

데이터베이스 정리

만든이: 안수균(sg0xad)

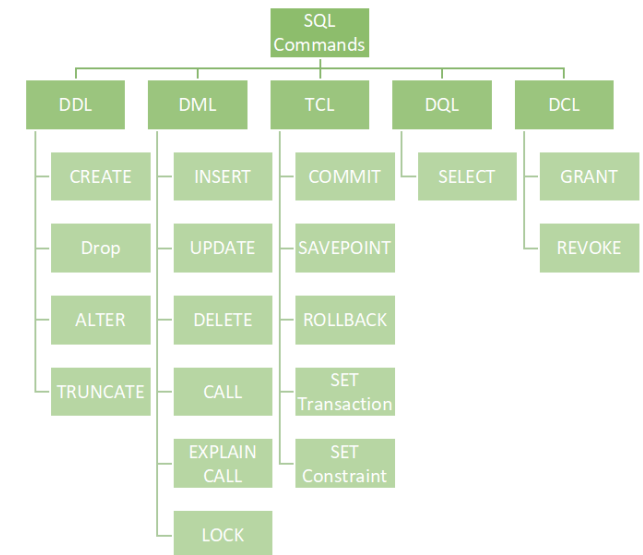
SQL이란?

Structured Query Language의 약자로 사람과 데이터베이스간의 소통하는 언어이다.

언어다보니 사투리처럼 여러가지 SQL이 있으며 대표적으로 MySQL, PostgreSQL, MSSQL, Oracle, MariaDB, SQLite 등이 있다.

다른 DBMS의 SQL을 DBMS에 실행하면 되는 것도 있고 안되는 것도 있다. 이 때문에 중구난방이 될 수 있으므로 표준이 정해져 있는데 그걸 **ANSI SQL**이라고 부른다.

SQL을 배워도 잊어버릴 수 있으니 그럴 때는 cheat sheet를 이용하도록 하자.
(<https://cheatography.com/tag/database/>)



DBMS 설치

DBMS는 Database Management System의 약자이다.

MySQL은 여기서 설치 할 수 있다. (5버전 8버전 다 괜찮음)

<https://dev.mysql.com/downloads/installer/>

Select Version:
8.0.34

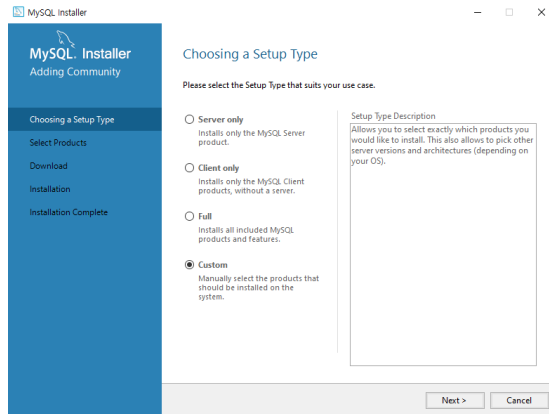
Select Operating System:
Microsoft Windows

| | | | |
|---|--------|--------|--------------------------|
| Windows (x86, 32-bit), MSI Installer (mysql-installer-web-community-8.0.34.0.msi) | 8.0.34 | 2.4M | Download |
| Windows (x86, 32-bit), MSI Installer (mysql-installer-community-8.0.34.0.msi) | 8.0.34 | 331.3M | Download |

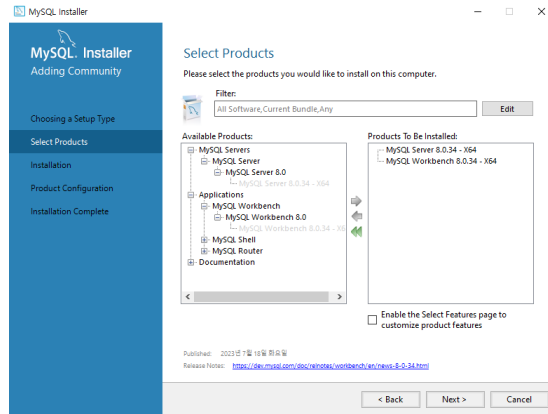
위에 것을 설치하나 밑에 것을 설치하나 똑같다.
위에 것은 2.4M을 받고나서 나중에 실행하면 331M을
추가로 더 받음.

DBMS 설치

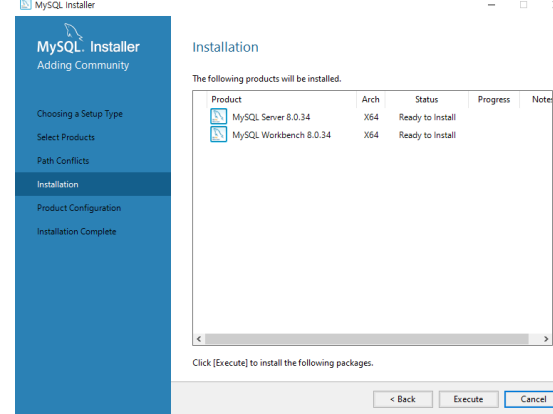
Workbench는 데이터베이스를 시각적으로 편하게 관리 해주는 프로그램이다.
Workbench보다 더 좋은 프로그램 중에 대표적으로 datagrip가 있다. (유료)



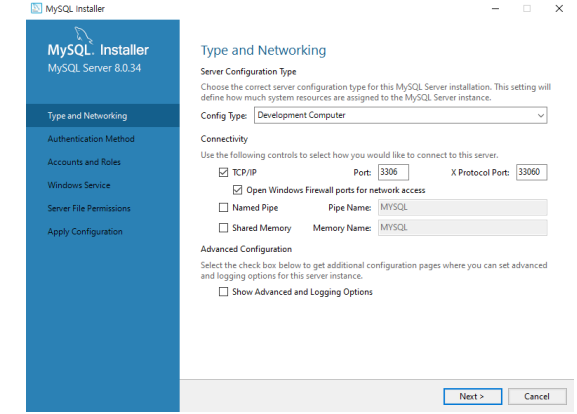
1. Custom을 선택한다.



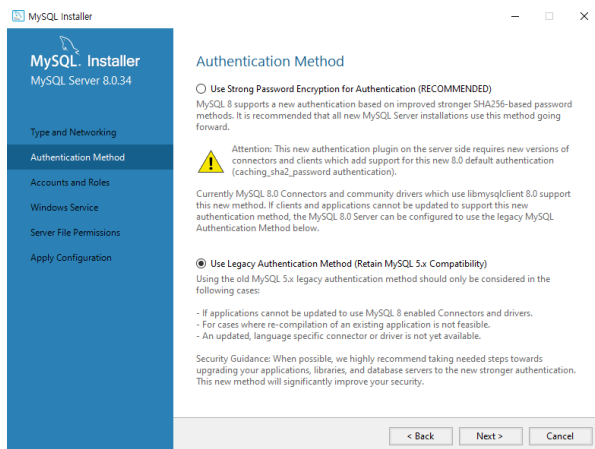
2. MySQL Server과 Mysql Workbench를 선택한다.



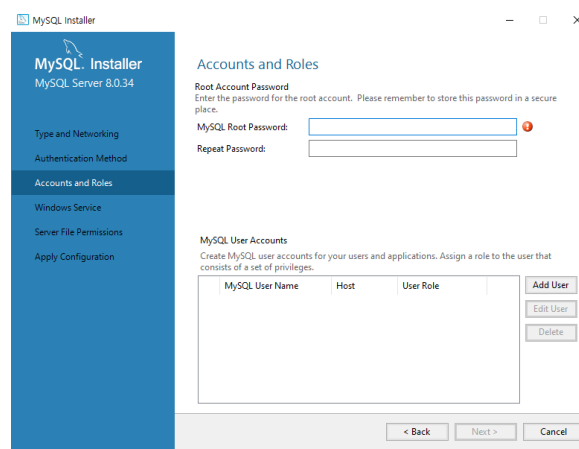
3. Excute를 눌러 설치



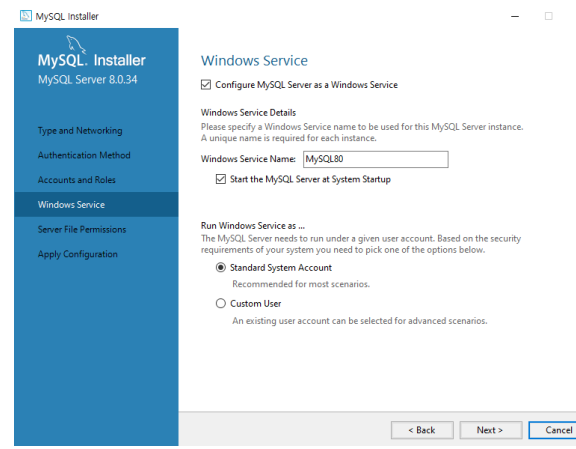
4. 포트 3306으로 하고 Next



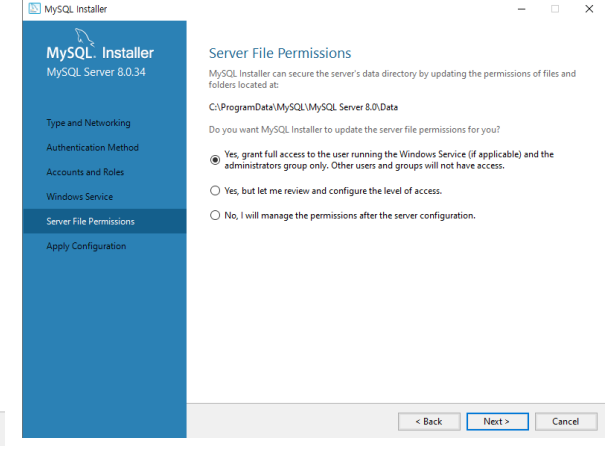
5. 밑에 것을 선택하고 Next



6. 패스워드를 입력하고 Next



6. Configure MySQL Server as a Windows Service 반드시 체크 된 상태에서 Next



7. 그대로 Next하고 Excute

DB를 엑셀이라고 생각하자

| 아이디 | 이름 | 전화번호 | 이메일 | 연봉 | 팀 | 역할 | 프로젝트 | |
|-----|------|--------------------------------|--|-------|------|----|------------|--|
| 1 | 제로초 | 010-1234-5678 010-2345-6789 | zerohch0@gmail.com | 10000 | 개발팀 | 팀장 | 홈페이지 AI | |
| 2 | 원초 | 010-1234-5679 | onecho@gmail.com | 4000 | 개발팀 | 신입 | 홈페이지 | |
| 3 | 투초 | 010-1244-5678 | twocho@gmail.com | 6000 | 기획팀 | 과장 | 홈페이지 AI | |
| 4 | 쓰리초 | 010-1334-5678 | threecho@gmail.com | 5000 | 디자인팀 | 대리 | 홈페이지 | |
| 5 | 포초 | | fourcho@gmail.com | 8000 | 개발팀 | 과장 | AI | |
| 6 | 파이버초 | | fivecho@gmail.com | 3000 | 개발팀 | 인턴 | 홈페이지 | |

이렇게 데이터가 저장 되어있는 형태를 테이블이라고 부른다.

아이디, 이름, 전화번호, 이메일, 연봉, 팀, 역할, 프로젝트는 필드(속성)이라고 부른다.

세로축을 Column(열)이라고 부르고, 가로축은 Row(행)이라고 부르며 레코드라고도 부른다.

각 열들은 비어 있을 수 있다.

제1정규형으로 만들기

| 아이디 | 이름 | 전화번호 | 이메일 | 연봉 | 팀 | 역할 | 프로젝트 |
|-----|------|--------------------------------|--|-------|------|----|------------|
| 1 | 제로초 | 010-1234-5678 010-2345-6789 | zerohch0@gmail.com | 10000 | 개발팀 | 팀장 | 홈페이지 AI |
| 2 | 원초 | 010-1234-5679 | onecho@gmail.com | 4000 | 개발팀 | 신입 | 홈페이지 |
| 3 | 투초 | 010-1244-5678 | twocho@gmail.com | 6000 | 기획팀 | 과장 | 홈페이지 AI |
| 4 | 쓰리초 | 010-1334-5678 | threecho@gmail.com | 5000 | 디자인팀 | 대리 | 홈페이지 |
| 5 | 포초 | | fourcho@gmail.com | 8000 | 개발팀 | 과장 | AI |
| 6 | 파이브초 | | fivecho@gmail.com | 3000 | 개발팀 | 인턴 | 홈페이지 |

전화번호 속성값과 프로젝트 열에는 두개 이상의 값이 들어있는 것들이 있다.
제1정규형은 모든 열에는 원자값만 포함해야 한다.

즉, 각 열은 더 이상 분해할 수 없는 단일 값이어야 하므로 아래와 같이 테이블을 분해해서 제1정규형을 만족시켜야 한다.

| 아이디 | 이름 | 이메일 | 연봉 | 팀 | 역할 |
|-----|------|--|-------|------|----|
| 1 | 제로초 | zerohch0@gmail.com | 10000 | 개발팀 | 팀장 |
| 2 | 원초 | onecho@gmail.com | 4000 | 개발팀 | 신입 |
| 3 | 투초 | twocho@gmail.com | 6000 | 기획팀 | 과장 |
| 4 | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 대리 |
| 5 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | 과장 |
| 6 | 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 인턴 |

사원 테이블

| 이름 | 전화번호 |
|-----|---------------|
| 제로초 | 010-1234-5678 |
| 제로초 | 010-2345-6789 |
| 원초 | 010-1234-5679 |
| 투초 | 010-1244-5678 |
| 쓰리초 | 010-1334-5678 |

전화번호 테이블

| 이름 | 프로젝트 |
|------|------|
| 제로초 | 홈페이지 |
| 제로초 | AI |
| 원초 | 홈페이지 |
| 투초 | 홈페이지 |
| 쓰리초 | 홈페이지 |
| 포초 | AI |
| 파이브초 | 홈페이지 |

프로젝트 테이블

아이디가 필요한 이유(기본키, 외래키)

기본키: Primary Key(PK)

외래키: Foreign Key(FK)

| 이름 | 이메일 | 연봉 | 팀 | 역할 |
|------|--------------------|-------|------|----|
| 제로초 | zerohch0@gmail.com | 10000 | 개발팀 | 팀장 |
| 원초 | onecho@gmail.com | 4000 | 개발팀 | 신입 |
| 투초 | twocho@gmail.com | 6000 | 기획팀 | 과장 |
| 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 대리 |
| 포초 | fourcho@gmail.com | 8000 | 개발팀 | 과장 |
| 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 인턴 |
| 제로초 | zerocho@gmail.com | 8000 | 기획팀 | 팀장 |

사원 테이블

| 이름 | 전화번호 |
|-----|---------------|
| 제로초 | 010-1234-5678 |
| 제로초 | 010-2345-6789 |
| 원초 | 010-1234-5679 |
| 투초 | 010-1244-5678 |
| 쓰리초 | 010-1334-5678 |

전화번호 테이블

| 이름 | 프로젝트 |
|------|------|
| 제로초 | 홈페이지 |
| 제로초 | AI |
| 원초 | 홈페이지 |
| 투초 | 홈페이지 |
| 쓰리초 | 홈페이지 |
| 포초 | AI |
| 파이브초 | 홈페이지 |

프로젝트 테이블

1. 동명이인이 존재할 경우 각 레코드를 구별할 수가 없을뿐더러 동명이인 때문에 전화번호 테이블과 프로젝트 테이블의 이름이랑 매칭이 안된다.

| 이름 | 이메일(unique, 키) | 연봉 | 팀 | 역할 |
|------|--------------------|-------|------|----|
| 제로초 | zerohch0@gmail.com | 10000 | 개발팀 | 팀장 |
| 원초 | onecho@gmail.com | 4000 | 개발팀 | 신입 |
| 투초 | twocho@gmail.com | 6000 | 기획팀 | 과장 |
| 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 대리 |
| 포초 | fourcho@gmail.com | 8000 | 개발팀 | 과장 |
| 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 인턴 |
| 제로초 | zerocho@gmail.com | 8000 | 기획팀 | 팀장 |

사원 테이블

| 이름(외래키) | 전화번호 |
|-------------------|---------------|
| zerohch0@gmail.cc | 010-1234-5678 |
| zerohch0@gmail.cc | 010-2345-6789 |
| onecho@gmail.com | 010-1234-5679 |
| twocho@gmail.com | 010-1244-5678 |
| threecho@gmail.co | 010-1334-5678 |

전화번호 테이블

| 이름(외래키) | 프로젝트 |
|-------------------|------|
| zerohch1@gmail.cc | 홈페이지 |
| zerohch1@gmail.cc | AI |
| onecho@gmail.com | 홈페이지 |
| twocho@gmail.com | 홈페이지 |
| threecho@gmail.co | 홈페이지 |
| fourcho@gmail.com | AI |
| fivecho@gmail.com | 홈페이지 |
| twocho@gmail.com | AI |

프로젝트 테이블

2. 그래서 고유한(unique) 이메일을 키로 설정해서 각 레코드를 구별시키고, 사원테이블에서 키를 가져와서 전화번호 테이블과 프로젝트 테이블에 외래키로 연결할 수 있다.

| 아이디(수정X) | 이름 | 이메일(unique, 키) | 연봉 | 팀 | 역할 |
|----------|------|--------------------|-------|------|----|
| 1 | 제로초 | zerohch1@gmail.com | 10000 | 개발팀 | 팀장 |
| 2 | 원초 | onecho@gmail.com | 4000 | 개발팀 | 신입 |
| 3 | 투초 | twocho@gmail.com | 6000 | 기획팀 | 과장 |
| 4 | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 대리 |
| 5 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | 과장 |
| 6 | 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 인턴 |
| 7 | 제로초 | zerocho@gmail.com | 8000 | 기획팀 | 팀장 |

사원 테이블

| 아이디(기본키) | 사원아이디(외래키) | 전화번호 |
|----------|------------|---------------|
| 1 | 1 | 010-1234-5678 |
| 2 | 1 | 010-2345-6789 |
| 3 | 2 | 010-1234-5679 |
| 4 | 3 | 010-1244-5678 |
| 5 | 4 | 010-1334-5678 |

전화번호 테이블

| 아이디 | 사원아이디(외래키) | 프로젝트 |
|-----|------------|------|
| 1 | 1 | 홈페이지 |
| 2 | 1 | AI |
| 3 | 2 | 홈페이지 |
| 4 | 3 | 홈페이지 |
| 5 | 4 | 홈페이지 |
| 6 | 5 | AI |
| 7 | 6 | 홈페이지 |
| 8 | 3 | AI |

프로젝트 테이블

3. 하지만 이메일은 수정될 수 있기 때문에 바뀌지 않는 고유한 키를 따로 만들어서 설정해 놓는 게 좋다.
이렇게 따로 만드는 키를 인조키라고 부르며, 고유한 키를 기본키(PK)라고 부른다.

