의 모든 도메인이 더 이상 분해될 수 없는 원자 값만으로 구성되어야 한다.
릴레이션의 속성 값 중에서 한 개의 기본키에 대해 두 개 이상의 값을 가지는 반복 집합이 있어서는 안된다. 만약에 반복 집합이 있다면 제거해야 한다.

유저번호	유저ID	수강명	성취도
1	홍철	{C++코딩테스트, 프런트특강}	{90%, 10%}
2	범석	{코드포스특강, DS특강}	{7%, 8%}



유저번호	유저ID	수강명	성취도
1	홍철	C++코딩테스트	90%
1	홍철	프런트특강	10%
2	범석	코드포스특강	7%
2	범석	DS특강	8%

그림처럼 홍철이라는 id에 수강명이 {C++코딩테스트, 프런트특강} 이었는데 이것을 나눠서 반복 집합을 제거하는 것을 볼 수 있다.

제 2 정규형

릴레이션이 제 1 정규형이며 부분 함수의 종속성을 제거한 형태를 말한다.

부분 함수의 종속성 제거란 기본키가 아닌 모든 속성이 기본키에 완전 함수 종속적인 것을 말한다.

유저번호	유저ID	수강명	성취도
1	홍철	C++코딩테스트	90%
1	홍철	프런트특강	10%
2	범석	코드포스특강	7%
2	범석	DS특강	8%



유저번호	유저ID	유저ID	수강명	성취도
1	홍철	홍철	C++코딩테스트	90%
2	범석	홍철	프런트특강	10%
		범석	코드포스특강	7%
		범석	DS특강	8%

앞의 그림을 보면 기본키인 {유저ID, 수강명}과 완전 종속된 유저번호 릴레이션과 '{유저ID, 수강명}에 따른 성취도' 릴레이션으로 분리된 것을 볼 수 있습니다.

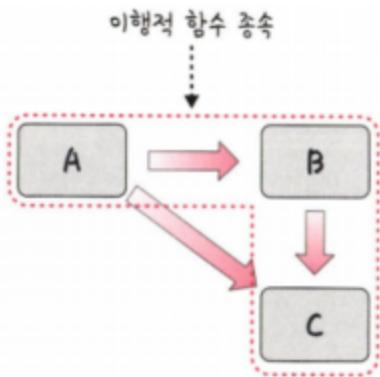
이때 주의할 점은 릴레이션을 분해할 때 동등한 릴레이션으로 분해해야 하고, 정보 손실이 발생하지 않는 무손실 분해로 분해되어야 한다는 것입니다.

제 3 정규형

제 2 정규형이고 기본키가 아닌 모든 속성이 이행적 함수 종속(transitive FD)을 만족하지 않는 상태를 말한다

이행적 함수 종속

이행적 함수 종속이란 $A \rightarrow B$ 와 $B \rightarrow C$ 가 존재하면 논리적으로 $A \rightarrow C$ 가 성립하는데, 이때 집합 C가 집합 A에 이행적으로 함수 종속이 되었다고 한다



예를 들어 무무쇼핑몰이 있다고 해봅시다. 유저ID와 등급, 할인율이 정해져 있는 테이블을 다음과 같이 분해하는 것을 말합니다.

▼ 그림 4-22 제3정규형

유저ID	등급	할인율
홍철	플래티넘	30%
범수	다이아	50%
가영	마스터	70%

↓

유저ID	등급
홍철	플래티넘
범수	다이아
가영	마스터

등급	할인율
플래티넘	30%
다이아	50%
마스터	70%

보이스/코드 정규형

보이스/코드 정규형(BCNF)은 제 3 정규형이고, 결정자가 후보키가 아닌 함수 종속 관계를 제거하여 릴레이션의 함수 종속 관계에서 모든 결정자가 후보키인 상태를 말한다.

- 결정자

함수 종속 관계에서 특정 종속자를 결정짓는 요서 $X \rightarrow Y$ 일 때 X는 결정자 Y는 종속자

요구 사항은 다음과 같다고 해봅시다.

- 각 수강명에 대해 한 학번은 오직 한 강사의 강의만 수강한다.
- 각 강사는 한 수강명만 담당한다.
- 한 수강명은 여러 강사가 담당할 수 있다.

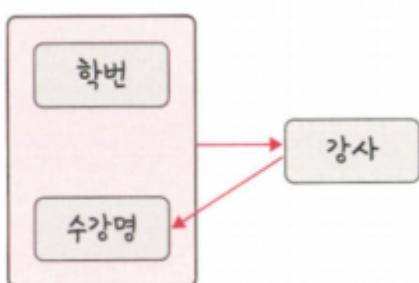
▼ 그림 4-23 학번-수강명-강사 릴레이션

학번	수강명	강사
I2010	코딩테스트	큰돌
I2010	MEVN	재엽
I2011	코딩테스트	큰돌
I2011	MEVN	가영
NULL	룰	범석



앞의 릴레이션을 보면 {학번, 수강명} 또는 {학번, 강사}가 후보키가 되며, 만약 범석이라는 강사가 '룰'이라는 수강명을 담당한다고 했을 때 이를 삽입하면 학번이 NULL이 되는 문제점이 발생합니다. 또한, 이 릴레이션은 다음과 같은 함수 종속 다이어그램을 가집니다.

▼ 그림 4-24 학번-강사-수강명 함수 종속



즉, 강사 속성이 결정자이지만 후보키가 아니므로 이 강사 속성을 분리해야 합니다.

▼ 그림 4-25 보이스/코드 정규형을 만족한 릴레이션



앞의 그림처럼 룰-범석이 제대로 들어갔으며 학번-강사/수강명-강사로 잘 분해된 모습을 볼 수 있죠?

참고로 이렇게 정규형 과정을 거쳐 테이블을 나눈다고 해서 성능이 100% 좋아지는 것은 아닙니다. 성능이 좋아질 수도 나빠질 수도 있습니다. 테이블을 나누게 되면 어떠한 쿼리를 조인을 해야 하는 경우도 발생해서 오히려 느려질 수도 있기 때문에 서비스에 따라 정규화 또는 비정규화 과정을 진행해야 합니다.

4-3. 트랜잭션과 무결성

4-3-1. 트랜잭션

트랜잭션은 데이터베이스에서 하나의 논리적 기능을 수행하기 위한 작업의 단위 데이터베이스에 접근하는 방법은 쿼리이므로, 즉 여러 개의 쿼리들을 하나로 묶는 단위를 말한다.