4. 전화번호 테이블과 프로젝트 테이블의 외래키도 중복이 되니까 기본키를 따로 둘 수 있다.
(외래키와 전화번호를 함께 복합키로 설정 할 수 있지만 전화번호가 바뀔 수 있으므로 권장X)

아이디를 연이은 숫자로 할 때 단점

| 아이디(수정X) | 이름 | 이메일(unique, 키) | 연봉 | 팀 | 역할 |
|----------|------|--|-------|------|----|
| 1 | 제로초 | zerohch1@gmail.com | 10000 | 개발팀 | 팀장 |
| 2 | 원초 | onecho@gmail.com | 4000 | 개발팀 | 신입 |
| 3 | 투초 | twocho@gmail.com | 6000 | 기획팀 | 과장 |
| 4 | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 대리 |
| 5 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | 과장 |
| 6 | 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 인턴 |
| 7 | 제로초 | zerocho@gmail.com | 8000 | 기획팀 | 팀장 |

이렇게 기본키를 연이은 숫자로 하면 새로 회원가입을 했을 때 아이디가 8번으로 나오면 회원수가 8명밖에 없다는 등의 회사의 정보가 간접적으로 노출 될 수 있다.

| 아이디(수정X, 기본키) | 이름 | 이메일(unique, 키) | 연봉 | 팀 | 역할 |
|---------------|------|--|-------|------|----|
| ab341ed | 제로초 | zerohch1@gmail.com | 10000 | 개발팀 | 팀장 |
| 23c2124 | 원초 | onecho@gmail.com | 4000 | 개발팀 | 신입 |
| 124fasc | 투초 | twocho@gmail.com | 6000 | 기획팀 | 과장 |
| 31fdsava | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 대리 |
| fafwe2 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | 과장 |
| asdfaf12 | 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 인턴 |
| asdfasf3 | 제로초 | zerocho@gmail.com | 8000 | 기획팀 | 팀장 |

그래서 기본키를 이런식으로 절대 겹치지 않는 랜덤한 문자열로 설정하는 경우도 있다.

UUID 550e8400-e29b-41d4-a716-446655440000

Nano ID "Y1StGX98_Z5jdHi6B-myT"

랜덤한 문자열에는 대표적으로 위와 같은 UUID와 Nano ID 등이 있다.

일대일, 일대다, 다대다 관계(ERD)



자주 안쓰이는 데이터들은 하나의 테이블에 같이 있어야 할 것들을 이렇게 1:1로 분리를 한다.



사원 한명이 전화번호를 여러 개 가질 수 있지만, 사원테이블의 전화번호 하나는 사원 한명에게만 속해 있다.
1:N

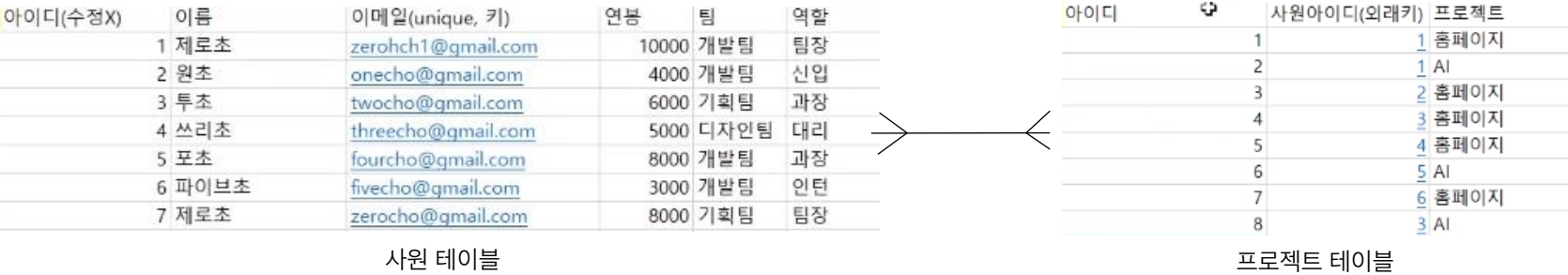


사원 한명이 여러 프로젝트를 가질 수 있고, 각 프로젝트는 여러 사원에게 속해 있을 수 있다.
M:N

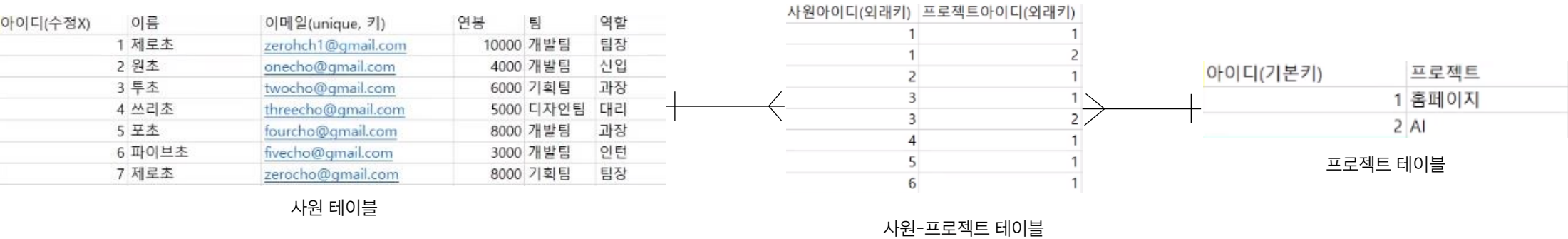
이렇게 되면 잘못된 테이블이다.

예를들어, 프로젝트 명인 홈페이지를 홈페이지1로 바꿀 경우 모두 바뀌줘야 하며 실수로 한 개 못바꾸면 새로운 프로젝트가 생긴거나 다름이 없기 때문이다.

다대다 관계일 때




위와 같이 M:N 일 경우에는 아래와 같이 프로젝트 테이블과 사원-프로젝트 테이블로 분리를 한다.



사원과 사원-프로젝트 테이블은 1:N, 프로젝트와 사원-프로젝트 테이블도 1:N
다대다 관계일 때는 이렇게 1:N 관계 두개로 분해된다.

soft delete VS hard delete

| 아이디(수정X, 기본키) | 이름 | 이메일(unique, 키) | 연봉 | 팀 | 역할 |
|---------------|---|--|-------|------|----|
| 1 | 제로초 | zerohch1@gmail.com | 10000 | 개발팀 | 팀장 |
| | | | | | |
| 3 | 투초 | twocho@gmail.com | 6000 | 기획팀 | 과장 |
| 4 | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 대리 |
| 5 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | 과장 |
| 6 | 파이버초 | fivecho@gmail.com | 3000 | 개발팀 | 인턴 |
| 7 | 제로초  | zerocho@gmail.com | 8000 | 기획팀 | 팀장 |

원초라는 사원이 퇴사를 해서 위와 같이 레코드를 지웠다고 가정을 해보자. (hard delete)
그런데 나중에 원초라는 사람이 경력증명서를 떼어달라고 하면 데이터를 지워버려서 줄 수 없는 상황이 생길 수있다.

| 아이디(수정X, 기본키) | 이름 | 이메일(unique, 키) | 연봉 | 팀 | 역할 | 퇴사일 |
|---------------|------|--|-------|------|----|--|
| 1 | 제로초 | zerohch1@gmail.com | 10000 | 개발팀 | 팀장 | |
| 2 | 원초 | onecho@gmail.com | 4000 | 개발팀 | 신입 |  2023-06-19 |
| 3 | 투초 | twocho@gmail.com | 6000 | 기획팀 | 과장 | |
| 4 | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 대리 | |
| 5 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | 과장 | |
| 6 | 파이버초 | fivecho@gmail.com | 3000 | 개발팀 | 인턴 | |
| 7 | 제로초 | zerocho@gmail.com | 8000 | 기획팀 | 팀장 | |

그래서 실제로 데이터를 지워버리는 게 아니라 간접적으로 퇴사했다고 데이터를 추가할 수 있다. (soft delete)

제2정규형으로 만들기

기본키(복합키)

| 사원아이디(외래키) | 프로젝트아이디(외래키) | 마감일 |
|------------|--------------|------------|
| 1 | 1 | 2023-07-16 |
| 1 | 2 | 2024-03-15 |
| 2 | 1 | 2023-07-16 |
| 3 | 1 | 2023-07-16 |
| 3 | 2 | 2024-03-15 |
| 4 | 1 | 2023-07-16 |
| 5 | 1 | 2023-07-16 |
| 6 | 1 | 2023-07-16 |

사원-프로젝트 테이블

사원-프로젝트 테이블에 마감일을 추가했다고 했을 때, 한 프로젝트의 마감일을 수정하면 해당 프로젝트의 다른 열들도 전부 수정해줘야 하는 문제가 생긴다.

사원아이디와 프로젝트아이디는 복합키이고, 마감일은 프로젝트아이디에만 관련이 있고 사원아이디와는 관련이 없다. (부분함수종속)
이럴 때 제2정규형에 위반이 된다.

어떤 열이 있을 때 복합키 모두에 속하는 게 아니면 제2정규형으로 만들어야한다.

| 아이디(기본키) | 프로젝트 | 마감일 |
|----------|------|------------|
| 1 | 홈페이지 | 2023-07-16 |
| 2 | AI | 2024-03-15 |

프로젝트 테이블

그래서 마감일 속성은 사원-프로젝트 테이블이 아닌, 프로젝트 테이블에 있어야 한다. (제2정규형)

제3정규형으로 만들기

| 아이디(수정X, 기본키) | 이름 | 이메일(unique, 키) | 연봉 | 팀 | 직책 | 최저연봉 |
|---------------|------|--|-------|------|----|------|
| 1 | 제로초 | zerohch1@gmail.com | 10000 | 개발팀 | 팀장 | 8000 |
| 2 | 원초 | onecho@gmail.com | 4000 | 개발팀 | 신입 | 3000 |
| 3 | 투초 | twocho@gmail.com | 6000 | 기획팀 | 과장 | 6000 |
| 4 | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 대리 | 4000 |
| 5 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | 과장 | 6000 |
| 6 | 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 인턴 | 2500 |
| 7 | 제로초 | zerocho@gmail.com | 8000 | 기획팀 | 팀장 | 8000 |

사원 테이블

사원 테이블에 최저연봉을 추가했다고 했을 때, 예를들어 팀장의 연봉을 수정하면 다른 팀장도 전부 수정해야 하는 문제가 생긴다.

이름은 직책과 관련있고, 직책은 최저연봉과 관련이 있으면, 이름은 최저연봉과 관련이 있다. (이행적 함수종속)
(이름->직책, 직책->최저연봉, 이름->최저연봉)

이렇게 삼단논법 형태일 때 제3정규형으로 만들어야 한다.

| 아이디(수정X, 기본키) | 이름 | 이메일(unique, 키) | 연봉 | 팀 | 직책(외래키) |
|---------------|------|--|-------|------|---------|
| 1 | 제로초 | zerohch1@gmail.com | 10000 | 개발팀 | 1 |
| 2 | 원초 | onecho@gmail.com | 4000 | 개발팀 | 4 |
| 3 | 투초 | twocho@gmail.com | 6000 | 기획팀 | 2 |
| 4 | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 3 |
| 5 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 5 |
| 7 | 제로초 | zerocho@gmail.com | 8000 | 기획팀 | 1 |

사원 테이블

| 아이디(기본키) | 직책 | 최저연봉 |
|----------|----|------|
| 1 | 팀장 | 8000 |
| 2 | 과장 | 6000 |
| 3 | 대리 | 5000 |
| 4 | 신입 | 3000 |
| 5 | 인턴 | 2500 |

직책 테이블

그래서 직책 테이블을 따로 만들어서 최저연봉을 관리해야 한다. (제3정규형)

역정규화

사원-프로젝트 테이블에서 사원의 직책을 알려면? 테이블을 여러 번 타고 들어가서 직책을 알아야 하므로 성능적으로 떨어진다.

| 사원아이디(외래키) | 프로젝트아이디(외래키) |
|------------|--------------|
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 3 | 1 |
| 3 | 2 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |

사원-프로젝트 테이블

| 아이디(수정X, 기본키) | 이름 | 이메일(unique, 키) | 연봉 | 팀 | 직책 |
|---------------|------|--------------------|-------|------|----|
| 1 | 제로초 | zerohch1@gmail.com | 10000 | 개발팀 | 1 |
| 2 | 원초 | onecho@gmail.com | 4000 | 개발팀 | 4 |
| 3 | 두초 | twocho@gmail.com | 6000 | 기획팀 | 2 |
| 4 | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | 3 |
| 5 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 5 |
| 7 | 제로초 | zerocho@gmail.com | 8000 | 기획팀 | 1 |

사원 테이블

| 아이디(기본키) | 직책 | 최저연봉 |
|----------|----|------|
| 1 | 팀장 | 8000 |
| 2 | 과장 | 6000 |
| 3 | 대리 | 5000 |
| 4 | 신입 | 3000 |
| 5 | 인턴 | 2500 |

직책 테이블

| 사원아이디(외래키) | 프로젝트아이디(외래키) | 직책아이디 |
|------------|--------------|-------|
| 1 | 1 | 1 |
| 1 | 2 | 1 |
| 2 | 1 | 4 |
| 3 | 1 | 2 |
| 3 | 2 | 2 |
| 4 | 1 | 3 |
| 5 | 1 | 2 |
| 6 | 1 | 5 |
| 7 | 2 | 1 |

사원-프로젝트 테이블

제2정규형 위반을 감수하는 대신 사원-프로젝트 테이블에 직책아이디를 뒀서 성능을 높이는 방식으로 가는 경우도 있다.

정규형 정리

제2정규형

전화번호는 이름이랑 관련이 있지, 언어랑은 관련되어 있지 않기에 제2정규형 위반

| 이름 | 언어 | 전화번호 |
|-----|------|---------------|
| 제로초 | JS | 010-1234-5678 |
| 제로초 | TS | 010-1234-5678 |
| 제로초 | JAVA | 010-1234-5678 |

| 이름 | 언어 |
|-----|------|
| 제로초 | JS |
| 제로초 | TS |
| 제로초 | JAVA |

| 이름 | 전화번호 |
|-----|---------------|
| 제로초 | 010-1234-5678 |

제3정규형

이름이 서비스와 관련이 있고, 서비스는 소속과 관련이 있으면,
이름은 소속과 관련이 있으므로 제3정규형 위반

| 아이디 | 이름 | 소속 | 서비스 |
|-----|-----|--------|-------|
| 1 | 제로초 | 네이버 | 네이버웹툰 |
| 2 | 원초 | 카카오 | 카카오톡 |
| 3 | 투초 | 우아한형제들 | 배달의민족 |

| 아이디 | 이름 | 서비스 |
|-----|-----|-------|
| 1 | 제로초 | 네이버웹툰 |
| 2 | 원초 | 카카오톡 |
| 3 | 투초 | 배달의민족 |

| 서비스(키) | 소속 |
|--------|--------|
| 네이버웹툰 | 네이버 |
| 카카오톡 | 카카오 |
| 배달의민족 | 우아한형제들 |

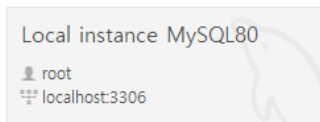
왜 소속이 아니라 서비스가 키?

오른쪽 테이블과 같이 서비스가 추가될 때면
소속(키)가 중복돼 버릴 수 있기 때문에 서비스
가 키가 되어야 한다.