이에 대한 특징은 원자성, 일관성, 독립성, 지속성이 있으며 이를 한꺼번에 ACID 특징이라고 한다

원자성 (all or nothing)

트랜잭션과 관련된 일이 모두 수행되었거나 되지 않았거나를 보장하는 특징이다.

예를 들어 트랜잭션을 커밋했는데, 문제가 발생하여 롤백하는 경우 그 이후에 모두 수행되지 않음을 보장하는 것

만약 100을 가진 a가 0을 가진 b에게 50을 준다고 하면

- a의 잔고 조회
- a에게 50 빼기
- b에게 50 추가

이러한 과정을 거치지만 a b 가 각각 50씩 갖게 되는 결과만 확인할 수 있다

그리고 만약 이 과정을 취소하고 싶다면 모든 과정을 취소해야 한다. 그렇게 되면 a는 다시 100 b는 0을 가지게 되어야 하는데 a 50 b 0 이렇게 일부만 되돌려서는 안된다 → all or nothing

트랜잭션 단위로 여러 로직들을 묶을 때 외부 API를 호출하는 것이 있으면 안된다

만약 있다면 롤백이 일어났을 때 어떻게 해야 할 것인지에 대한 해결 방법이 있어야 하고 트랜잭션 전파를 신경 써서 관리해야 한다

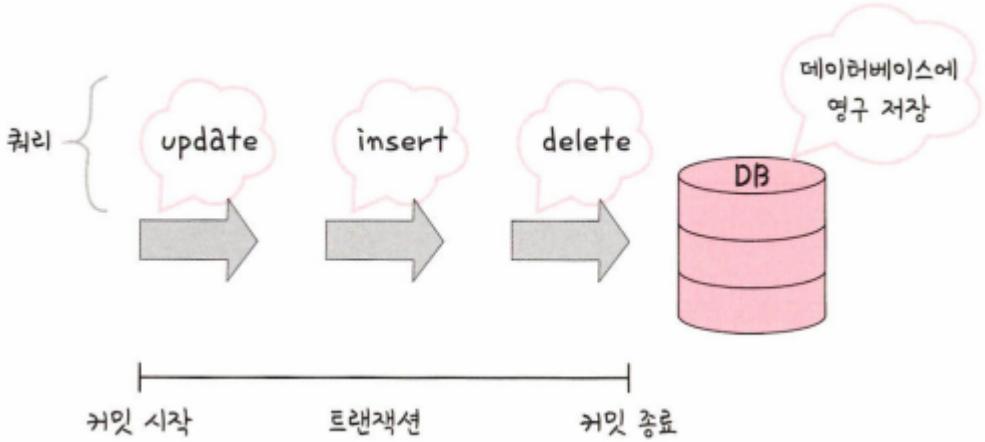
커밋

여러 쿼리가 성공적으로 처리되었다고 확정하는 명령어

트랜잭션 단위로 수행되며 변경된 내용이 모두 영구적으로 저장되는 것을 말한다

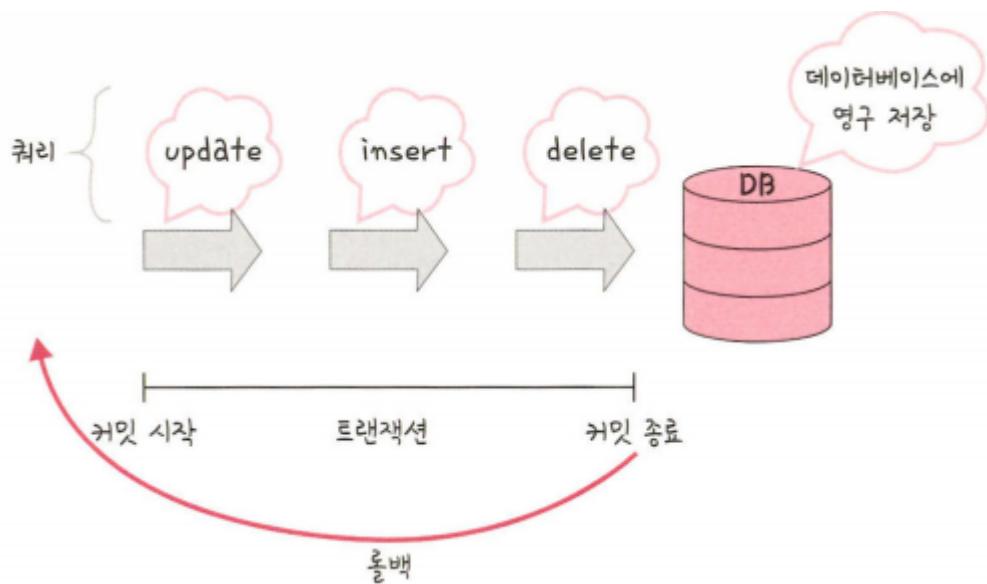
커밋이 수행되었다를 하나의 트랜잭션이 성공적으로 수행되었다 라고 말하기도 한다

커밋



그림처럼 update insert delete의 쿼리가 하나의 트랜잭션 단위로 수행되고 이후에 데이터베이스에 영구 저장된다

롤백



하지만 예러나 여수 때문에 트랜잭션 전으로 돌려야 한다면 롤백을 사용한다

롤백은 [트랜잭션으로 처리한 하나의 묶음 과정을 이어나기 전으로 돌리는 취소](#)를 말함

이러한 커밋과 롤백 덕에 데이터의 무결성이 보장된다. 또 데이터 변경 전에 변경 사항을 쉽게 확인할 수 있고 해당 작업을 그룹화할 수 있다

트랜잭션 전파

트랜잭션을 수행할 때 컨넥션 단위로 수행하기 때문에 컨넥션 객체를 넘겨서 수행해야 한다
하지만 이를 매번 넘겨주기가 어렵기도 하고 귀찮기도 하다.

이를 넘겨서 수행하지 않고 여러 트랜잭션 관련 메서드의 호출을 하나의 트랜잭션에 묶어도록 하는 것을 트랜잭션 전파라고 한다.