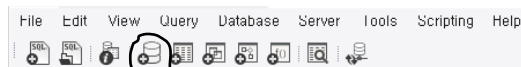
| 소속(키) | 서비스 |
|--------|-------|
| 네이버 | 네이버페이 |
| 네이버 | 네이버웹툰 |
| 카카오 | 카카오톡 |
| 우아한형제들 | 배달의민족 |

워크벤치로 스키마 만들기

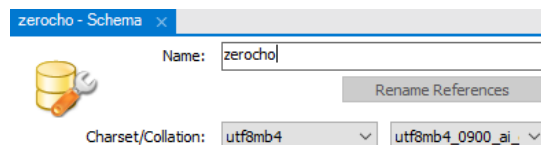
MySQL Connections



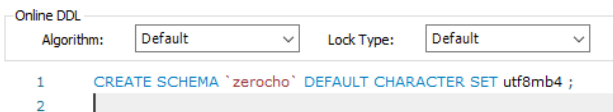
1. 워크벤치를 실행한 후 메인화면에서 이미 만들어져있는 Connection을 클릭



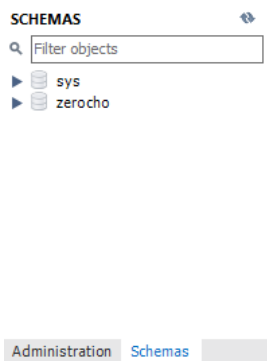
2. 해당 메뉴 버튼을 클릭



3. 위의 사진처럼 그대로 한 뒤 Apply 클릭
(utf8은 한글이나 유니코드를 호환하고 mb4는 이모지를 호환해준다.
Collation은 Default으로 해도 된다.)



4. 위의 사진처럼 이미 작성되어있는 코드는 SQL 문법에 따른 DDL(Data Definition Language)이며 스키마를 만드는 것이다. (이제부터 이런 코드의 형식들은 기억해두는 게 좋다) Apply를 클릭해서 다음으로 넘어간다.

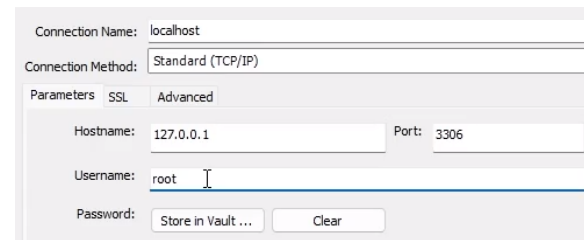


5. 왼쪽 네비게이터의 Schemas메뉴를 보면 스키마가 생성된 것을 볼 수 있다.
(일반적으로 데이터베이스->스키마->테이블 구조로 되어있지만 MySQL은 스키마와 데이터베이스를 같은 취급한다)

만약 메인화면에 Connection이 없다면?

MySQL Connections

만약 워크벤치를 처음 실행 했을 때 Connection이 없으면 +버튼을 누른다.



포트는 3306으로 맞추고 Store in Vault를 클릭해서 비밀번호를 입력하고 Test Connection을 눌러 성공인지 확인한 다음 OK를 클릭한다.

※ 워크벤치는 MySQL 데이터베이스만 사용할 수 있다.
(그러나 MariaDB는 예외이다. MySQL이랑 같은 제작사가 만들었으므로)

앞으로 만들 테이블 정리

| 아이디(수정X, 기본키) | 이름 | 이메일(unique, 키) | 연봉 | 팀 | 직책(외래키) | 퇴사일 |
|---------------|------|--|-------|------|---------|-----|
| 1 | 제로초 | zerohch1@gmail.com | 10000 | 개발팀 | | 1 |
| 2 | 원초 | onecho@gmail.com | 4000 | 개발팀 | | 4 |
| 3 | 투초 | twocho@gmail.com | 6000 | 기획팀 | | 2 |
| 4 | 쓰리초 | threecho@gmail.com | 5000 | 디자인팀 | | 3 |
| 5 | 포초 | fourcho@gmail.com | 8000 | 개발팀 | | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 3000 | 개발팀 | 🔄 | 5 |
| 7 | 제로초 | zerocho@gmail.com | 8000 | 개발팀 | | 1 |

사원 테이블

| 사원아이디(외래키) | 프로젝트아이디(외래키) |
|------------|--------------|
| 1 | 1 |
| 1 | 2 |
| 2 | 1 |
| 3 | 1 |
| 3 | 2 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |

사원-프로젝트 테이블

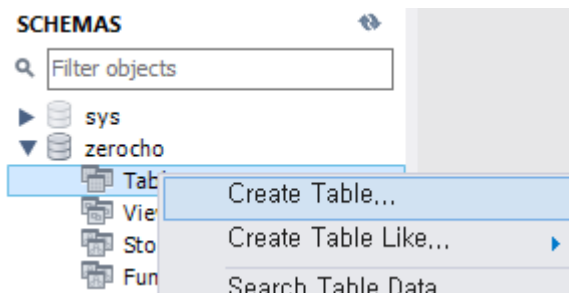
| 아이디(기본키) | 프로젝트 | 마감일 |
|----------|------|------------|
| 1 | 홈페이지 | 2023-07-16 |
| 2 | AI | 2024-03-15 |

프로젝트 테이블

| 아이디(기본키) | 직책 | 최저연봉 |
|----------|----|----------------------|
| 1 | 팀장 | 8000 |
| 2 | 과장 | 6000 |
| 3 | 대리 | 5000 |
| 4 | 산업 | 3000 |
| 5 | 인턴 | 2500 |

직책 테이블

워크벤치로 직책 테이블 만들기






1. Tables에 마우스 오른쪽 클릭하고
Create Table로 테이블을 생성 할 수 있다.



Table Name: Schema: **zerocho**
Charset/Collation: Engine:

2. 기본적으로 테이블을 만들 때는 FK가 없는 것부터 먼저 만들어야 하기 때문에
우선 직책 테이블부터 만들기로 하고, 위 사진처럼 설정한다.
(Engine는 대부분 InnoDB랑 MyISAM을 많이 쓴다)

| Column Name | Datatype | PK | NN | UQ | B | UN | ZF | AI | G | Default/Expression |
|--|-------------|-------------------------------------|-------------------------------------|-------------------------------------|--------------------------|-------------------------------------|--------------------------|-------------------------------------|--------------------------|--------------------|
|  id | INT(11) | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | |
|  name | VARCHAR(10) | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
|  min_salary | INT | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | 2500 |

직책 테이블

3. 사진과 같이 컬럼을 만들 수 있으며 자료형과 옵션을 설정할 수 있으며 완성 후 Apply를 클릭한다.

자료형

INT: 정수를 의미하며 INT(11)이런식으로 자릿수를 설정할 수 있다. (11자리)
VARCHAR(): 문자열이며 지정한 용량이 가변적이다.
TEXT(): 게시글처럼 긴 데이터는 VARCHAR() 대신에 쓴다.
CHAR(): 문자를 저장하며 지정된 용량이 고정적이다.
DATE: 날짜만 저장한다.
DATETIME(): 날짜와 시간을 저장한다.
BOOLEAN: 숫자 0, 1만 저장 가능하다.
TINYINT(1): BOOLEAN이 없었을 옛날에 썼던 것이며 0, 1만 저장가능하다

옵션

PK: 기본키, NN은 Not Null의 약자이며 값이 필수여야 하는 의미이다.
UQ: Unique의 약자이며 고유함을 보장한다.(PK는 굳이 체크 안해도 됨)
B: Binary의 약자이며 데이터들이 이진수일 때 사용한다.
UN: Unsigned의 약자이며 음수를 허용 안하는 대신 더 많은 데이터 저장 가능하다.
ZF: Zero Fill의 약자이며 남은 용량을 0로 채우겠다는 의미이다.
AI: Auto Increment의 약자이며 아이디를 매겨줄 때 자동으로 증가시켜준다.
G: Generated의 약자이며 다른 컬럼 값에 의존한다는 의미이다.
Default/Expression: 아무 값도 안넣었을 때 기본값을 지정할 수 있다.

직책 테이블 DDL

Online DDL

Algorithm: Default Lock Type: Default

```
1 CREATE TABLE `zerocho`.`role` (  
2   `id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT COMMENT '직책 아이디',  
3   `name` VARCHAR(10) NOT NULL COMMENT '직책 이름',  
4   `min_salary` INT UNSIGNED NOT NULL DEFAULT 2500 COMMENT '최저연봉',  
5   PRIMARY KEY (`id`),  
6   UNIQUE INDEX `id_UNIQUE` (`id` ASC) VISIBLE,  
7   UNIQUE INDEX `name_UNIQUE` (`name` ASC) VISIBLE)  
8 ENGINE = InnoDB  
9 DEFAULT CHARACTER SET = utf8mb4  
10 COMMENT = '직책 테이블';  
11
```

Apply를 클릭하고 나면 워크벤치가 자동으로 SQL을 만들어 준다.

MySQL에서 문자열들은 ` (백틱)으로 감싸주며, 안감싸도 된다. (백틱은 예약어나 따옴표나 띄어쓰기에도 대응이 되기 때문에 안전하다)

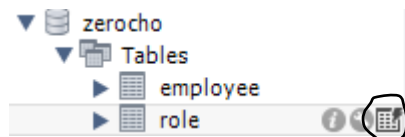
UNIQUE 고유한 값을 의미하고 INDEX는 정렬과 검색에 자주 쓰이기 때문에 INDEX로 만들어진 것이다. (기본적으로 UQ는 자동으로 INDEX를 만들어 준다)

INDEX 뒤에 있는 문자열은 INDEX 이름이며, 겹치지만 않으면 마음대로 수정해도 된다.

ASC는 오름차순으로 정렬을 의미한다. (DESC는 내림차순)

VISIBLE은 작동하는 INDEX를 의미한다.

직책 테이블 데이터 넣어보기(DML)



1. role에서 세번째 아이콘 클릭

| Result Grid | | | |
|-------------|------|------|------------|
| Filter Rows | | | |
| | id | name | min_salary |
| | NULL | 팀장 | 8000 |
| | NULL | 과장 | 6000 |
| | NULL | 대리 | 5000 |
| | NULL | 신입 | 3000 |
| | NULL | 인턴 | 2500 |

2. 아래에 이런식으로 데이터를 추가하고 Apply를 클릭한다.
(id는 AI(AutoIncrement)를 설정 했으므로 데이터를 넣을 필요가 없다.)

Review the SQL Script to be Applied on the Database

```
1  INSERT INTO `zerocho`.`role` (`name`, `min_salary`) VALUES ('팀장', '8000');
2  INSERT INTO `zerocho`.`role` (`name`, `min_salary`) VALUES ('과장', '6000');
3  INSERT INTO `zerocho`.`role` (`name`, `min_salary`) VALUES ('대리', '5000');
4  INSERT INTO `zerocho`.`role` (`name`, `min_salary`) VALUES ('신입', '3000');
5  INSERT INTO `zerocho`.`role` (`name`, `min_salary`) VALUES ('인턴', '2500');
6
```

3. 마찬가지로 Apply를 클릭하고나면 워크벤치가
SQL을 자동으로 생성해준다.

Review the SQL Script to be Applied on the Database

```
1  INSERT INTO `zerocho`.`role` (`name`, `min_salary`) VALUES ('팀장', '8000'),
2  ('과장', '6000'), ('대리', '5000'), ('신입', '3000'), ('인턴', '2500');
3
```

이렇게 한줄로 만들 수도 있다. (참고)

워크벤치로 사원 테이블 만들기

| Column Name | Datatype | PK | NN | UQ | B | UN | ZF | AI | G | Default/Expression |
|-------------|--------------|-------------------------------------|-------------------------------------|-------------------------------------|--------------------------|-------------------------------------|--------------------------|-------------------------------------|--------------------------|--------------------|
| id | INT | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | |
| name | VARCHAR(45) | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| email | VARCHAR(100) | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| salary | INT | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| team | VARCHAR(20) | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| quit_date | DATETIME | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |
| created_at | DATETIME | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | CURRENT_TIMESTAMP |
| role_id | INT | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | |

사원 테이블

기본값으로 CURRENT_TIMESTAMP를 하면 현재시간을 뜻한다. (NOW()로도 가능)

role_id 외래키를 구현하려면?

Columns Indexes **Foreign Keys** Triggers Partitioning Options

1. 외래키를 구현하려면 아래 Foreign Keys 탭을 누른다.

| Foreign Key Name | Referenced Table |
|------------------|---|
| employee_role_fk | <div><div>`zerocho`.`role`</div><div>`zerocho`.`employee`</div><div>`zerocho`.`role`</div><div>Specify Manually...</div></div> |

2. 외래키 이름을 정하고 참조할 테이블을 선택한다.

| Column | Referenced Column |
|---|-------------------|
| <input type="checkbox"/> id | |
| <input type="checkbox"/> name | |
| <input type="checkbox"/> email | |
| <input type="checkbox"/> salary | |
| <input type="checkbox"/> team | |
| <input type="checkbox"/> quit_date | |
| <input type="checkbox"/> created_at | |
| <input checked="" type="checkbox"/> role_id | id |

3. 외래키를 구현시킬 속성을 체크하고 참조 컬럼을 선택한다.

Foreign Key Options

On Update:

CASCADE

On Delete:

CASCADE

☐ Skip in SQL generation

NO ACTION은 자식이 기본키를 참조하고 있을 때는 기본키를 수정하지 못한다. (RESTRICT와 동일)

CASCADE는 기본키를 수정하면 참조하고 있는 외래키도 수정된다.

SET NULL은 기본키를 수정하면 참조하고 있는 외래키가 비워지며 끊어진다.

Foreign Key Options

On Update:

CASCADE

On Delete:

SET NULL

☐ Skip in SQL generation

NO ACTION은 자식이 기본키를 참조하고 있을 때는 기본키를 삭제하지 못한다. (RESTRICT와 동일)

CASCADE는 기본키를 삭제하면 참조하고 있는 외래키의 레코드가 통째로 삭제된다.

SET NULL은 기본키를 삭제하면 참조하고 있는 외래키가 삭제된다.

사원테이블 DDL

Online DDL

Algorithm: Lock Type:

```
1 CREATE TABLE `zerocho`.`employee` (  
2   `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,  
3   `name` VARCHAR(45) NOT NULL,  
4   `email` VARCHAR(100) NOT NULL,  
5   `salary` INT UNSIGNED NOT NULL,  
6   `team` VARCHAR(20) NOT NULL,  
7   `quit_date` DATETIME NULL,  
8   `created_at` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,  
9   `role_id` INT UNSIGNED NULL,  
10  PRIMARY KEY (`id`),  
11  UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE,  
12  INDEX `employee_role_fk_idx` (`role_id` ASC) VISIBLE,  
13  CONSTRAINT `employee_role_fk`  
14    FOREIGN KEY (`role_id`)  
15    REFERENCES `zerocho`.`role` (`id`)  
16    ON DELETE NO ACTION  
17    ON UPDATE NO ACTION)
```

CONSTRAINT는 employee_role_fk라는 이름으로 된 제약이며 FOREIGN KEY로 외래키를 지정하고 REFERENCES로 role_id 컬럼을 role 테이블의 id 컬럼을 참조하도록 한다.