일관성

허용된 방식으로만 데이터를 변경 해야하는 것을 의미

데이터베이스에 기록된 모든 데이터는 여러가지 조건 규칙에 따라 유효함을 가져야 한다

a가 0원을 가지고 있는데 타인에게 500원을 입금하는 것을 불가능한 것처럼

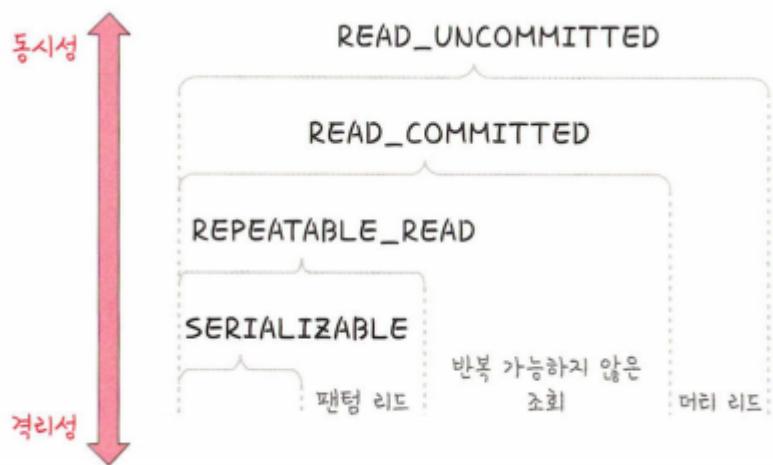
격리성

트랜잭션 수행 시 서로 끼어들지 못하는 것을 말한다

복수의 병렬 트랜잭션을 서로 격리되어 마치 순차적으로 실행되는 것처럼 작동되어야 하고, 데이터베이스는 여러 사용자가 같은 데이터에 접근할 수 있어야 한다

그냥 순차적으로 하면 쉽게 되겠지만 그렇게 하면 성능은 좋지 않을 것이다

격리성은 여러 개의 격리 수준으로 나뉘어 격리성을 보장한다



위로 갈수록 동시성이 강해지지만 격리성은 약해지고, 아래로 갈수록 동시성은 약해지고 격리성은 강해진다.

그리고 각각의 단계마다 나타나는 현상이 있다. (팬텀리드, 반복 가능하지 않은 조회, 더티리드)

팬텀리드

한 트랜잭션 내에서 동일한 쿼리를 보냈을 때 해당 조회 결과가 다른 경우를 말한다

예를 들어 사용자 A가 회원 테이블에서 age가 12 이상인 회원들을 조회하는 쿼리를 보낸다고 할 때

이 결과로 세 개의 테이블이 조회된하고 해보자. 그 다음 사용자 B가 age가 15인 회원 레코드를 삽입한다. 그러면 그다음 세 개가 아닌 네 개의 테이블이 조회되는 것이다

반복 가능하지 않은 조회

한 트랜잭션 내의 같은 행에 두 번 이상 조회가 발생했는데, 그 값이 다른 경우를 가리킨다.
예를 들어 사용자 A가 큰돌의 보석 개수가 100개라는 값을 가진 데이터였는데, 그 이후 사용자 B가 그 값을 1로 변경해서 커밋했다고 하면 사용자 A는 100이 아닌 1을 읽게 된다
팬텀 리드와 다른 점은 반복 가능하지 않은 조회는 행 값이 달라질 수도 있는데, 팬텀 리드는 다른 행이 선택될 수도 있다는 것을 의미한다

더티 리드

반복 가능하지 않은 조회와 유사하며 한 트랜잭션이 실행 중일 때 다른 트랜잭션에 의해 수정되었지만 아직 커밋되지 않은 행의 데이터를 읽을 수 있을 때 발생한다. 예를 들어 사용자 A가 큰돌 보석 개수 100을 1로 변경한 내용이 커밋되지 않은 상태라도 그 이후 사용자 B가 조회한 결과가 1로 나오는 경우를 말한다

격리수준

SERIALIZABLE

트랜잭션을 순차적으로 진행시키는 것을 말한다

여러 트랜잭션이 동시에 같은 행에 접근할 수 없다. 이 수준은 매우 엄격한 수준으로 해당 행에 대해 격리시키고, 이후 트랜잭션이 이 행에 대해 일어난다면 기다려야 한다. 그렇기 때문에 교착 상태가 일어날 확률도 많고 가장 성능이 떨어지는 격리 수준이다

REPEATABLE_READ

하나의 트랜잭션이 수정한 행을 다른 트랜잭션이 수정할 수 없도록 막아주지만 새로운 행을 추가하는 것은 막지 않는다. 따라서 이후에 추가된 행이 발견될 수도 있다.

READ_COMMITTED

가장 많이 사용되는 격리 수준. MySQL8.0, PostgreSQL, SQL Server 오라클에서 기본값으로 설정되어 있다

READ_COMMITTED와는 달리 다른 트랜잭션이 커밋하지 않은 정보는 읽을 수 없다. 즉 커밋 완료된 데이터에 대해서만 조회를 허용한다.

하지만 어떤 트랜잭션이 접근한 행을 다른 트랜잭션이 수정할 수 있다. 예를 들어 트랜잭션 A가 수정한 행을 트랜잭션 B가 수정할 수도 있다. 이 때문에 트랜잭션 A가 같은 행을 다시 읽을 때 다른 내용이 발견될 수 있다.

READ_UNCOMMITTED

가장 낮은 수준으로 하나의 트랜잭션이 커밋되기 이전에 다른 트랜잭션에 노출되는 문제가 있지만 가장 빠르다

이는 데이터 무결성을 위해 되도록이면 사용하지 않는 것이 이상적이나, 몇몇 행이 제대로 조회되지 않더라도 괜찮은 거대한 양의 데이터를 어림잡아 집계하는 데는 사용하면 좋다

지속성

성공적으로 수행된 트랜잭션은 영원히 반영되어야 하는 것을 의미한다. 이는 데이터베이스에 시스템 장애가 발생해도 원래 상태로 복구하는 회복 가능성이 이어야 함을 뜻하며, 데이터베이스는 이를 위해 체크섬, 저널링, 룰백 등의 기능을 제공한다

- 체크섬

중복 검사의 한 형태로 오류 정정을 통해 송신된 자료의 무결성을 보호하는 단순한 방법

- 저널링

파일 시스템 또는 데이터베이스 시스템에 변경 사항을 반영하기 전에 로깅하는 것, 트랜잭션 등 변경 사항에 대한 로그를 남기는 것

4-3-2. 무결성

데이터의 정확성, 일관성, 유효성을 유지하는 것

무결성이 유지되어야 데이터베이스에 저장된 데이터 값과 그 값에 해당하는 현실 세계의 실제 값이 일치하는지에 대한 신뢰가 생긴다. 무결성의 종류는 다음과 같다.

개체 무결성	기본키로 선택된 필드는 빈 값을 허용하지 않는다
참조 무결성	서로 참조 관계에 있는 두 테이블의 데이터는 항상 일관된 값을 유지해야 한다
고유 무결성	특정 속성에 대해 고유한 값을 가지도록 조건이 주어진 경우 그 속성 값은 모두 고유한 값을 가진다
NULL 무결성	특정 속성 값에 NULL이 올 수 없다는 조건이 주어진 경우 그 속성 값은 NULL이 될 수 없다는 제약 조건

4-4. 데이터베이스의 종류

4-4-1. 관계형 데이터베이스

(RDBMS)는 행과 열을 가지는 표 형식 데이터를 저장하는 형태의 데이터베이스, SQL 이라는 언어를 써서 조작.

MySQL, PostgreSQL, 오라클, SQL Server, MSSQL 등이 있다

관계형 데이터베이스의 경우 표준 SQL은 지키기는 하지만 각각의 제품에 특화시킨 SQL을 사용한다. 예를 들어 오라클의 경우 PL/SQL이라고 하며 SQL Server에서는 T-SQL, MySQL은 SQL을 쓴다

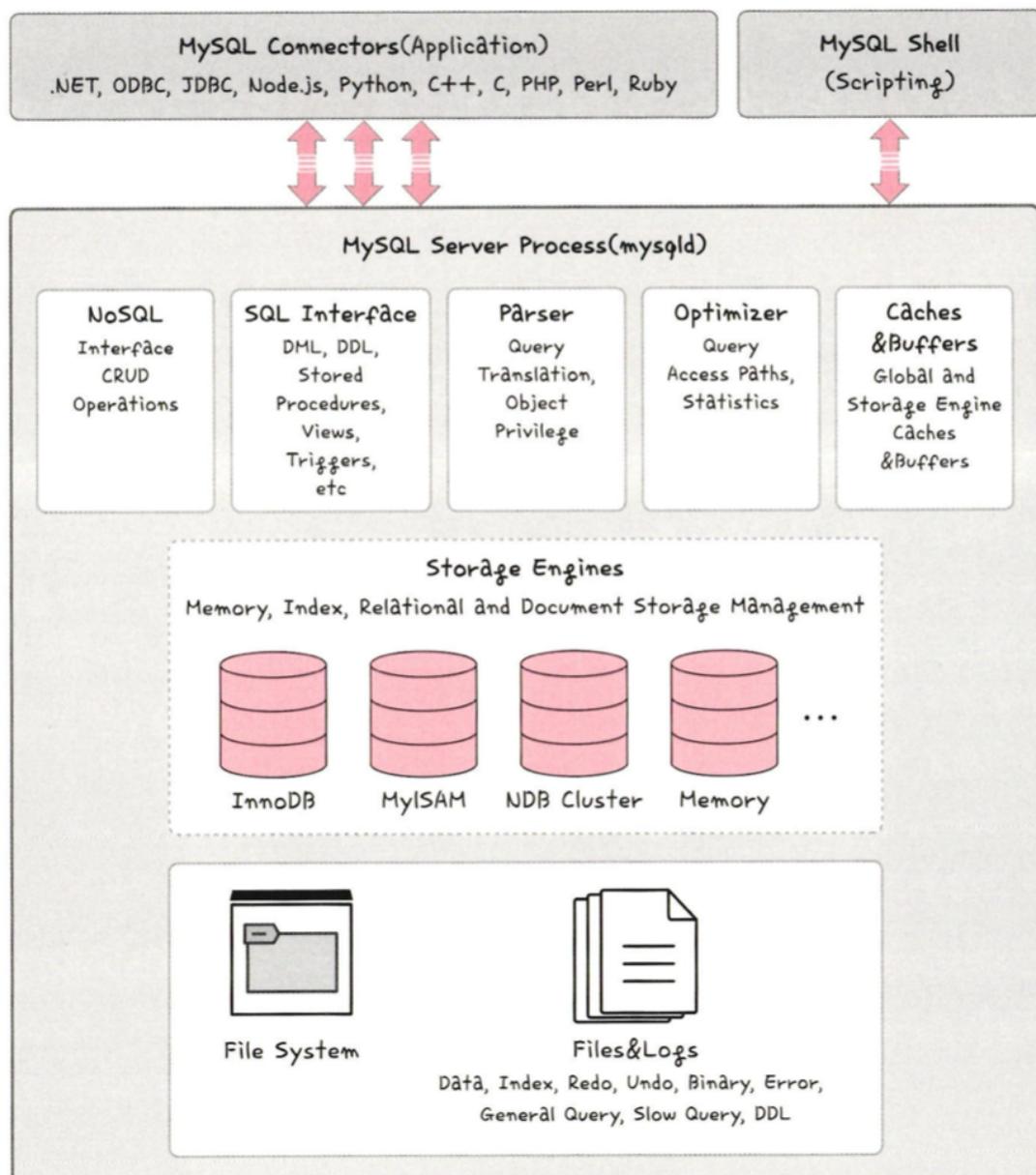
MySQL

대부분의 운영체제와 호환되며 현재 가장 많이 사용하는 데이터베이스이다.

C, C++로 만들어졌으며 MyISAM 인덱스 압축 기술, B-트리 기반의 인덱스, 스레드 기반의 메모리 할당 시스템, 매우 빠른 조인, 최대 64개의 인덱스를 제공합니다. 대용량 데이터베이스를 위해 설계되어 있고 룰백, 커밋, 이중 암호 지원 보안 등의 기능을 제공하며 많은 서비스에서 사용합니다.

MySQL의 스토리지 엔진 아키텍처는 다음과 같습니다.

▼ 그림 4-30 MySQL 스토리지 엔진 아키텍처



데이터베이스의 심장과도 같은 역할을 하는 곳이 바로 스토리지 엔진인데, 모듈식 아키텍처로 쉽게 스토리지 엔진을 바꿀 수 있으며 데이터 웨어하우징, 트랜잭션 처리, 고가용성 처리

에 강점을 두고 있다. 스토리지 엔진 위에는 커넥터 API 및 서비스 계층을 통해 MySQL 데이터베이스와 쉽게 상호 작용할 수 있다

또한 MySQL은 쿼리를 캐시를 지원해서 입력된 쿼리문에 대한 전체 결과 집합을 저장하기 때문에 사용자가 작성한 쿼리가 캐시에 있는 쿼리와 동일하면 서버는 단순히 구문 분석, 최적화 및 실행을 건너뛰고 캐시의 출력만 표시

PostgreSQL

MySQL 다음으로 개발자들이 선호하는 데이터베이스 기술

디스크 조각이 차지하는 영역을 회수할 수 있는 장치인 VACUUM이 특징이다. 최대 테이블 키기는 32TB이며 SQL뿐만 아니라 JSON을 이용해서 데이터에 접근할 수 있다. 지정 시간에 복구하는 기능, 로깅, 접근 제어, 중첩된 트랜잭션, 백업 등을 할 수 있다.