사원 테이블에 데이터 넣어보기(INSERT INTO, DML)

| id | name | email | salary | team | quit_date | created_at | role_id |
|----|------|--------------------|--------|------|-----------|------------|---------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | | 1 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | | 1 |
| 3 | 두초 | twocho@gmail.com | 8000 | 기획팀 | NULL | | 1 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | | 2 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | | 3 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | | 3 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | | 4 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | | 4 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | | 4 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | | 5 |

(ID는 Auto_Increment 옵션을 설정 했으므로 값 넣을 때 비워도 된다)

```

1 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('제로초', 'zerocho@gmail.com', '10000', '개발팀', '1');
2 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('원초', 'onecho@gmail.com', '6000', '디자인팀', '1');
3 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('두초', 'twocho@gmail.com', '8000', '기획팀', '1');
4 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('쓰리초', 'threecho@gmail.com', '7000', '기획팀', '2');
5 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('포초', 'fourcho@gmail.com', '9000', '개발팀', '2');
6 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('파이브초', 'fivecho@gmail.com', '6000', '기획팀', '3');
7 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('식스초', 'sixcho@gmail.com', '6000', '개발팀', '3');
8 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('세븐초', 'sevencho@gmail.com', '5000', '개발팀', '4');
9 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('에잇초', 'eightcho@gmail.com', '4000', '디자인팀', '4');
10 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('나인초', 'ninecho@gmail.com', '3000', '개발팀', '4');
11 • INSERT INTO zerocho.`employee` (`name`, email, salary, team, role_id) VALUES ('텐초', 'tencho@gmail.com', '2500', '기획팀', '5');

```

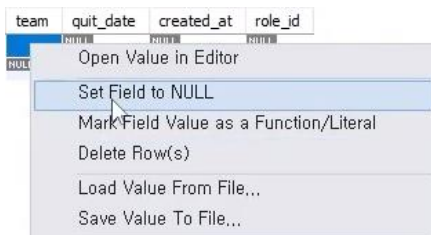
SQL문은 번개 아이콘을 눌러서 적용할 수 있다. →



참고

| id | name | email | salary | team | quit_date | created_at | role_id |
|------|------|--------------------|--------|------|-----------|------------|---------|
| NULL | 제로초 | zerohcho@gmail.com | 10000 | NULL | NULL | NULL | NULL |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL | NULL |

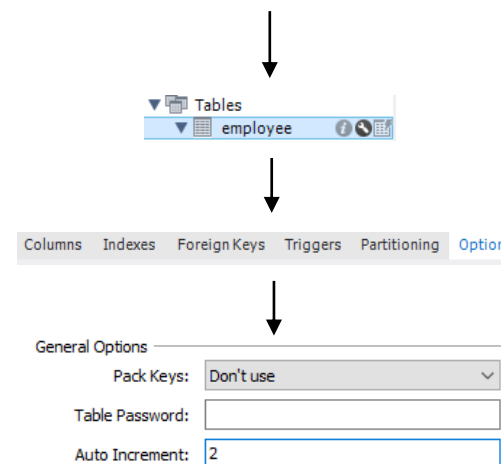
만약 team 속성에 아무것도 없을 때 NULL로 되게 하고 싶은데 계속 빈문자열로 되어 있으면



해당 속성을 Set Field to NULL로 설정한다

| id | name | email | salary | team | quit_date | created_at | role_id |
|----|------|--------------------|--------|------|-----------|---------------------|---------|
| 2 | 제로초 | zerohcho@gmail.com | 10000 | 개발팀 | NULL | 2023-06-20 21:57:11 | 1 |

Auto_Increment는 한번 쿼리를 실패해도 id가 count 돼서 다음 레코드의 id가 증가되어있다. 이를 바꾸려면



Auto Increment를 수정하고 apply

나머지 테이블 SQL

프로젝트 테이블

```
CREATE TABLE `zerocho`.`project` (  
  `id` INT UNSIGNED NOT NULL AUTO_INCREMENT,  
  `name` VARCHAR(45) NOT NULL,  
  `due_date` DATE NULL DEFAULT NULL,  
  `created_at` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,  
  PRIMARY KEY (`id`))  
COMMENT = '프로젝트 테이블';
```

```
INSERT INTO `zerocho`.`project` (`name`, `due_date`)  
VALUES ('홈페이지', '2023-07-15');
```

```
INSERT INTO `zerocho`.`project` (`name`, `due_date`)  
VALUES ('AI', '2024-01-31');
```

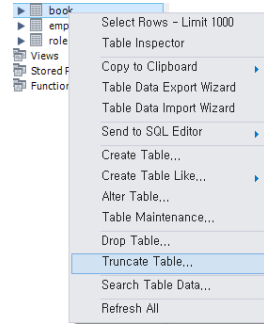
사원-프로젝트 테이블

```
CREATE TABLE `zerocho`.`employee_project` (  
  `employee_id` INT UNSIGNED NOT NULL,  
  `project_id` INT UNSIGNED NOT NULL,  
  PRIMARY KEY (`employee_id`, `project_id`),  
  INDEX `project_fk_idx` (`project_id` ASC) VISIBLE,  
  CONSTRAINT `employee_fk`  
    FOREIGN KEY (`employee_id`)  
      REFERENCES `zerocho`.`employee` (`id`)  
      ON DELETE CASCADE  
      ON UPDATE CASCADE,  
  CONSTRAINT `project_fk`  
    FOREIGN KEY (`project_id`)  
      REFERENCES `zerocho`.`project` (`id`)  
      ON DELETE CASCADE  
      ON UPDATE CASCADE)  
COMMENT = '사원-프로젝트 테이블';
```

```
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('1', '1');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('1', '2');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('2', '1');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('3', '2');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('4', '2');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('5', '2');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('6', '1');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('6', '2');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('7', '1');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('8', '1');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('9', '2');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('10', '2');  
INSERT INTO `zerocho`.`employee_project` (`employee_id`, `project_id`) VALUES ('11', '1');
```


Tip

레코드 전부 삭제

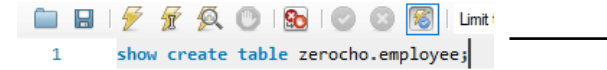


TRUNCATE 'zerocho'.book

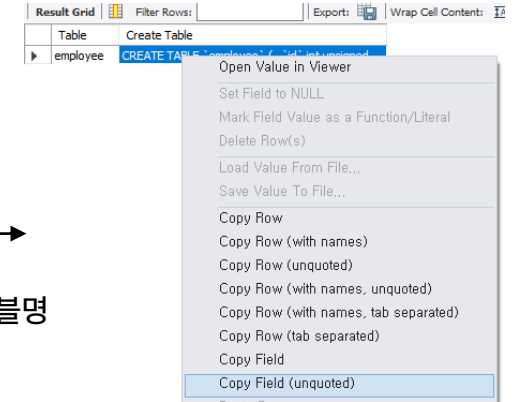
SQL

Truncate Table하면 해당 테이블의 모든 레코드를 지운다.
(Drop Table은 테이블 자체를 삭제)

DDL 복사

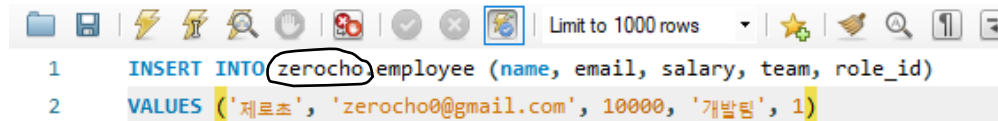


1. 스크립트에서 show create table 스키마명.테이블명
입력 후 번개 아이콘 클릭



2. 아래 리스트에서 해당 테이블을 마우스 오른쪽 클릭
한 후 Copy Field (unquoted) 하면 DDL이 복사가 된다.
(ERDCloud 같은 곳으로 내보내기 해서 활용 가능)

스키마명 생략

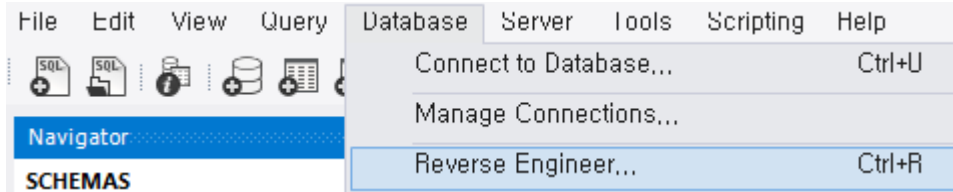


명령어마다 스키마를 명시하고 싶지 않다면

```
1 • USE zerocho;
2 • INSERT INTO employee (name, email, salary, team, role_id)
3 • VALUES ('제로초', 'zerocho0@gmail.com', 10000, '개발팀', 1);
```

앞에 USE 스키마명을 쓰면 된다

ERD 만들기



1. 메뉴에서 Database -> Reverse Engineer

Set Parameters for Connecting to a DBMS

Stored Connection: Local instance MySQL80 Select from saved connection settings

Connection Method: Standard (TCP/IP) Method to use to connect to the RDBMS

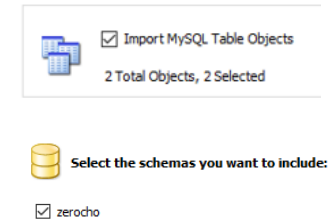
Parameters SSL Advanced

Hostname: localhost Port: 3306 Name or IP address of the server host - and TCP/IP port.

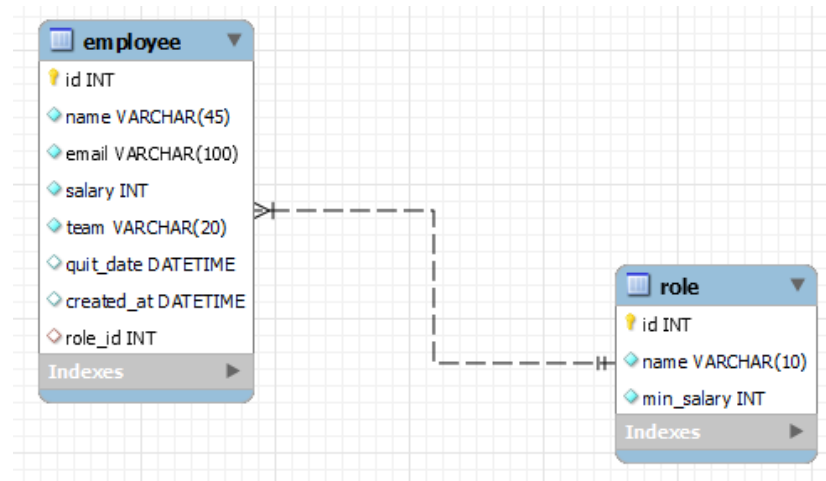
Username: root Name of the user to connect with.

Password: Store in Vault ... Clear The user's password. Will be requested later if it's not set.

2. 위 사진처럼 설정하고 계속 Next

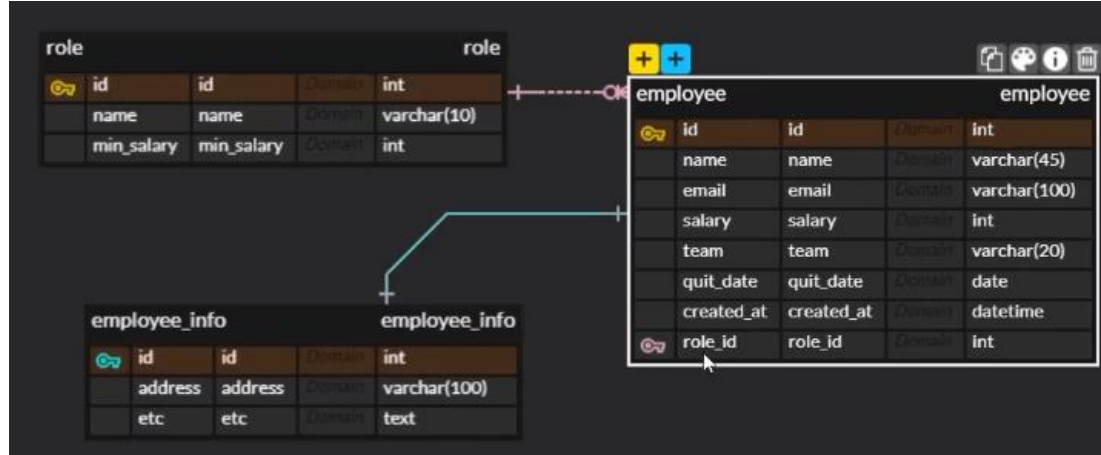


Next 과정에서 위 두개는 체크 된 상태로



3. ERD가 그려진다.

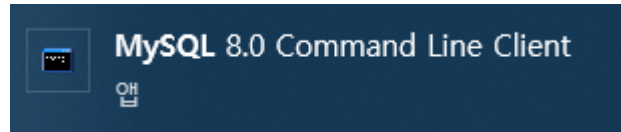
식별 관계 vs 비식별 관계



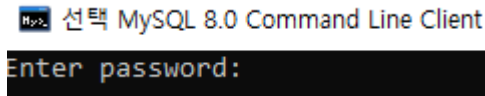
employee 테이블의 role_id는 role 테이블의 id를 참조하고있다.
이렇게 참조관계일 때는 비식별 관계이다.

employee_info는 employee의 id를 그대로 물려받으면서 1:1 관계이다. 이럴 때는 식별 관계이다.

MySQL 프롬프트



1. 워크벤치 말고도 MySQL 프롬프트를 통해 데이터베이스를 관리 할 수 있다.



2. 실행하고 나서 패스워드를 입력한다.
(처음엔 Root로 로그인)

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| sys |
| zerocho |
+-----+
5 rows in set (0.00 sec)
```

3. 데이터베이스 목록 출력
- show databases;

```
mysql> show tables;
+-----+
| Tables_in_zerocho |
+-----+
| employee |
| role |
+-----+
2 rows in set (0.00 sec)
```

6. 들어간 DB 안에 저장된 테이블 목록 출력
- show tables;

```
mysql> select database();
+-----+
| database() |
+-----+
| zerocho |
+-----+
1 row in set (0.00 sec)
```

5. 현재 어떤 DB에 접속되어 있는지 확인
- select database();

```
mysql> use zerocho
Database changed
```

4. 특정 DB 안으로 들어가기
- use [데이터베이스 이름];

```
mysql> desc employee;
+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+
| id | int unsigned | NO | PRI | NULL | auto_increment |
| name | varchar(45) | NO | | NULL | |
| email | varchar(100) | NO | UNI | NULL | |
| salary | int unsigned | NO | | NULL | |
| team | varchar(20) | NO | | NULL | |
| quit_date | datetime | YES | | NULL | |
| created_at | datetime | YES | | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
| role_id | int unsigned | YES | MUL | NULL | |
+-----+
8 rows in set (0.00 sec)
```

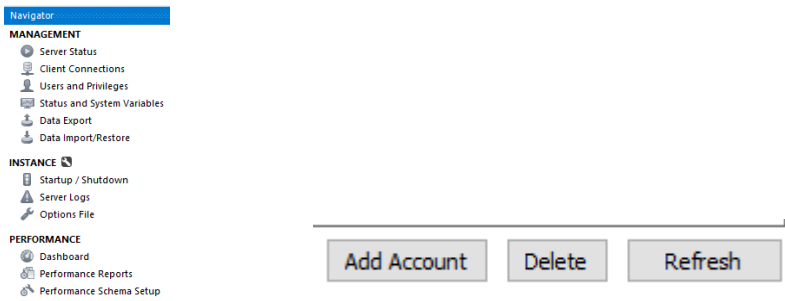
6. 특정 table 형식 출력
- desc [테이블 이름]
- describe [테이블 이름]

```
mysql> select * from employee;
+----+-----+-----+-----+-----+-----+-----+-----+
| id | name | email | salary | team | quit_date | created_at | role_id |
+----+-----+-----+-----+-----+-----+-----+-----+
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 3 | 두초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 |
+----+-----+-----+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

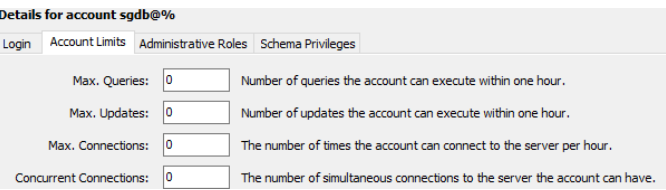
7. 특정 table 안에 저장된 모든 데이터 출력
- select * from [테이블 이름];

MySQL 권한 관리 (참고)

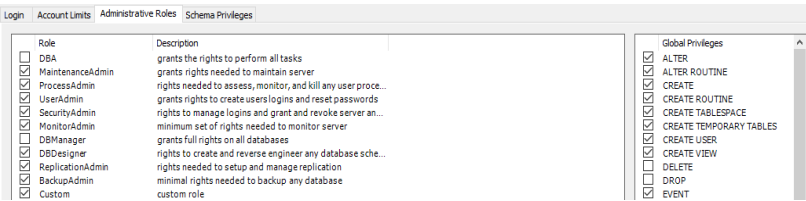
DB를 관리하다 보면 데이터를 전부 지워버리는 실수를 할 수도 있기 때문에 권한을 설정할 수 있다.



2. Add Account를 클릭한 다음 Login Name와 Password를 설정한다.

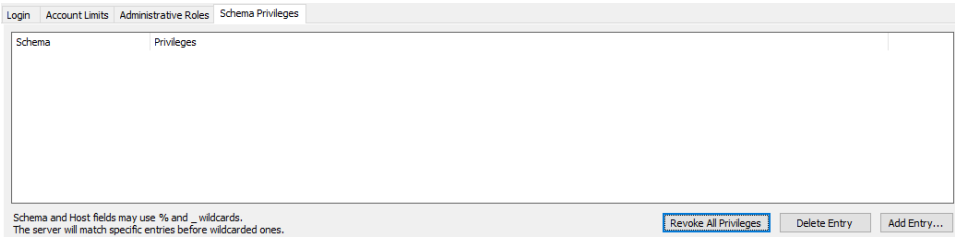


3. Account Limits에서 쿼리를 동시에 몇 개 보낼 수 있는지 제한 가능

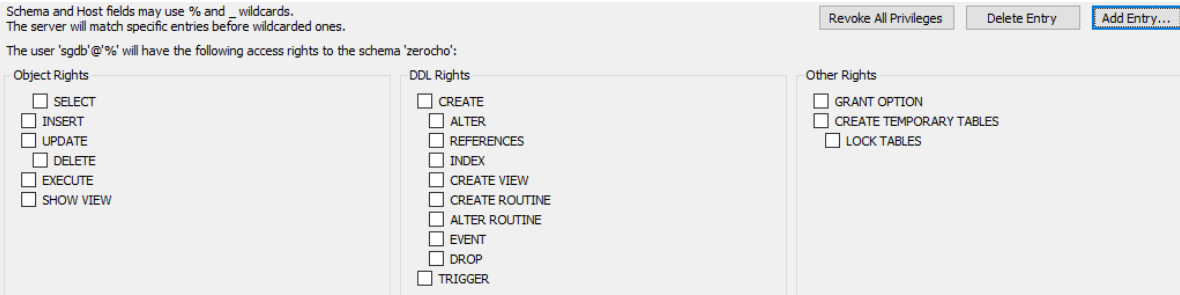


3. Administration Roles 탭에서 체크 해제해서 특정 쿼리를 제한 할 수 있다.

1. Administration 탭에서 Users and Privileges 클릭



4. Add Entry를 클릭해서 스키마를 생성하거나 선택할 수 있다.



5. 여기서 스키마별로 권한을 제어할 수 있다.