4-4-2. NoSQL 데이터베이스

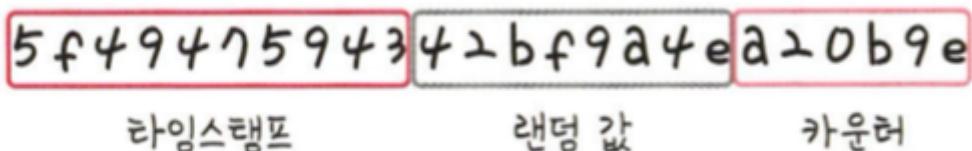
(Not Only SQL)이라는 슬로건에서 생겨난 데이터베이스

SQL을 사용하지 않는 데이터베이스를 말하며, 대표적으로 MongoDB와 redis 등이 있다

MongoDB

JSON을 통해 데이터에 접근할 수 있고, Binary JSON 형태로 데이터가 저장되며 와이어드 타이거 엔진이 기본 스토리지 엔진으로 장착된 키-값 데이터 모델에서 확장된 도큐먼트 기반의 데이터베이스이다. 확장성이 뛰어나며 빅데이터를 저장할 때 성능이 좋고 고가용성과 샤딩 레플리카셋을 지원한다. 또한 스키마를 정해 놓지 않고 데이터를 삽입할 수 있기 때문에 다양한 도메인의 데이터베이스를 기반으로 분석하거나 로깅 등을 구현할 때 강점을 보인다.

또한 MongoDB는 도큐먼트를 생성할 때마다 다른 컬렉션에서 중복된 값을 지니기 힘든 유니크한 값인 ObjectId가 생성된다



이는 기본키로 유닉스 시간 기반의 타임스탬프 (4바이트), 랜던 값 (5바이트), 카운터 (3바이트)로 이루어 짐

redis

인메모리 데이터베이스이자 키-값 데이터 모델 기반의 데이터베이스이다
기본적인 데이터 탑재는 문자열이며 최대 512MB까지 저장할 수 있다

pub/sub 기능을 통해 채팅 시스템, 다른 데이터베이스 앞단에 두어 사용하는 캐싱 계층, 단순한 키-값이 필요한 세션 정보 관리, 정렬된 셋 자료구조를 이용한 실시간 순위표 서비스에 사용한다

4-5. 인덱스

4-5-1. 인덱스의 필요성

인덱스는 데이터를 빠르게 찾을 수 있는 하나의 장치이다.

예를 들어 책의 마지막 장에 있는 찾아보기를 생각하면 된다

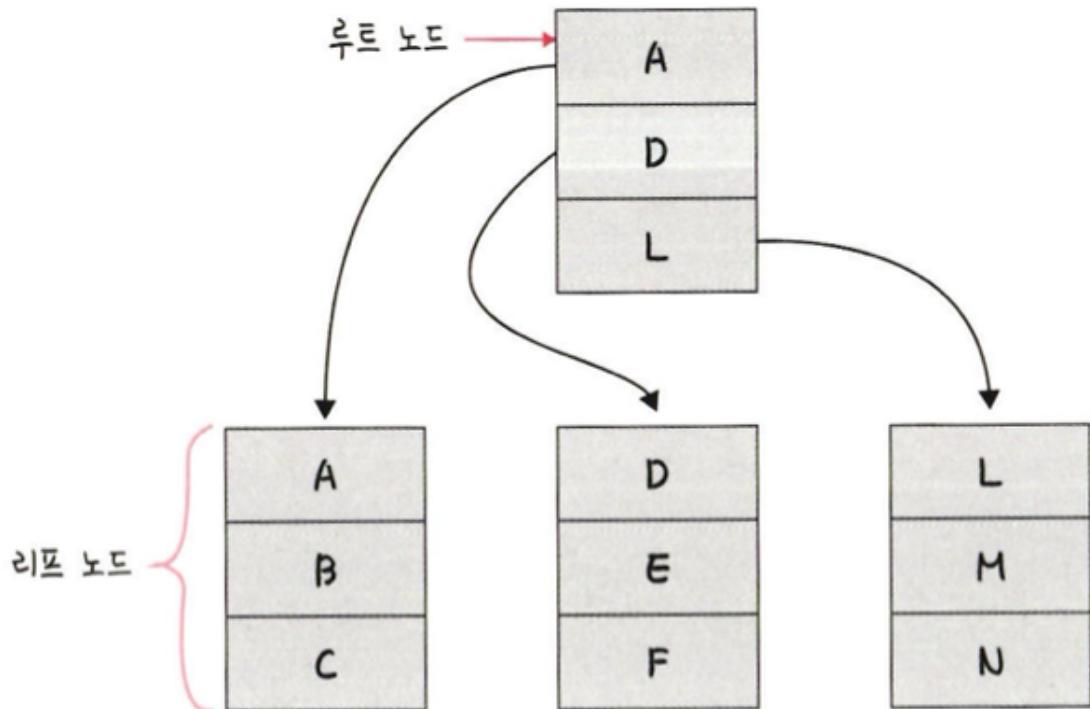
책의 본문에 있고 그 본문 안에 내가 찾고자 하는 항목을 찾아보기를 통해 빠르게 찾을 수 있다. 이와 마찬가지로 인덱스를 설정하면 테이블 안에 내가 찾고자 하는 데이터를 빠르게 찾을 수 있다

4-5-2. B-트리

인덱스는 보통 B-트리라는 자료구조로 이루어져 있다. 이는 루트 노드, 리프 노드, 그리고 루트 노드와 리프 노드 사이에 있는 브랜치 노드로 나뉜다