CREATE TABLE (SQL 연습)

```
create table book1 (  
    book_id int,  
    book_name varchar(20),  
    author varchar(20),  
    publisher varchar(20),  
    price integer);
```

```
MariaDB [sgdb]> desc book1;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | YES | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| author | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

5 rows in set (0.014 sec)

```
create table book4 (  
    book_id integer,  
    book_name varchar(20),  
    author varchar(20),  
    publisher varchar(20),  
    price int,  
    primary key(book_id, book_name)); // book_id와 book_name 둘다  
                                     기본키로 등록
```

```
MariaDB [sgdb]> desc book4;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | NO | PRI | NULL | |
| book_name | varchar(20) | NO | PRI | NULL | |
| author | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

5 rows in set (0.011 sec)

primary key를 각각 두개의 필드에 입력하면 두개로 묶여지는 기본키가 안된다.

```
create table book2 (  
    book_id int,  
    book_name varchar(20),  
    author varchar(20),  
    publisher varchar(20),  
    price int,  
    primary key (book_id)); // book_id 필드를 기본키로 등록
```

```
MariaDB [sgdb]> desc book2;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | NO | PRI | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| author | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

5 rows in set (0.011 sec)

```
create table book5 (  
    book_id integer,  
    book_name varchar(20) not null,  
    author varchar(20),  
    publisher varchar(20) unique,  
    price integer default 10000 check(price >= 1000), // price가 1000보다 작으면  
    primary key (book_id));                          10000이 채워지지 않는다
```

```
MariaDB [sgdb]> desc book5;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | NO | PRI | NULL | |
| book_name | varchar(20) | NO | | NULL | |
| author | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | UNI | NULL | |
| price | int(11) | YES | | 10000 | |

5 rows in set (0.011 sec)

```
create table book3 (  
    book_id int primary key, // 이렇게도 기본키 등록이 가능  
    book_name varchar(20),  
    author varchar(20),  
    publisher varchar(20),  
    price int);
```

```
MariaDB [sgdb]> desc book3;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | NO | PRI | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| author | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

5 rows in set (0.009 sec)

```
create table book7(  
    -> name varchar(20),  
    -> gender varchar(2) check(gender = '남자' or gender='여자'));  
  
create table book8(  
    -> price int check(price >= 10000 and price <= 30000),  
    -> gender varchar(2) check(gender = '남자' or gender = '여자'));
```

논리 기호를 이용해서도 조건식을 적을 수 있다

ALTER TABLE (SQL 연습)

첫 테이블

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | YES | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

4 rows in set (0.011 sec)

1. 새로운 필드 추가 : alter table book6 add author integer;

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | YES | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |
| author | int(11) | YES | | NULL | |

5 rows in set (0.011 sec)

2. 필드의 타입 변경 : alter table book6 modify author varchar(20);

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | YES | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |
| author | varchar(20) | YES | | NULL | |

5 rows in set (0.021 sec)

3. 기존 필드 삭제 : alter table book6 drop column author;

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | YES | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

4 rows in set (0.011 sec)

4. NULL을 허용하지 않도록 수정 : alter table book6 modify book_id integer not null;

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | NO | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

4 rows in set (0.011 sec)

5. NULL을 다시 허용하도록 수정 : alter table book6 modify book_id integer;

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | YES | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

4 rows in set (0.011 sec)

ALTER TABLE (SQL 연습)

첫 테이블

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | YES | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

4 rows in set (0.011 sec)

1. book_id 필드를 기본키로 설정 : alter table book6 add primary key(book_id);

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | NO | PRI | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

4 rows in set (0.011 sec)

2. publisher 필드명을 pub로 변경 : alter table book6 change publisher pub varchar(20);

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | NO | PRI | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| pub | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

4 rows in set (0.011 sec)

3. 기본키 속성 삭제 : alter table book6 drop primary key;

(기본키를 삭제해도 NULL은 그대로 NO로 되어있다)

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | NO | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| pub | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

4 rows in set (0.016 sec)

4. 기본값 추가 또는 변경 : alter table book6 alter column price set default 10000;

```
MariaDB [sgdb]> desc book6;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | NO | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| pub | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | 10000 | |

4 rows in set (0.012 sec)

INSERT INTO (SQL 연습)

기본 필드

```
MariaDB [sgdb]> desc book1;
```

| Field | Type | Null | Key | Default | Extra |
|-----------|-------------|------|-----|---------|-------|
| book_id | int(11) | YES | | NULL | |
| book_name | varchar(20) | YES | | NULL | |
| author | varchar(20) | YES | | NULL | |
| publisher | varchar(20) | YES | | NULL | |
| price | int(11) | YES | | NULL | |

5 rows in set (0.012 sec)

1. insert into book1 (book_id, book_name, author, publisher, price)
values (10, 'book1', 'author1', 'pub1', 2000);

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|---------|-----------|-------|
| 10 | book1 | author1 | pub1 | 2000 |

1 row in set (0.000 sec)

2. insert into book1
values(20, 'book2', 'author2', 'pub2', 3500);
// 두번째로 데이터를 넣을 때는 필드 목록을 안써도 된다

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|---------|-----------|-------|
| 10 | book1 | author1 | pub1 | 2000 |
| 20 | book2 | author2 | pub2 | 3500 |

2 rows in set (0.000 sec)

3. insert into book1 (book_id, book_name, price)
values (30, 'book3', 2500);
// 특정 필드에 데이터를 넣기. 안들어간 필드는 해당 데이터에 NULL로 되어있다.

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|---------|-----------|-------|
| 10 | book1 | author1 | pub1 | 2000 |
| 20 | book2 | author2 | pub2 | 3500 |
| 30 | book3 | NULL | NULL | 2500 |

3 rows in set (0.000 sec)

4. insert into book1
values (40, null, 'author4', null, 1900);
// 필드 목록은 안쓰고 바로 특정 필드에 데이터를 넣을 때는 안쓸 필드 데이터에 NULL를 쓴다.

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|---------|-----------|-------|
| 10 | book1 | author1 | pub1 | 2000 |
| 20 | book2 | author2 | pub2 | 3500 |
| 30 | book3 | NULL | NULL | 2500 |
| 40 | NULL | author4 | NULL | 1900 |

4 rows in set (0.000 sec)

5. delete from book1;
// 테이블은 그대로 두고 안에 있는 레코드들을 모두 지운다

UPDATE (SQL 연습)

첫 테이블

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|---------|-----------|-------|
| 10 | book1 | author1 | pub1 | 2000 |

```
1 row in set (0.000 sec)
```

1. update book1 set price = 20000;
// price 값을 20000으로 업데이트 한다.

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|---------|-----------|-------|
| 10 | book1 | author1 | pub1 | 20000 |

```
1 row in set (0.000 sec)
```

2. update book1 set author='홍길동', publisher='신구문화사';
// 여러개로 데이터를 업데이트 한다.

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|--------|-----------|-------|
| 10 | book1 | 홍길동 | 신구문화사 | 20000 |

```
1 row in set (0.000 sec)
```

3. insert into book1 values(20, 'book2', '이순신', '조선출판사', 50000);

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|--------|-----------|-------|
| 10 | book1 | 홍길동 | 신구문화사 | 20000 |
| 20 | book2 | 이순신 | 조선출판사 | 50000 |

```
2 rows in set (0.000 sec)
```

4. update book1 set price = 60000;
// 이렇게 하면 price 필드의 모든 레코드들이 60000으로 바뀌어서 주의해야한다.

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|--------|-----------|-------|
| 10 | book1 | 홍길동 | 신구문화사 | 60000 |
| 20 | book2 | 이순신 | 조선출판사 | 60000 |

```
2 rows in set (0.000 sec)
```

6. update book1 set price = 20000 where book_id = 10;
// 모든 레코드들을 바꾸지 않고 book_id가 10인 레코드에서만 price를 20000으로 바꾼다.

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|--------|-----------|-------|
| 10 | book1 | 홍길동 | 신구문화사 | 20000 |
| 20 | book2 | 이순신 | 조선출판사 | 60000 |

```
2 rows in set (0.000 sec)
```

7. delete from book1 where book_id = 10;
// book_id가 10인 레코드만 지운다. where를 안쓰면 모든 레코드가 지워진다.

```
MariaDB [sgdb]> select * from book1;
```

| book_id | book_name | author | publisher | price |
|---------|-----------|--------|-----------|-------|
| 20 | book2 | 이순신 | 조선출판사 | 60000 |

```
1 row in set (0.000 sec)
```

외부로부터 DB 추가

이제부터 많은 데이터를 통한 SQL을 연습 할 것이므로 외부로부터 DB를 추가해보자

DB 다운로드 링크: https://drive.google.com/file/d/1ay3p0kDfT2j0G3SQ6CA16WzG8X_mvcGc/view?usp=drive_link

프롬프트에서 DB 추가 방법

```
MySQL 8.0 Command Line Client
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 32
Server version: 8.0.34 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> source c:\db.sql_
```

source [DB 경로]

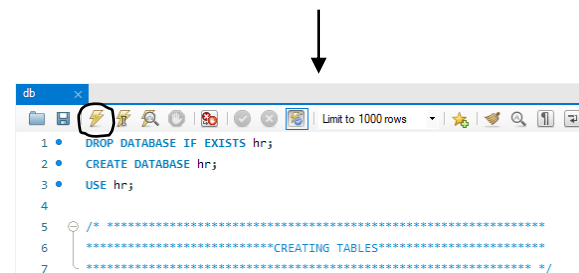
```
mysql> show databases;
+-----+
| Database |
+-----+
| hr        |
| information_schema |
| mysql     |
| performance_schema |
| sys       |
| zerocho   |
+-----+
6 rows in set (0.00 sec)
```

hr이라는 DB가 추가된 것을 볼 수 있다.

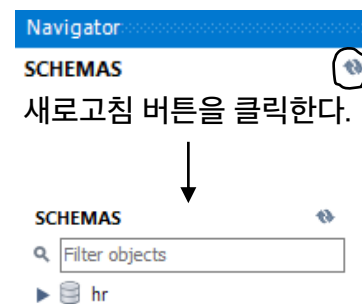
워크벤치에서 DB 추가 방법



메뉴에서 해당 아이콘을 클릭해서 SQL파일을 추가한다.



번개 아이콘을 클릭해서 스크립트를 적용시킨다.



hr이라는 스키마가 추가된 걸 볼 수 있다.

SELECT

1. salary가 10000이상인 employee_id, last_name 필드의 데이터(레코드)

```
select employee_id, last_name  
from employees  
where salary >= 10000;
```

2. salary가 5000 미만인 employee_id, first_name, salary 필드의 데이터(레코드)

```
select employee_id, first_name, salary  
from employees  
where salary < 5000;
```

3. salary가 10000이상이고 15000 미만인 employee_id, last_name, salary 필드의 데이터(레코드)

```
select employee_id, last_name, salary  
from employees  
where salary >= 10000 and salary < 15000;
```

4. salary가 15000이상이거나 5000 이하인 first_name, last_name, salary, hire_date 필드의 데이터(레코드)

```
select first_name, last_name, salary, hire_date  
from employees  
where salary >= 15000 or salary <= 5000;
```

5. hire_date가 1995년 1월 5일 이후인 first_name, last_name, hire_date 필드의 데이터(레코드)

```
select first_name, last_name, hire_date  
from employees  
where hire_date > '1995-01-05'; // 따옴표는 필수적으로 붙여야한다.
```

6. hire_date가 1995년 1월 5일 이후이면서 salary가 10000이상인 모든 필드의 데이터(레코드)

```
select *  
from employees  
where hire_date > '1995-01-05' and salary >= 10000;
```

SELECT

% : 개수에 상관 없이 아무 글자가 나와도 된다.

_ : 1개의 아무 글자가 나와도 된다.

1. last_name이 k로 시작하는 모든 필드의 데이터(레코드)

```
select *  
from employees  
where last_name like 'k%';      // k로 끝나는걸 찾을 때는 %k로 한다.
```

2. last_name이 4글자인 모든 필드의 데이터(레코드)

```
select *  
from employees  
where last_name like '____';    // 언더바 4개
```

3. last_name에 ei가 들어가는 모든 필드의 데이터(레코드)

```
select *  
from employees  
where last_name like '%ei%';
```

4. last_name이 t로 시작하면서 r로 끝나는 모든 필드의 데이터(레코드)

```
select *  
from employees  
where last_name like 't%r';
```

5. last_name이 t로 시작하면서 6글자인 모든 필드의 데이터(레코드)

```
select *  
from employees  
where last_name like 't_____';
```

SELECT

1. last_name, commission_pct 필드의 데이터를 앞에서부터 50개까지의 데이터(레코드)

```
select last_name, commission_pct from employees limit 0, 50;
```

2. last_name, commission_pct 필드의 데이터를 앞에서부터 4개의 레코드를 건너뛰고 2개를 출력 (페이지네이션에 활용)

```
select last_name, commission_pct from employees limit 2 offset 4;
```

3. commission_pct 필드의 값이 없는 모든 필드의 데이터(레코드)

```
select * from employees where commission_pct is null;
```

4. commission_pct 필드의 값이 없는 모든 필드의 데이터(레코드)

```
select * from employees where commission_pct is not null;
```

커서 방식의 페이지네이션

offset 방식은 모든 데이터를 내부적으로 조회해야 하는 단점이 있다.

```
1 • select * from zerocho.employee
2   where team = "개발팀" order by id desc LIMIT 5;
3 ✖ 13, 12, 11, 10, 9, 8, 7 6, 5, 4, 3, 2, 1
```



```
1 • select * from zerocho.employee
2   where team = "개발팀" and id < 9 order by id desc LIMIT 5;
3 ✖ 13, 12, 11, 10, 9, 8, 7 6, 5, 4, 3, 2, 1
```



```
1 • select * from zerocho.employee
2   where team = "개발팀" and id < 4 order by id desc LIMIT 5;
3 ✖ 13, 12, 11, 10, 9, 8, 7 6, 5, 4, 3, 2, 1
```

SELECT

1. 작은거부터 시작해서 순서대로 정렬 (오름차순)

select * from employees where salary >= 3000 and salary <= 4000 order by salary;

2. 큰거부터 시작해서 순서대로 정렬 (내림차순)

select * from employees where salary >= 3000 and salary <= 4000 order by salary desc;

3. 글자의 사전의 순서대로 정렬 (오름차순)

select * from employees where salary >= 3000 and salary <= 4000 order by last_name;

4. 글자의 사전의 반대 순서대로 정렬 (내림차순)

select * from employees where salary >= 3000 and salary <= 4000 order by last_name desc;

1. salary가 같아서 정렬이 안되는 레코드가 있으면 해당 레코드들을 last_name을 기준으로 정렬한다.

select employee_id, first_name, last_name, email
from employees
where salary >= 3000 and salary <= 4000
order by salary, last_name;

2. 마찬가지로 salary가 같아서 정렬이 안되는 레코드는 last_name을 기준으로 정렬한다.

select employee_id, first_name, last_name, email
from employees
where salary >= 3000 and salary <= 4000
order by salary desc, last_name;

※ desc의 반대는 asc

| employee_id | first_name | last_name | email | salary |
|-------------|------------|-----------|----------|---------|
| 192 | Sarah | Bell | SBELL | 4000.00 |
| 193 | Britney | Everett | BEVERETT | 3900.00 |
| 188 | Kelly | Chung | KCHUNG | 3800.00 |
| 189 | Jennifer | Dilly | JDILLY | 3600.00 |
| 137 | Renske | Ladwig | RLADWIG | 3600.00 |
| 141 | Trenna | Rajs | TRAJS | 3500.00 |
| 186 | Julia | Dellinger | JDELLING | 3400.00 |
| 129 | Laura | Bissot | LBISSOT | 3300.00 |
| 133 | Jason | Mallin | JMALLIN | 3300.00 |
| 194 | Samuel | McCain | SMCCAIN | 3200.00 |
| 125 | Julia | Nayer | JNAYER | 3200.00 |
| 138 | Stephen | Stiles | SSTILES | 3200.00 |
| 180 | Winston | Taylor | WTAYLOR | 3200.00 |
| 142 | Curtis | Davies | CDAVIES | 3100.00 |
| 181 | Jean | Fleaur | JFLEAUR | 3100.00 |
| 115 | Alexander | Khoo | AKHOO | 3100.00 |
| 196 | Alana | Walsh | AWALSH | 3100.00 |
| 187 | Anthony | Cabrio | ACABRIO | 3000.00 |
| 197 | Kevin | Feeney | KFEENEY | 3000.00 |

| employee_id | first_name | last_name | email | salary |
|-------------|------------|-----------|----------|---------|
| 187 | Anthony | Cabrio | ACABRIO | 3000.00 |
| 197 | Kevin | Feeney | KFEENEY | 3000.00 |
| 142 | Curtis | Davies | CDAVIES | 3100.00 |
| 181 | Jean | Fleaur | JFLEAUR | 3100.00 |
| 115 | Alexander | Khoo | AKHOO | 3100.00 |
| 196 | Alana | Walsh | AWALSH | 3100.00 |
| 194 | Samuel | McCain | SMCCAIN | 3200.00 |
| 125 | Julia | Nayer | JNAYER | 3200.00 |
| 138 | Stephen | Stiles | SSTILES | 3200.00 |
| 180 | Winston | Taylor | WTAYLOR | 3200.00 |
| 129 | Laura | Bissot | LBISSOT | 3300.00 |
| 133 | Jason | Mallin | JMALLIN | 3300.00 |
| 186 | Julia | Dellinger | JDELLING | 3400.00 |
| 141 | Trenna | Rajs | TRAJS | 3500.00 |
| 189 | Jennifer | Dilly | JDILLY | 3600.00 |
| 137 | Renske | Ladwig | RLADWIG | 3600.00 |
| 188 | Kelly | Chung | KCHUNG | 3800.00 |
| 193 | Britney | Everett | BEVERETT | 3900.00 |
| 192 | Sarah | Bell | SBELL | 4000.00 |