먼저 루트 노드와 리프 노드를 기반으로 설명하면 다음과 같다

▼ 그림 4-36 B-트리 예제 1

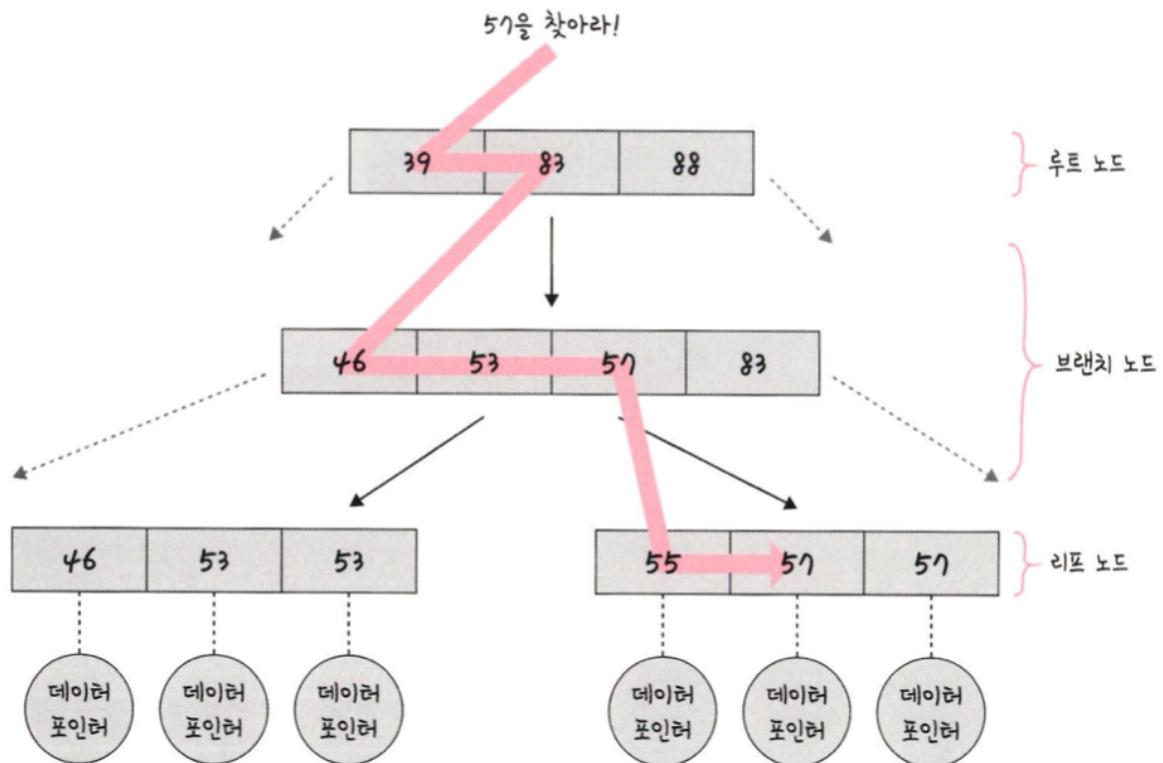


예를 들어 E를 찾는다고 하면 전체 테이블을 탐색하는 것이 아니라 E가 있을 법한 리프 노드로 들어가서 E를 탐색하면 쉽게 찾을 수 있다

이 자료 구조 없이 E를 탐색하고자 하면 ABCDE 다섯 번을 탐색해야 하지만, 이렇게 노드들로 나누면 두 번만에 리프 노드에서 찾을 수 있다

만약 키 57에 해당하는 데이터를 검색해야 한다고 해보자

▼ 그림 4-37 B-트리 예제 2



트리 탐색은 맨 위 루트 노드부터 탐색이 일어나며 브랜치 노드를 거쳐 리프 노드까지 내려온다. 57보다 같거나 클 때까지 \leq 를 기반으로 처음 루트 노드에서는 39, 83 이후 아래 노드로 내려와 46, 53, 57 등 정렬된 값을 기반으로 탐색하는 것을 볼 수 있다. 이렇게 루트 노드부터 시작하여 마지막 리프 노드에 도달해서 57이 가리키는 데이터 포인터를 통해 결과값을 반환하게 된다.

인덱스가 효율적인 이유와 대수 확장성

인덱스가 효율적인 이유는 효율적인 단계를 거쳐 모든 요소에 접근할 수 있는 균형 잡힌 트리 구조와 트리 깊이의 대수확장성 때문이다

대수확장성이란 트리 깊이가 리프 노드 수에 비해 매우 느리게 성장하는 것을 의미한다. 기본적으로 인덱스가 한 깊이씩 증가할 때마다 최대 인덱스 항목의 수는 4배씩 증가한다

▼ 표 4-3 트리의 대수확장성

트리 깊이	인덱스 항목의 수
3	64
4	256
5	1,024
6	4,096
7	16,384
8	65,536
9	262,144
10	1,048,576

표처럼 트리 깊이는 열 개짜리로, 100만 개의 레코드를 검색할 수 있다는 의미다
참고로 실제 인덱스는 이것보다 훨씬 더 효율적이며 그렇기 때문에 인덱스가 효율적이라고
볼 수 있다.

4-5-3. 인덱스 만드는 방법

인덱스를 만드는 방법은 데이터베이스마다 다르며 MySQL과 MongoDB를 기준으로 설명

MySQL

클러스터형 인덱스와 세컨더리 인덱스가 있다

클러스터형 인덱스는 테이블당 하나를 설정할 수 있다. primary key 옵션으로 기본키를 만들면 클러스터형 인덱스를 생성할 수 있고, 기본키로 만들지 않고 unique not null 옵션을 붙이면 클러스터형 인덱스로 만들 수 있다

`create index...` 명령어를 기반으로 만들면 세컨더리 인덱스를 만들 수 있다. 하나의 인덱스만 생성할 것이라면 클러스터형 인덱스를 만드는 것이 세컨더리 인덱스를 만드는 것보다 성능이 좋다

MongoDB

MongoDB의 경우 도큐먼트를 만들면 자동으로 ObjectId 가 형성되며 해당 키가 기본키로 설정된다. 그리고 세컨더리키도 부가적으로 설정해서 기본키와 세컨더리키를 같이 쓰는 복합 인덱스를 설정할 수 있다

4-5-4. 인덱스 최적화 기법

데이터베이스마다 조금씩 다르지만 기본적인 골조는 똑같다

MongoDB를 기반으로 인덱스 최적화 기법을 설명할 것이며 이를 기반으로 다른 데이터베이스에 웬만큼 적용할 수 있다

1. 인덱스는 비용이다

먼저 인덱스는 두 번 탐색하도록 강요한다

인덱스 리스트, 그다음 컬렉션 순으로 탐색하기 때문이며, 관련 읽기 비용이 들게 된다.

또한 컬렉션이 수정되었을 때 인덱스도 수정되어야 한다. 마치 책의 본문이 수정되었을 때 목차나 찾아보기도 수정해야 하는 것과 마찬가지

이때 B-트리의 높이를 균형 있게 조절하는 비용도 들고, 데이터를 효율적으로 조회할 수 있도록 분산시키는 비용도 들게 된다.