필드 조작하기

1. 25000에서 salary를 뺀 계산 결과가 필드와 레코드 형식으로 나온다. (결과를 보여주는거지 필드로 생기는건 아니다)

```
select first_name, last_name, salary, 25000-salary  
from employees limit 0, 20;
```

2. 계산한 결과의 필드를 salary_gap이라는 이름으로 출력하게 한다.

```
select first_name, last_name, salary, 25000-salary as salary_gap  
from employees limit 0, 20;
```

3. salary/12의 결과를 월급이라는 필드의 이름으로 출력하게 해준다.

```
select first_name, last_name, salary, salary/12 as 월급  
from employees limit 0, 20;
```

4. salary/12의 결과가 1000이상인 레코드만 출력하게 한다.

```
select * from employees where salary/12 >= 1000;
```

```
MariaDB [hr]> select first_name, last_name, salary, 25000-salary  
-> from employees limit 0, 20;
```

| first_name | last_name | salary | 25000-salary |
|-------------|------------|----------|--------------|
| Steven | King | 24000.00 | 1000.00 |
| Neena | Kochhar | 17000.00 | 8000.00 |
| Lex | De Haan | 17000.00 | 8000.00 |
| Alexander | Hunold | 9000.00 | 16000.00 |
| Bruce | Ernst | 6000.00 | 19000.00 |
| David | Austin | 4800.00 | 20200.00 |
| Valli | Pataballa | 4800.00 | 20200.00 |
| Diana | Lorentz | 4200.00 | 20800.00 |
| Nancy | Greenberg | 12000.00 | 13000.00 |
| Daniel | Faviet | 9000.00 | 16000.00 |
| John | Chen | 8200.00 | 16800.00 |
| Ismael | Sciarra | 7700.00 | 17300.00 |
| Jose Manuel | Urman | 7800.00 | 17200.00 |
| Luis | Popp | 6900.00 | 18100.00 |
| Den | Raphaely | 11000.00 | 14000.00 |
| Alexander | Khoo | 3100.00 | 21900.00 |
| Shelli | Baida | 2900.00 | 22100.00 |
| Sigal | Tobias | 2800.00 | 22200.00 |
| Guy | Himuro | 2600.00 | 22400.00 |
| Karen | Colmenares | 2500.00 | 22500.00 |

```
20 rows in set (0.003 sec)
```


날짜(date) 필드 다루기

1. 5월달인 레코드만 출력

```
select * from employees where hire_date like '___-05-__';
```

2. 1995년인 레코드만 출력

```
select * from employees where hire_date like '1995%';
```

3. hire_date를 오름차순으로

```
select * from employees order by hire_date;
```

4. hire_date를 내림차순으로

```
select * from employees order by hire_date desc;
```

날짜 내장함수

1. 현재 날짜와 시간을 출력하는 내장 함수이다

```
select now();
```

2. 현재 년 월 일만 뽑아서 출력

```
select date(now());
```

3. 현재 년도만 출력

```
select year(now());
```

4. hire_date에서 년도만 추출해서 출력

```
select hire_date, year(hire_date) from employees limit 0, 5;
```

5. hire_date에서 월만 추출해서 출력

```
select hire_date, month(hire_date) from employees limit 0, 5;
```

6. hire_date에서 일만 추출해서 출력

```
select hire_date, day(hire_date) from employees limit 0, 5;
```

7. hire_date에서 요일과 월의 이름을 출력

```
select hire_date, dayname(hire_date), monthname(hire_date) from employees limit 0, 5;
```

집계함수

1. 테이블에서 select에 지정된 집계함수의 전체 결과값을 출력한다.

select count(*), max(salary), min(salary), avg(salary), sum(salary) from employees; // count(*) 전체 레코드 개수를 세는 것

```
MariaDB [hr]> select count(*), max(salary), min(salary), avg(salary), sum(salary) from employees;
+-----+-----+-----+-----+-----+
| count(*) | max(salary) | min(salary) | avg(salary) | sum(salary) |
+-----+-----+-----+-----+-----+
|      107 |      24000.00 |       2100.00 | 6461.682243 | 691400.00 |
+-----+-----+-----+-----+-----+
1 row in set (0.000 sec)
```

2. where에 지정된 조건식을 만족하는 레코드의 집계함수 결과값을 출력한다

select count(*), max(salary), min(salary), avg(salary), sum(salary) from employees where year(hire_date) = 1998;

3. last_name필드의 레코드가 몇개가 있는가, commission_pct는 필드의 레코드가 몇개가 있는가

select count(last_name), count(commission_pct) from employees;

※ avg()함수와 sum()함수는 숫자 레코드에만 가능하다.

※ 글자 레코드 중에서 max는 사전순으로 정렬 했을 때 가장 뒤에 정렬된 레코드, min은 가장 앞에 정렬된 레코드

※ 입사일 레코드 중에서 min은 가장 빠른 날짜, max는 가장 늦은 날짜

집계함수 (연습문제)

1. salary가 4000 이하인 직원들은 몇명인가?
2. job_id가 'IT_PROG'인 직원들의 평균 연봉(salary)
3. job_id가 'SH_CLERK'인 직원들의 최저 연봉
4. ommission_pct가 NULL인 직원들의 평균 연봉
5. job_id에 'CLERK'단어가 들어간 직원들의 숫자와 평균 연봉
6. 1998년 1월 1일 이후 입사한 직원들의 평균 연봉
7. 9월에 입사한 직원들의 평균 연봉

집계함수 (정답)

1. salary가 4000 이하인 직원들은 몇명인가?

```
select count(*) from employees where salary <= 4000;
```

2. job_id가 'IT_PROG'인 직원들의 평균 연봉(salary)

```
select avg(salary) from employees where job_id = 'it_prog';
```

3. job_id가 'SH_CLERK'인 직원들의 최저 연봉

```
select min(salary) from employees where job_id = 'sh_clerk';
```

4. ommission_pct가 NULL인 직원들의 평균 연봉

```
select avg(salary) from employees where commmission_pct is null;
```

5. job_id에 'CLERK'단어가 들어간 직원들의 숫자와 평균 연봉

```
select count(*), avg(salary) from employees where job_id like '%clerk%';
```

6. 1998년 1월 1일 이후 입사한 직원들의 평균 연봉

```
select avg(salary) from employees where hire_date > '1998-01-01';
```

7. 9월에 입사한 직원들의 평균 연봉

```
select avg(salary) from employees where month month(hire_date) = 9; // 또는
```

```
select avg(salary) from employees where hire_date like '__-09-__';
```

group by

※ 가장 뒤에 order by 쓰면 오름차순

1. job_id 필드의 레코드가 같은 것끼리 그룹을 만든 다음 각 그룹별로 count, avg 집계함수를 적용한다.

select job_id, count(*), avg(salary) from employees group by job_id;

```
MariaDB [hr]> select job_id, count(*), avg(salary) from employees group by job_id;
```

| job_id | count(*) | avg(salary) |
|------------|----------|--------------|
| AC_ACCOUNT | 1 | 8300.000000 |
| AC_MGR | 1 | 12000.000000 |
| AD_ASST | 1 | 4400.000000 |
| AD PRES | 1 | 24000.000000 |
| AD_VP | 2 | 17000.000000 |
| FI_ACCOUNT | 5 | 7920.000000 |
| FI_MGR | 1 | 12000.000000 |
| HR_REP | 1 | 6500.000000 |
| IT_PROG | 5 | 5760.000000 |
| MK_MAN | 1 | 13000.000000 |
| MK_REP | 1 | 6000.000000 |
| PR_REP | 1 | 10000.000000 |
| PJ_CLERK | 5 | 2780.000000 |
| PJ_MAN | 1 | 11000.000000 |
| SA_MAN | 5 | 12200.000000 |
| SA_REP | 30 | 8350.000000 |
| SH_CLERK | 20 | 3215.000000 |
| ST_CLERK | 20 | 2785.000000 |
| ST_MAN | 5 | 7280.000000 |

2. hire_date의 연도가 같은 것끼리 그룹을 하고 연도가 같은 것들을 출력하고, 각 레코드 개수를 세고, 평균 연봉을 출력

select year(hire_date), count(*), avg(salary) from employees group by year(hire_date);

3. hire_date의 월이 같은 것끼리 그룹을 하고 월이 같은 것들을 출력하고, 각 레코드 개수를 세고, 평균 연봉을 출력

select month(hire_date), count(*), avg(salary) from employees group by month(hire_date);

4. hire_date의 요일이 같은 것끼리 그룹을 하고 요일이 같은 것들을 출력하고, 각 레코드의 개수를 세고, 평균 연봉을 출력

select dayname(hire_date), count(*), avg(salary) from employees group by dayname(hire_date);

group by (연습 문제)

1. salary가 4000 이하인 직원들의 job_id 별 인원수와 평균연봉
2. department_id 별 직원들의 인원수와 평균연봉, 최대연봉, 최저연봉
3. job_id가 'SH_CLERK'인 직원들의 department_id 별 인원수와 평균연봉
4. commission_pct가 NULL인 직원들의 job_id 별 평균 연봉
5. job_id에 'CLERK'단어가 들어간 직원들의 department_id 별 인원수와 평균연봉
6. 1998년 1월 1일 이후 입사한 직원들의 job_id 별 평균연봉
7. 9월에 입사한 직원들의 department_id 별 평균 연봉

group by (정답)

1. salary가 4000 이하인 직원들의 job_id 별 인원수와 평균연봉

```
select job_id, count(*), avg(salary) from employees where salary <= 4000 group by job_id;
```

2. department_id 별 직원들의 인원수와 평균연봉, 최대연봉, 최저연봉

```
select department_id, count(*), avg(salary), max(salary), min(salary) from employees group by department_id;
```

```
select department_id, count(*), avg(salary), max(salary), min(salary) from employees where department_id is not null group by department_id; // (null값 뺀거)
```

3. job_id가 'SH_CLERK'인 직원들의 department_id 별 인원수와 평균연봉

```
select department_id, count(*), avg(salary) from employees where job_id = 'sh_clerk' group by department_id;
```

4. commission_pct가 NULL인 직원들의 job_id 별 평균 연봉

```
select job_id, avg(salary) from employees where commission_pct is null group by job_id;
```

5. job_id에 'CLERK'단어가 들어간 직원들의 department_id 별 인원수와 평균연봉

```
select department_id, count(*), avg(salary) from employees where job_id like '%clerk%' group by department_id
```

6. 1998년 1월 1일 이후 입사한 직원들의 job_id 별 평균연봉

```
select job_id, avg(salary) from employees where hire_date > '1998-01-01' group by job_id;
```

7. 9월에 입사한 직원들의 department_id 별 평균 연봉

```
select department_id, avg(salary) from employees where month(hire_date) = 9 group by department_id;
```


having

1. job_id가 같은 것들 중 count(*)가 1인 그룹만 출력

```
select job_id, count(*), avg(salary) from employees group by job_id having count(*) = 1;
```

2. job_id가 같은 것들 중 count(*)가 1 이상인 그룹만 출력

```
select job_id, count(*), avg(salary) from employees group by job_id having count(*) > 1;
```

3. 평균연봉이 10000 이상인 job_id 별 직원들의 인원수와 평균연봉

```
select job_id, count(*), avg(salary) from employees group by job_id having avg(salary) >= 10000;
```

4. 평균연봉이 10000 이상인 department_id 별 직원들의 인원수와 평균연봉

```
select department_id, count(*), avg(salary) from employees group by department_id having avg(salary) >= 10000;
```

5. 인원수가 5명 이상인 job_id 별 직원들의 인원수와 평균연봉

```
select job_id, count(*), avg(salary) from employees group by job_id having count(*) >= 5;
```

6. 인원수가 30명 이상인 department_id 별 직원들의 인원수와 평균연봉

```
select department_id, count(*), avg(salary) from employees group by department_id having count(*) >= 30;
```

7. 평균연봉이 10000 이상이면서 인원수가 2명 이상인 job_id 별 직원들의 인원수와 평균연봉

```
select job_id, count(*), avg(salary) from employees group by job_id having avg(salary) >= 10000 and count(*) >= 2;
```

정리

1. 집계함수 : count(), avg(), sum(), max(), min()

- avg(), sum() 숫자 필드에만 사용가능
- count(*), max(), min() 모든 필드에 사용가능
 - 글자필드 : max -> 사전순서의 맨 뒤, min -> 사전순서의 맨 앞
 - 날짜필드 : max -> 가장 늦은(최근) 날짜, min -> 가장 이른(옛날) 날짜
- count(필드) : null은 세지 않는다.
- count(*) : 레코드 개수를 센다.
- 집계함수는 select 다음에 필드 대신 쓴다. 다른 필드는 쓸 수 없다.
 - select count(*) from employees; (O)
 - select last_name, count(*) from employees; (X)

2. group by <필드> 또는 <내장함수>

- group by 뒤에 나오는 필드나 내장함수 값이 같은 것들 끼리 그룹을 만들고 그 그룹별로 집계함수를 적용
- group by 뒤에 적은 <필드>, <내장함수>를 select 에 쓸 수 있다.
 - select job_id, count(*) from employees group by job_id; (O)
 - select year(hire_date), count(*) from employees group by year(hire_date); (O)
- group by 다음에 order by 사용 가능
 - => order by 다음에는 select에 나열된 필드나 집계함수를 쓸 수 있다.
 - select job_id, count(*) from employees group by job_id order by job_id; (O)
 - select job_id, count(*) from employees group by job_id order by count(*); (O)
 - select year(hire_date), count(*) from employees group by year(hire_date) order by year(hire_date); (O)
 - select year(hire_date), count(*) from employees group by year(hire_date) order by count(*); (O)

3. having <집계함수의 조건식>

- group by와 함께 사용
- group by로 만들어진 그룹들 중에서 having에 지정된 조건을 만족하는 그룹만 출력
 - select job_id, count(*) from employees group by job_id having count(*) >= 5;

CROSS JOIN

| id | name | email | salary | team | quit_date | created_at | role_id |
|----|------|--------------------|--------|------|-----------|---------------------|---------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 |

| id | name | min_salary |
|----|------|------------|
| 1 | 팀장 | 8000 |
| 2 | 과장 | 6000 |
| 3 | 대리 | 5000 |
| 4 | 신입 | 3000 |
| 5 | 인턴 | 2500 |

CROSS

`SELECT * FROM zerocho.employee JOIN zerocho.role;`

또는

`select * from zerocho.employee, zerocho.role;`

| id | name | email | salary | team | quit_date | created_at | role_id | id | name | min_salary |
|----|------|--------------------|--------|------|-----------|---------------------|---------|----|------|------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 5 | 인턴 | 2500 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 4 | 신입 | 3000 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 대리 | 5000 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 과장 | 6000 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 | 5 | 인턴 | 2500 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 | 4 | 신입 | 3000 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 과장 | 6000 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 5 | 인턴 | 2500 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 4 | 신입 | 3000 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 대리 | 5000 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 과장 | 6000 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 인턴 | 2500 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | 4 | 신입 | 3000 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | 3 | 대리 | 5000 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | 2 | 과장 | 6000 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | 1 | 팀장 | 8000 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 인턴 | 2500 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 4 | 신입 | 3000 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 3 | 대리 | 5000 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 2 | 과장 | 6000 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 1 | 팀장 | 8000 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 5 | 인턴 | 2500 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 4 | 신입 | 3000 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 3 | 대리 | 5000 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 2 | 과장 | 6000 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 1 | 팀장 | 8000 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 5 | 인턴 | 2500 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 4 | 신입 | 3000 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 3 | 대리 | 5000 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 2 | 과장 | 6000 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 1 | 팀장 | 8000 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 5 | 인턴 | 2500 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 3 | 대리 | 5000 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 2 | 과장 | 6000 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 1 | 팀장 | 8000 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 5 | 인턴 | 2500 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 3 | 대리 | 5000 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 2 | 과장 | 6000 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 1 | 팀장 | 8000 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 5 | 인턴 | 2500 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 3 | 대리 | 5000 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 2 | 과장 | 6000 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 1 | 팀장 | 8000 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 5 | 인턴 | 2500 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 4 | 신입 | 3000 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 3 | 대리 | 5000 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 2 | 과장 | 6000 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 1 | 팀장 | 8000 |

CROSS JOIN은 Cartesian Product라고도 불리며, 두 테이블 간의 가능한 모든 조합을 생성해준다.
CROSS JOIN은 테이블간의 조합을 구할 때 말고는 사용할 일이 거의 없다.