그렇게 때문에 쿼리에 있는 필드에 인덱스를 무작정 다 설정하는 것은 답이 아니다. 또한, 컬렉션에서 가져와야 하는 양이 많을수록 인덱스를 사용하는 것이 비효율적이다.

2. 항상 테스팅하라

인덱스 최적화 기법은 서비스 특징에 따라 달라진다.

서비스에서 사용하는 객체의 깊이, 테이블의 양 등이 다르기 때문이다

그렇게 때문에 항상 테스팅하는 것이 중요하다. explain() 함수를 통해 인덱스를 만들고 쿼리를 보낸 이후에 테스팅을 하며 걸리는 시간을 최소화 해야 한다.

3. 복합 인덱스는 같음, 정렬, 다중 값, 카디널리티 순이다

보통 여러 필드를 기반으로 조회를 할 때 복합 인덱스를 생성하는데, 이 인덱스를 생성할 때는 순서가 있고 생성 순서에 따라 인덱스 성능이 달라진다. 같음, 정렬, 다중 값, 카디널리티 순으로 생성해야 한다.

1. 어떠한 값과 같음을 비교하는 ==이나 equal이라는 쿼리가 있다면 제일 먼저 인덱스로 설정합니다.
2. 정렬에 쓰는 필드라면 그다음 인덱스로 설정합니다.
3. 다중 값을 출력해야 하는 필드, 즉 쿼리 자체가 >이거나 < 등 많은 값을 출력해야 하는 쿼리에 쓰는 필드라면 나중에 인덱스를 설정합니다.
4. 유니크한 값의 정도를 카디널리티라고 합니다. 이 카디널리티가 높은 순서를 기반으로 인덱스를 생성해야 합니다. 예를 들어 age와 email이 있다고 해봅시다. 어떤 것이 더 높죠? 당연히 email입니다. 즉, email이라는 필드에 대한 인덱스를 먼저 생성해야 하는 것입니다.

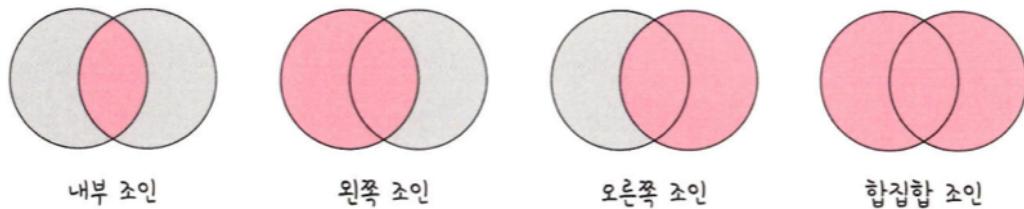
4-6. 조인의 종류

조인이란 하나의 테이블이 아닌 두 개 이상의 테이블을 묶어서 하나의 결과물을 만드는 것을 말한다. MySQL에서는 JOIN이라는 쿼리로, MongoDB에서는 lookup이라는 쿼리로 이를 처리할 수 있다.

참고로 MongoDB는 조인 연산 (lookup)에 대해 관계형 데이터베이스보다 성능이 떨어지기 때문에 여러 테이블을 조인하는 작업이 많을 경우 MongoDB보다는 관계형 데이터베이스를 사용해야 한다

조인의 종류 중 대표적인 내부 조인, 왼쪽 조인, 오른쪽 조인, 합집합 조인을 살펴보자

▼ 그림 4-38 조인의 종류



앞의 그림처럼 두 테이블 간의 교집합이 있다고 할 때, 다음과 같은 네 가지 조인이 있습니다.

- **내부 조인(inner join)**: 왼쪽 테이블과 오른쪽 테이블의 두 행이 모두 일치하는 행이 있는 부분만 표기합니다.
- **왼쪽 조인(left outer join)**: 왼쪽 테이블의 모든 행이 결과 테이블에 표기됩니다.
- **오른쪽 조인(right outer join)**: 오른쪽 테이블의 모든 행이 결과 테이블에 표기됩니다.
- **합집합 조인(full outer join)**: 두 개의 테이블을 기반으로 조인 조건에 만족하지 않는 행까지 모두 표기합니다.

4-6-1. 내부 조인

내부 조인은 두 테이블 간에 교집합을 나타낸다

SQL

```
SELECT * FROM TableA A  
INNER JOIN TableB B ON  
A.key = B.key
```

4-6-2. 왼쪽 조인

왼쪽 조인은 테이블 B의 일치하는 부분의 레코드와 함께 테이블 A를 기준으로 완전한 레코드 집합을 생성한다. 만약 테이블 B에 일치하는 항목이 없으면 해당 값은 null 값이 된다.

4-6-3. 오른쪽 조인

오른쪽 조인은 테이블 A에서 일치하는 부분의 레코드와 함께 테이블 B를 기준으로 완전한 레코드 집합을 생성한다. 만약 테이블 A에 일치하는 항목이 없으면 해당 값은 null 값이 됨

SQL

```
SELECT * FROM TableA A  
RIGHT JOIN TableB B ON  
A.key = B.key
```

4-6-4. 합집합 조인

합집합 조인(완전 외부 조인)은 양쪽 테이블에서 일치하는 레코드와 함께 테이블 A와 테이블 B의 모든 레코드 집합을 생성한다. 이때 일치하는 항목이 없으면 누락된 쪽에 null 값이 포함되어 출력된다.

SQL

```
SELECT * FROM TableA A  
FULL OUTER JOIN TableB B ON  
A.key = B.key
```

4-7. 조인의 원리

앞서 설명한 조인은 조인의 원리를 기반으로 조인 작업이 이루어진다. 조인의 원리인 중첩 루프 조인, 정렬 병합 조인, 해시 조인에 대해 알아보자

앞서 설명한 조인의 종류는 이 원리를 기반으로 조인을 하는 것이다.