employee 테이블의 레코드 개수: 11
role 테이블의 레코드 개수: 5
11x5 해서 총 55개의 레코드를 출력

INNER JOIN

```
SELECT * FROM zerocho.employee;
```

| | id | name | email | salary | team | quit_date | created_at | role_id |
|---|----|------|--------------------|--------|------|-----------|---------------------|---------|
| ▶ | 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 |
| | 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 |
| | 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 |
| | 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 |
| | 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 |
| | 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 |
| | 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 |
| | 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| | 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 4 |
| | 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| | 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 |

1. 사원테이블에서 role_id(외래키)만 보고는 직책명을 알 수가 없다.
직책명을 알려면 role테이블과 연결(조인)을 시켜줘야 한다.

```
SELECT *
FROM zerocho.employee e JOIN zerocho.role r
ON e.role_id = r.id;
```

3. 테이블 이름에 별명을 지어서 더 편하게 쿼리를 쓸 수 있게 해줄 수 있다.
(employee = e, role = r)

다른 방식의 INNER JOIN

```
select * from zerocho.employee e, zerocho.role r
where e.role_id = r.id;
```

| | id | name | email | salary | team | quit_date | created_at | role_id | id | name | min_salary |
|---|----|------|--------------------|--------|------|-----------|---------------------|---------|----|------|------------|
| ▶ | 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| | 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| | 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| | 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | 2 | 과장 | 6000 |
| | 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 2 | 과장 | 6000 |
| | 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 3 | 대리 | 5000 |
| | 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 3 | 대리 | 5000 |
| | 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 |
| | 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 |
| | 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 |
| | 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 5 | 인턴 | 2500 |

```
SELECT *
FROM zerocho.employee JOIN zerocho.role
ON zerocho.employee.role_id = zerocho.role.id
```

| | id | name | email | salary | team | quit_date | created_at | role_id | id | name | min_salary |
|---|----|------|--------------------|--------|------|-----------|---------------------|---------|----|------|------------|
| ▶ | 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| | 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| | 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| | 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | 2 | 과장 | 6000 |
| | 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 2 | 과장 | 6000 |
| | 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 3 | 대리 | 5000 |
| | 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 3 | 대리 | 5000 |
| | 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 |
| | 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 |
| | 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 |
| | 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 5 | 인턴 | 2500 |

2. JOIN을 통해 테이블을 연결할 수 있고 ON으로 조건을 정할 수 있다.
(employee 테이블의 role_id와 role테이블의 id는 서로 같은 목적을 가진 값이므로
각 테이블끼리 조인이 돼서 서로 알맞게 맞춰진다)

| id | name | email | salary | team | quit_date | created_at | role_id | id | name | min_salary |
|----|------|-------------------|--------|------|-----------|---------------------|---------|----|------|------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 |

4. 외관상으로 role_id와 id가 같이 있어서 중복되므로 이럴 때는 아래와 같이 쿼리를 작성할 수 있다.

```
SELECT e.id, e.name as '사원명',
email, team, r.name as '직책명', min_salary
FROM zerocho.employee e JOIN zerocho.role r
ON e.role_id = r.id
```

| id | 사원명 | email | team | 직책명 | min_salary |
|----|-----|-------------------|------|-----|------------|
| 1 | 제로초 | zerocho@gmail.com | 개발팀 | 팀장 | 8000 |
| 2 | 원초 | onecho@gmail.com | 디자인팀 | 팀장 | 8000 |
| 3 | 투초 | twocho@gmail.com | 기획팀 | 팀장 | 8000 |

LEFT, RIGHT JOIN (OUTER)

INNER JOIN은 JOIN을 했을 때 NULL이 안나오지만
OUTER JOIN은 JOIN했을 때 NULL이 나올 수 있다.

```
SELECT * FROM zerocho.employee e LEFT JOIN zerocho.employee_project ep
ON e.id = ep.employee_id;
```

| id | name | email | salary | team | quit_date | created_at | role_id | employee_id | project_id |
|----|------|--------------------|--------|------|-----------|---------------------|---------|-------------|------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 1 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 2 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 1 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 2 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | 4 | 2 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 1 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 2 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 7 | 1 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 8 | 1 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 9 | 2 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 10 | 2 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 11 | 1 |

1. employee 테이블과 employee_project 테이블을 LEFT JOIN 해서
사원이 속한 project_id를 알아볼 수 있다. 하지만 모든 사원이 프로젝트
를 참여하고 있으므로 INNER JOIN과 차이점을 확인 할 수 없다.

```
SELECT * FROM zerocho.employee e JOIN zerocho.employee_project ep
ON e.id = ep.employee_id;
```

| id | name | email | salary | team | quit_date | created_at | role_id | employee_id | project_id |
|----|------|--------------------|--------|------|-----------|---------------------|---------|-------------|------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 1 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 2 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 1 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 2 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 1 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 2 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 7 | 1 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 8 | 1 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 9 | 2 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 10 | 2 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 11 | 1 |

3. INNER JOIN을 한다면 아예 쓰리초의 레코드가 출력되지 않으므로
NULL이 안나온다는 걸 알 수 있다.

| employee_id | project_id |
|-------------|------------|
| 2 | 1 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 11 | 1 |
| 1 | 2 |
| 3 | 2 |

| | |
|----|--|
| 4 | Open Value in Editor |
| 5 | Set Field to NULL |
| 6 | Mark Field Value as a Function/Literal |
| 9 | |
| 10 | Delete Row(s) |

2. 차이점을 확인하기 위해 일단
employee_project 테이블의 employee_id
4번을 지운다. (쓰리초)

```
SELECT * FROM zerocho.employee e LEFT JOIN zerocho.employee_project ep
ON e.id = ep.employee_id
LEFT JOIN zerocho.project p ON ep.project_id = p.id;
```

| id | name | email | salary | team | quit_date | created_at | role_id | employee_id | project_id | id | name | due_date | created_at |
|----|------|--------------------|--------|------|-----------|---------------------|---------|-------------|------------|------|------|------------|---------------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 1 | 1 | 종료일자 | 2023-07-15 | 2023-08-21 19:06:59 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 2 | 2 | AT | 2024-01-31 | 2023-08-21 19:06:59 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 1 | 1 | 종료일자 | 2023-07-15 | 2023-08-21 19:06:59 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 2 | 2 | AT | 2024-01-31 | 2023-08-21 19:06:59 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | NULL | NULL | NULL | 종료일자 | 2023-07-15 | 2023-08-21 19:06:59 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 2 | 2 | AT | 2024-01-31 | 2023-08-21 19:06:59 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 1 | 1 | 종료일자 | 2023-07-15 | 2023-08-21 19:06:59 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 2 | 2 | AT | 2024-01-31 | 2023-08-21 19:06:59 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 7 | 1 | 1 | 종료일자 | 2023-07-15 | 2023-08-21 19:06:59 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 8 | 1 | 1 | 종료일자 | 2023-07-15 | 2023-08-21 19:06:59 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 9 | 2 | 2 | AT | 2024-01-31 | 2023-08-21 19:06:59 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 10 | 2 | 2 | AT | 2024-01-31 | 2023-08-21 19:06:59 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 11 | 1 | 1 | 종료일자 | 2023-07-15 | 2023-08-21 19:06:59 |

4. 프로젝트명까지 알고 싶으면 두번 연달아 JOIN을 하면 볼 수 있다.

```
SELECT * FROM zerocho.employee e LEFT JOIN zerocho.employee_project ep
ON e.id = ep.employee_id;
```

| id | name | email | salary | team | quit_date | created_at | role_id | employee_id | project_id |
|----|------|--------------------|--------|------|-----------|---------------------|---------|-------------|------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 1 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 2 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 1 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 2 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | NULL | NULL |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 1 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 2 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 7 | 1 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 8 | 1 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 9 | 2 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 10 | 2 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 11 | 1 |

3. 다시 JOIN을 해보면 이번엔 NULL이 나와있는 부분을 확인할 수 있다.
(쓰리초는 아무 프로젝트에 속해있지 않기에)

LEFT와 RIGHT 차이점은?

```
SELECT * FROM zerocho.employee e LEFT JOIN zerocho.employee_project ep
ON e.id = ep.employee_id;
```

LEFT JOIN은 왼쪽이 기준이다.

```
SELECT * FROM zerocho.employee e RIGHT JOIN zerocho.employee_project ep
ON e.id = ep.employee_id;
```

RIGHT JOIN은 오른쪽이 기준이다.

| id | name | email | salary | team | quit_date | created_at | role_id | employee_id | project_id |
|----|------|--------------------|--------|------|-----------|---------------------|---------|-------------|------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 1 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 1 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 1 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 7 | 1 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 8 | 1 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 11 | 1 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 2 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 2 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 2 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인 | NULL | 2023-08-21 07:04:50 | 4 | 9 | 2 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 10 | 2 |

RIGHT JOIN 했을 때는 employee_project 테이블이 기준이므로
이전에 employee_id(외래키)의 3번(쓰리초)을 지웠기에
위 사진과 같이 NULL 표시가 없는 걸 알 수 있다.

FULL OUTER JOIN

FULL OUTER JOIN은 LEFT JOIN과 RIGHT JOIN을 합친 것이다.

| id | name | due_date | created_at |
|----|------|------------|---------------------|
| 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 3 | 신규 | 2023-08-15 | NULL |

1. 우선, FULL OUTER JOIN이 어떤 JOIN인지 확인하기 위해 project 테이블에 신규 프로젝트를 추가한다.
(단, 신규 프로젝트에는 아무 것도 연결하면 안된다)

```
SELECT * FROM zerocho.employee e LEFT JOIN zerocho.employee_project ep
ON e.id = ep.employee_id LEFT JOIN zerocho.project p ON ep.project_id = p.id
```

| id | name | email | salary | team | quit_date | created_at | role_id | employee_id | project_id | id | name | due_date | created_at |
|----|------|--------------------|--------|------|-----------|---------------------|---------|-------------|------------|------|------|------------|---------------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | NULL | NULL | NULL | NULL | NULL | NULL |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 7 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 8 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 4 | 9 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 10 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 11 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |

2. LEFT JOIN은 쓰리초가 속해있는 프로젝트가 없기에 NULL 뜨는 부분을 알 수 있다.

```
SELECT * FROM zerocho.employee e LEFT JOIN zerocho.employee_project ep
ON e.id = ep.employee_id LEFT JOIN zerocho.project p ON ep.project_id = p.id
UNION
SELECT * FROM zerocho.employee e LEFT JOIN zerocho.employee_project ep
ON e.id = ep.employee_id RIGHT JOIN zerocho.project p ON ep.project_id = p.id
```

```
SELECT * FROM zerocho.employee e LEFT JOIN zerocho.employee_project ep
ON e.id = ep.employee_id RIGHT JOIN zerocho.project p ON ep.project_id = p.id
```

| id | name | email | salary | team | quit_date | created_at | role_id | employee_id | project_id | id | name | due_date | created_at |
|----|------|--------------------|--------|------|-----------|---------------------|---------|-------------|------------|----|------|------------|---------------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 7 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 8 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 11 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 4 | 9 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 10 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 3 | 신규 | | | | NULL | 2023-08-15 | | | | | | | |

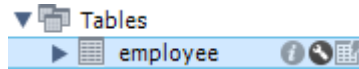
3. RIGHT JOIN은 신규프로젝트에 아무도 속해있지 않기 때문에 NULL 뜨는 부분을 알 수 있다.

| id | name | email | salary | team | quit_date | created_at | role_id | employee_id | project_id | id | name | due_date | created_at |
|----|------|--------------------|--------|------|-----------|---------------------|---------|-------------|------------|------|------|------------|---------------------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 | 2 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 3 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | NULL | NULL | NULL | NULL | NULL | NULL |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 5 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 6 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 7 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 8 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 4 | 9 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 10 | 2 | 2 | AI | 2024-01-31 | 2023-08-21 19:06:59 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 11 | 1 | 1 | 홈페이지 | 2023-07-15 | 2023-08-21 19:06:59 |
| 3 | 신규 | | | | NULL | 2023-08-15 | | | | | | | |

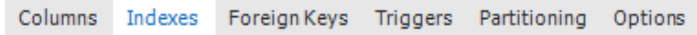
4. LEFT JOIN과 RIGHT JOIN을 UNION으로 합치면 (FULL JOIN) 두가지의 NULL 정보를 함께 볼 수 있다.

INDEX

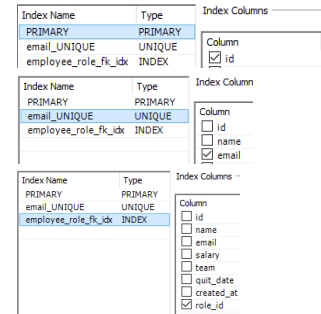
INDEX 확인 방법



1. 테이블의 옵션에 들어간 후



2. Indexs 탭에 들어가면



3. 해당 테이블의 Index들을 확인할 수 있다.
(PK, UQ, FK들에 보통 Index를 많이 걸어놓는다)

INDEX가 필요한 이유

| id | name | email | salary | team | quit_date | created_at | role_id |
|----|------|---------------------|--------|-------|-----------|---------------------|---------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자... | NULL | 2023-08-21 07:04:50 | 1 |
| 3 | 두초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 4 | 쓰리초 | threecho@gmail.... | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 6000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 |
| 7 | 식스초 | sixcho@gmail.com | 6000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 |
| 8 | 세븐초 | sevencho@gmail.... | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 9 | 에잇초 | eightcho@gmail.c... | 4000 | 디자... | NULL | 2023-08-21 07:04:50 | 4 |
| 10 | 나인초 | ninecho@gmail.com | 3000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 |

예를 들어 식스초를 찾는다고 한다면 7번 조회를 해 나가야 한다.
하지만 INDEX로 지정되어 있는 id, email, role_id 중에
특정 데이터를 찾는다고 하면 한번에 찾을 수 있다.

UNIQUE가 INDEX인 이유

새로운 데이터를 넣을 때마다 기존의 데이터랑 겹치지 않도록 전부
조회해야 하므로 INDEX를 걸어놔서 불필요한 조회를 막는다.

INDEX의 단점

위치를 알려주는 INDEX를 어딘가에 따로 저장해야 하기 때문
에 데이터베이스의 용량이 늘어난다.

INDEX의 특징

| Index Name | Type | Index Columns |
|----------------|---------|---------------|
| PRIMARY | PRIMARY | |
| project_fk_idx | INDEX | |

| Column | # | Order |
|---|---|-------|
| <input checked="" type="checkbox"/> employee_id | 1 | ASC |
| <input checked="" type="checkbox"/> project_id | 2 | ASC |

| Index Name | Type | Index Columns |
|----------------|---------|---------------|
| PRIMARY | PRIMARY | |
| project_fk_idx | INDEX | |

| Column | # | Order |
|--|---|-------|
| <input type="checkbox"/> employee_id | | ASC |
| <input checked="" type="checkbox"/> project_id | 1 | ASC |

PRIMARY는 복합키(PK)이므로 INDEX,
project_fk_idx는 외래키이므로 INDEX이다.
그럼 employee_id는 왜 INDEX가 아닐까?

| Index Name | Type | Index Columns |
|----------------|---------|---------------|
| PRIMARY | PRIMARY | |
| project_fk_idx | INDEX | |

| Column | # | Order |
|---|---|-------|
| <input checked="" type="checkbox"/> employee_id | 1 | ASC |
| <input checked="" type="checkbox"/> project_id | 2 | ASC |

→ PRIMARY 안에서 처음에 employee_id로 정렬을 해놓고 employee_id
중에 중복된게 있다면 2순위로 project_id 기준으로 정렬된다.
그래서 PRIMARY KEY가 사실상 employee_id INDEX인거나 마찬가지다.
(참고로 검색 조건(where)에 2순위인 project_id만 사용한다면 해당
INDEX는 작동하지 않는다. 항상 1순위가 포함되어야 한다)
(참고로 여기서 정렬은 select 했을 때의 정렬을 말한다)

EXPLAIN

EXPLAIN은 쿼리 성능 체크할 때 쓰인다.

```
EXPLAIN SELECT * FROM zerocho.employee;
```

```
EXPLAIN SELECT * FROM zerocho.employee where name = '제르초';
```

EXPLAIN 결과는 마찬가지로

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|----|-------------|----------|------------|------|---------------|------|---------|------|------|----------|-------|
| 1 | SIMPLE | employee | NULL | ALL | NULL | NULL | NULL | NULL | 11 | 100.00 | NULL |

ALL은 전부 조회했다는 의미

11개를 조회 했다는 의미

만약 조건(where)에 INDEX가 지정되어있는 조건을 쓴다면?

```
EXPLAIN SELECT * FROM zerocho.employee where id = 7;
```

| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|----|-------------|----------|------------|-------|---------------|---------|---------|-------|------|----------|-------|
| 1 | SIMPLE | employee | NULL | const | PRIMARY | PRIMARY | 4 | const | 1 | 100.00 | NULL |

ALL이 아니라면 INDEX를 활용해서 찾았다는 의미

1개를 조회 했다는 의미

TRANSACTION

InnoDB만 TRANSACTION을 지원

| id | name | min_salary |
|----|------|------------|
| 1 | 팀장 | 8000 |
| 2 | 과장 | 6000 |
| 3 | 대리 | 5000 |
| 4 | 신입 | 3000 |
| 5 | 인턴 | 2500 |

```
UPDATE zerocho.role SET min_salary = 4000 where id = 4;
```

```
UPDATE zerocho.employee SET salary = 4000 where salary < 4000 AND role_id = 4;
```

1. 예를 들어 직책테이블의 신입 최저연봉을 4000으로 올리는 쿼리문을 작성을 했다고 가정을 해보자. 그런데 두번 째 쿼리문에 오타를 내어 사원 테이블에 연봉 4000 미만 신입들의 연봉이 업데이트가 안됐다. 이럴 때는 쿼리를 적용하기 이전으로 롤백 시켜줘야하므로 TRANSACTION이 필요하다.