4-7-1. 중첩 루프 조인

중첩 for 문과 같은 원리로 조건에 맞는 조인을 하는 방법

랜덤 접근에 대한 비용이 많이 증가하므로 대용량의 테이블에서는 사용하지 않는다

예를 들어 t1 t2 테이블을 조인한다고 했을 때 첫 번째 테이블에서 행을 한 번에 하나씩 읽고 그다음 테이블에서도 행을 하나씩 읽어 조건에 맞는 레코드를 찾아 결과값을 반환

의사 코드

```
for each row in t1 matching reference key {  
    for each row in t2 matching reference key {  
        if row satisfies join conditions, send to client  
    }  
}
```

참고로 중첩 루프 조인에서 발전한 조인할 테이블을 작은 블록으로 나눠서 블록 하나씩 조인하는 블록 중첩 루프 조인이라는 방식도 있다

4-7-2. 정렬 병합 조인

정렬 병합 조인이란 각각의 테이블을 조인할 필드 기준으로 정렬하고 정렬이 끝난 이후에 조인 작업을 수행하는 조인이다. 조인할 때 쓸 적절한 인덱스가 없고 대용량의 테이블들을 조인하고 조인 조건으로 < > 등 범위 비교 연산자가 있을 때 사용

4-7-3. 해시 조인

해시 테이블을 기반으로 조인하는 방법

두 개의 테이블을 조인한다고 했을 때 하나의 테이블이 메모리에 온전히 들어간다면 보통 중첩 루프 조인보다 더 효율적이다. (메모리에 올릴 수 없을 정도로 크다면 디스크를 사용하는 비용이 발생된다) 또한 동등(=) 조인에서만 사용할 수 있다

MySQL의 경우 MySQL8.0.18 릴리스와 함께 이 기능을 사용할 수 있게 되었으며 이를 기반으로 해시 조인의 과정을 살펴보자

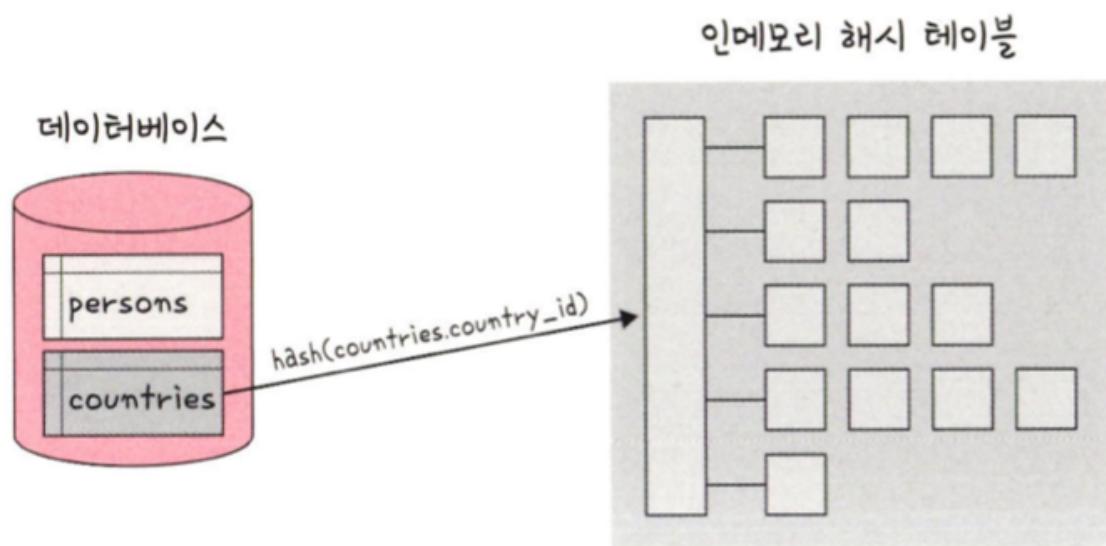
MySQL의 해시 조인 단계는 빌드 단계, 프로브 단계로 나뉜다

빌드 단계

빌드 단계는 입력 테이블 중 하나를 기반으로 메모리 내 해시 테이블을 빌드하는 단계다

예를 들어 person와 countries라는 테이블을 조인한다고 했을 때 둘 중에 바이트가 더 작은 테이블을 기반으로 해서 테이블을 빌드한다.

▼ 그림 4-40 빌드 단계

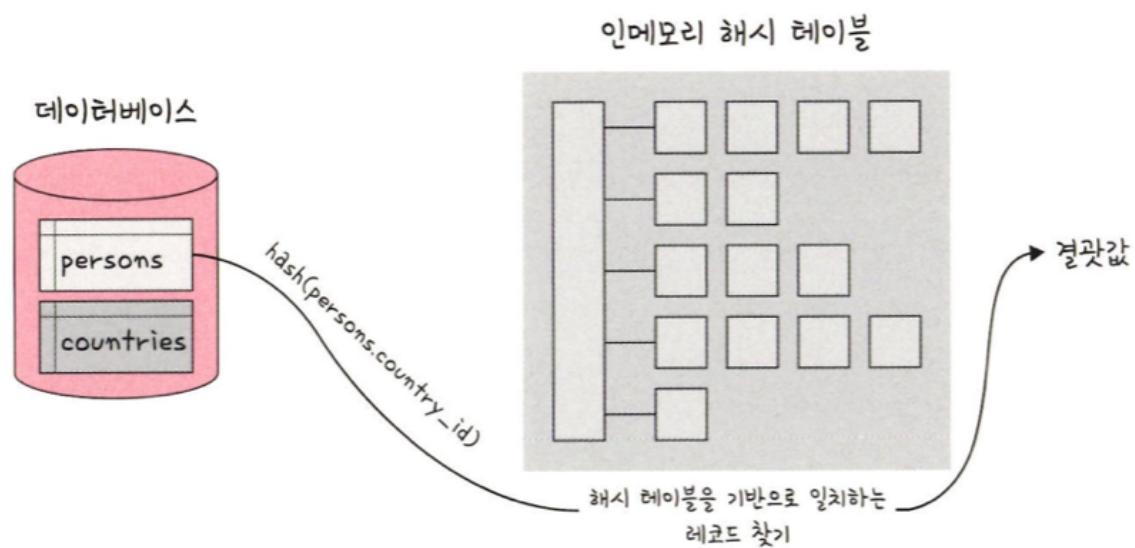


또한 조인에 사용되는 필드가 해시 테이블의 키로 사용된다. `countries.country_id`가 키로 사용되는 것을 볼 수 있다

프로브 단계

프로브 단계 동안 레코드 읽기를 시작하며, 각 레코드에서 `persons.country_id`에 일치하는 레코드를 찾아서 결과값으로 반환한다

▼ 그림 4-41 프로브 단계



이를 통해 각 테이블은 한 번씩만 읽게 되어 중첩해서 두 개의 테이블을 읽는 중첩 루프 조인 보다 보통은 성능이 더 좋다. 참고로 사용 가능한 메모리양은 시스템 변수 `join_buffer_size`에 의해 제어되며 런타임 시에 조정할 수 있다.