ROLLBACK;

3. 하지만 쿼리문에서 오타를 내었기에 ERROR가 발생하므로 ROLLBACK;을 입력하고 ROLLBACK; 부분을 드래그 해서 실행 시키면 롤백 시킬 수 있다.

COMMIT;

4. 성공을 확인했으면 COMMIT;을 입력해서 실제 반영이 되도록 적용시킬 수 있다.

```
START TRANSACTION;
```

```
UPDATE zerocho.role SET min_salary = 4000 where id = 4;
```

```
UPDATE zerocho.employee SET salary = 4000 where salary < 4000 AND role_id = 4;
```

2. TRANSACTION을 사용할 때는 앞에 START TRANSACTION을 입력한 후 적용할 쿼리문을 드래그해서 번개 버튼을 눌러 실행. (TRANSACTION 모드이므로 실제 적용은 안됨)
참고: START TRANSACTION 대신 BEGIN도 가능

```
START TRANSACTION;
```

```
UPDATE zerocho.role SET min_salary = 4000 where id = 4;
```

```
UPDATE zerocho.employee SET salary = 4000 where salary < 4000 AND role_id = 4;
```

4. 이번엔 오타를 수정해보고 실행해서 정상 작동하는지 확인한다.

ACID

Atomicity: TRANSACTION은 하나의 단위이므로 전부 성공하든지 전부 실패하든지 해야한다.

Consistency: TRANSACTION이 수행되고 난 뒤에도 정상적인 데이터를 갖고 있어야한다.

Isolation: TRANSACTION간에는 서로 영향을 끼치지 않고 고립되어있는, 별개이어야 한다. (즉, 순서가 존재)

Durability: TRANSACTION이 한번 성공했으면 DB에 완전히 저장되고 실패했으면 완전히 원상태로 복구되어야 한다.

TRANSACTION은 ACID를 만족시킨다.

대부분의 경우에는 서버프레임워크가 알아서 판단해서 COMMIT과 ROLLBACK을 시켜준다.

TRANSACTION 격리 수준

Isolation -> LOCK, Stored Procedure, Trigger, View

LOCK에는 Shared, Exclusive가 있지만 DB를 전문적으로 할 사람만

A: START TRANSACTION;

A: SELECT balance FROM wallet WHERE id = 1 // 100000

A: UPDATE wallet WHERE id = 1 SET balance = balance - 70000 // 30000

B: START TRANSACTION;

B: SELECT balance FROM wallet WHERE id = 1 // 30000

A: COMMIT;

B: UPDATE wallet WHERE id = 1 SET balance = balance - 70000

B: COMMIT;

1. A한테 7만원 빼고 B한테 또 7만원을 빼려 하지만
잔액이 3만원이므로 7만원을 뺄 수 없는 상황

READ Committed

A: START TRANSACTION;

A: SELECT balance FROM wallet WHERE id = 1 // 100000

A: UPDATE wallet WHERE id = 1 SET balance = balance - 70000 // 30000

B: START TRANSACTION;

B: SELECT balance FROM wallet WHERE id = 1 // 100000

A: COMMIT; // balance 30000

B: SELECT balance FROM wallet WHERE id = 1 // 30000

B: UPDATE wallet WHERE id = 1 SET balance = balance - 70000

B: COMMIT;

3. READ Committed는 그냥 10만원으로 받아들여서 dirty read를 해결.
그런데 SELECT를 커밋 전후는 값이 달라지며 non-repeatable read 문제 발생

Serializable

Repeatable READ의 문제점은 phantom read라는 문제가 있었고,
그것을 극복한게 Serializable이다.

하지만 InnoDB에서는 phantom read가 발생하지 않기 때문에
Serializable랑 Repeatable READ가 차이가 없다.

READ Uncommitted

A: START TRANSACTION;

A: SELECT balance FROM wallet WHERE id = 1 // 100000

A: UPDATE wallet WHERE id = 1 SET balance = balance - 70000 // 30000

B: START TRANSACTION;

B: SELECT balance FROM wallet WHERE id = 1 // 30000 dirty read (커밋되지 않은 데이터를 읽어버림)

A: ROLLBACK; // balance: 100000

B: UPDATE wallet WHERE id = 1 SET balance = balance - 70000

B: COMMIT;

2. 그런데 A의 COMMIT이 ROLLBACK이면 10만원으로 돌아옴.
근데 B는 이미 잔액이 30000으로 출력되버림.
그래서 B의 마지막 UPDATE는 실행 x

READ Committed

A: START TRANSACTION;

A: SELECT balance FROM wallet WHERE id = 1 // 100000

A: UPDATE wallet WHERE id = 1 SET balance = balance - 70000 // 30000

B: START TRANSACTION;

B: SELECT balance FROM wallet WHERE id = 1 // 100000

A: COMMIT; // balance 30000

B: SELECT balance FROM wallet WHERE id = 1 // 30000

B: SELECT balance FROM wallet WHERE id = 1 // 30000

B: UPDATE wallet WHERE id = 1 SET balance = balance - 70000

B: COMMIT;

4. COMMIT 전에 있어야할 SELECT를 못하도록 막고 COMMIT 밑으로 이동시킴.
이렇게 되면 Repeatable READ 상태 (InnoDB가 사용)

LOCK

트랜잭션에 어떤 트랜잭션이 접근하지 못하도록 차단했다는 의미에서 LOCK이다. (START TRANSACTION)
(dirty read, non-repeatable read 문제들을 해결)

LOCK을 해제하려면 COMMIT이나 ROLLBACK

Isolation(격리) 수준의 유형

READ Uncommitted <- dirty read 문제 발생 (격리수준 가장 낮음)

READ Committed <- non-repeatable read 문제 발생

Repeatable READ <- InnoDB가 사용

Serializable (격리수준 가장 높음)

Stored Procedure

```
START transaction;
UPDATE role SET min_salary = 4000 WHERE id = 4;
UPDATE employee SET salary = 4000 WHERE role_id = 4 AND salary < 4000;
COMMIT;
```

1. 쿼리문에서 salary나 id 값을 수정하려 한다면 3번 바뀌어야하므로 귀찮을 수 있다.
이럴 때 함수 느낌의 Stored Procedure를 사용하며, 재사용도 가능하다.
(min_salary와 salary는 세트, id와 role_id도 세트)

```
DELIMITER //
CREATE PROCEDURE 프로시저 이름(매개변수)
BEGIN
    프로시저가 실행될 때 수행될 SQL 문
END //
DELIMITER ;
```

2. 프로시저 생성 구조

```
use zerocho;
DELIMITER //
CREATE PROCEDURE increaseMinSalary(money INT, rid INT)
BEGIN
    START TRANSACTION;
    UPDATE zerocho.role SET min_salary = money WHERE id = rid;
    UPDATE zerocho.employee SET salary = money WHERE role_id = rid AND salary < money;
    COMMIT;
END //
DELIMITER ;
```

3. 프로시저 생성은 이렇게 생성할 수 있다.

```
CALL increaseMinSalary(3000, 4);
```

3. 프로시저 호출

```
DROP PROCEDURE zerocho.increaseMinSalary;
```

4. 프로시저 삭제

```
SHOW PROCEDURE STATUS;
```

4. 프로시저 목록 출력

Trigger

```
1 UPDATE zerocho.role SET min_salary = money WHERE id = rid;
2 UPDATE zerocho.employee SET salary = money WHERE role_id = rid AND salary < money;
```

1. 1번 쿼리처럼 업데이트를 하면 자동으로 2번째 쿼리를 실행되게 하고 싶을 때 Trigger를 사용한다.

```
DELIMITER //
CREATE TRIGGER 트리거 이름
트리거_실행시간 트리거이벤트 ON 연결할 테이블
FOR EACH ROW
BEGIN
    트리거가 실행될 때 수행될 SQL 문
END //
DELIMITER ;
```

2. 트리거 생성 구조

```
DELIMITER //
CREATE TRIGGER updateSalaryWhenMinSalaryChange AFTER UPDATE ON zerocho.role
FOR EACH ROW
BEGIN
    UPDATE zerocho.employee SET salary = NEW.min_salary WHERE role_id = NEW.id AND salary < NEW.min_salary;
END //
DELIMITER ;
```

BEFORE: 연결된 이벤트가 실행되기 전에 트리거를 실행
AFTER: 연결된 이벤트가 실행된 후에 트리거를 실행

OLD: 업데이트가 발생하기 전의 데이터
NEW: 업데이트가 발생한 후의 데이터

3. 트리거는 이렇게 생성할 수 있다.

| id | name | email | salary | team | quit_date | created_at | role_id |
|----|------|--------------------|--------|------|-----------|---------------------|---------|
| 1 | 제로초 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 2 | 원초 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 3 | 투초 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 |
| 4 | 쓰리초 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 |
| 5 | 포초 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 |
| 6 | 파이브초 | fivecho@gmail.com | 10000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 |
| 7 | 식스초 | sixcho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 8 | 세븐초 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 9 | 에잇초 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 10 | 나인초 | ninecho@gmail.com | 4000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 |
| 11 | 텐초 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 |

| id | name | min_salary |
|----|------|------------|
| 1 | 팀장 | 8000 |
| 2 | 과장 | 6000 |
| 3 | 대리 | 10000 |
| 4 | 신입 | 3000 |
| 5 | 인턴 | 2500 |

4. 대리 최저연봉을 10000으로 바꾸면 직책이 대리인 사원의 연봉이 자동으로 10000으로 바뀌어져 있는 걸 알 수 있다.

```
drop TRIGGER updateSalaryWhenMinSalaryChange;
```

5. 트리거 삭제

```
show TRIGGERS;
```

6. 트리거 목록 출력

View

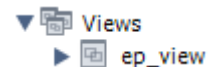
```
SELECT employee_id, project_id, role_id, e.name as employee_name, salary, team
FROM zerocho.employee e LEFT JOIN zerocho.role r ON e.role_id = r.id
LEFT JOIN zerocho.employee_project ep ON e.id = ep.employee_id;
```

| id | name | email | salary | team | quit_date | created_at | role_id | id | name | min_salary | employee_id | project_id |
|----|------|--------------------|--------|------|-----------|---------------------|---------|----|------|------------|-------------|------------|
| 1 | 제로조 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 | 1 | 1 |
| 1 | 제로조 | zerocho@gmail.com | 10000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 | 1 | 2 |
| 2 | 원조 | onecho@gmail.com | 6000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 | 2 | 1 |
| 3 | 투조 | twocho@gmail.com | 8000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 1 | 1 | 팀장 | 8000 | 3 | 2 |
| 4 | 쓰리조 | threecho@gmail.com | 7000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 2 | 2 | 과장 | 6000 | NULL | NULL |
| 5 | 포조 | fourcho@gmail.com | 9000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 2 | 2 | 과장 | 6000 | 5 | 2 |
| 6 | 파이브조 | fivecho@gmail.com | 5000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 3 | 대리 | 5000 | 6 | 1 |
| 6 | 파이브조 | fivecho@gmail.com | 5000 | 기획팀 | NULL | 2023-08-21 07:04:50 | 3 | 3 | 대리 | 5000 | 6 | 2 |
| 7 | 식스조 | sixcho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 3 | 3 | 대리 | 5000 | 7 | 1 |
| 8 | 세븐조 | sevencho@gmail.com | 5000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 | 8 | 1 |
| 9 | 에잇조 | eightcho@gmail.com | 4000 | 디자인팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 | 9 | 2 |
| 10 | 나인조 | ninecho@gmail.com | 4000 | 개발팀 | NULL | 2023-08-21 07:04:50 | 4 | 4 | 신입 | 3000 | 10 | 2 |
| 11 | 텐조 | tencho@gmail.com | 2500 | 기획팀 | NULL | 2023-08-21 07:04:50 | 5 | 5 | 인턴 | 2500 | 11 | 1 |

1. 위 사진과 같이 출력 결과를 볼 때 매번 저렇게 긴 쿼리를 치기 보단 가상 테이블로 만들어 둘 수 있는데 그걸 View라고 한다.

```
CREATE VIEW zerocho.ep_view
AS SELECT employee_id, project_id, role_id, e.name as employee_name, salary, team
FROM zerocho.employee e LEFT JOIN zerocho.role r ON e.role_id = r.id
LEFT JOIN zerocho.employee_project ep ON e.id = ep.employee_id;
```

3. 뷰 생성은 이렇게 할 수 있다.



4. 스키마를 새로고침하면 View가 생성되어있는 것을 볼 수 있다.

CREATE VIEW 뷰 이름
AS 쿼리문

CREATE OR REPLACE VIEW 뷰 이름
AS 쿼리문

2. 뷰 생성 구조

CREATR OR REPLACE VIEW는 뷰를 업데이트 하는 것이다.

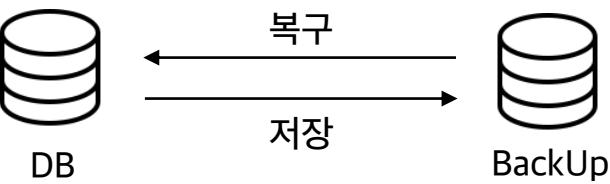
하지만 뷰 이름이 없다면 create 하므로 처음부터 그냥 CREATE OR REPLACE VIEW로 해도 된다.

```
SELECT * FROM zerocho.ep_view WHERE salary >= 6000;
```

5. 뷰의 장점은 뷰 안에서 쿼리를 작성할 수 있다.

Replication & Sharding

백업이란?



백업의 문제점

일요일에 백업을 했다고 가정을 하고, 3일 후 수요일에 DB에 문제가 생겨서 백업하려고 하면 월요일~수요일까지의 시차는 복구할 수가 없다.

Sharding

| Id | Title | Owner | Content | Created at |
|-------|-------|-------|---------|------------|
| 1 | | | | |
| 2 | | | | |
| ⋮ | | | | |
| 50000 | | | | |

post 테이블

커뮤니티 게시판의 경우 테이블의 데이터 양이 많아져 테이블을 쪼개는 방법을 쓸 수 있는데 그걸 Sharding라고 부른다.

Range 방식: 데이터를 특정 범위에 따라 여러 파티션으로 분할

- post1: 1~10000
- post2: 10001~20000
- post3: 20001~30000
- post4: 30001~40000
- post5: 40001~50000
- ⋮

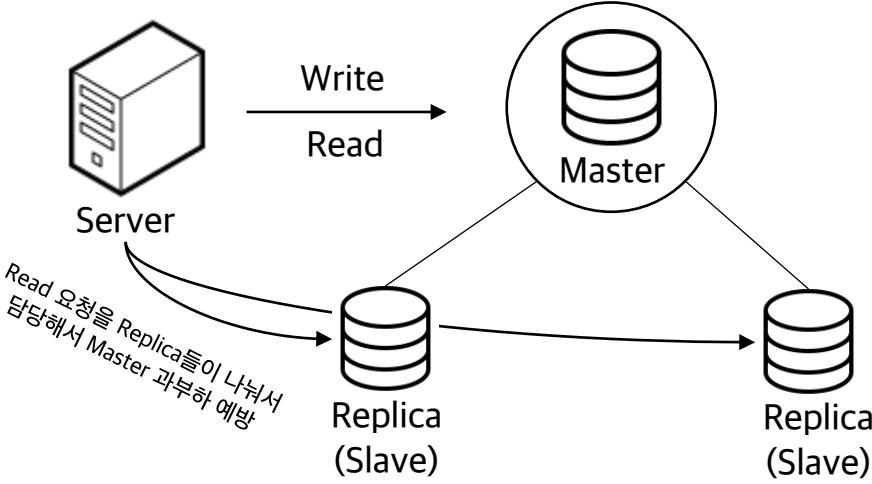
Modulo 방식: 데이터를 파티션 수로 나눈 나머지를 사용하여 데이터를 적절한 파티션에 할당

- (나머지가 1인 곳만)
- (나머지가 2인 곳만)
- (나머지가 3인 곳만)
- (나머지가 4인 곳만)
- (나머지가 5인 곳만)

Hash 방식: 해시 함수를 사용하여 데이터를 여러 파티션으로 분산

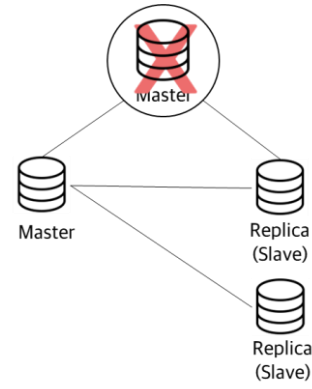
Replication

백업의 긴 시차 문제를 극복



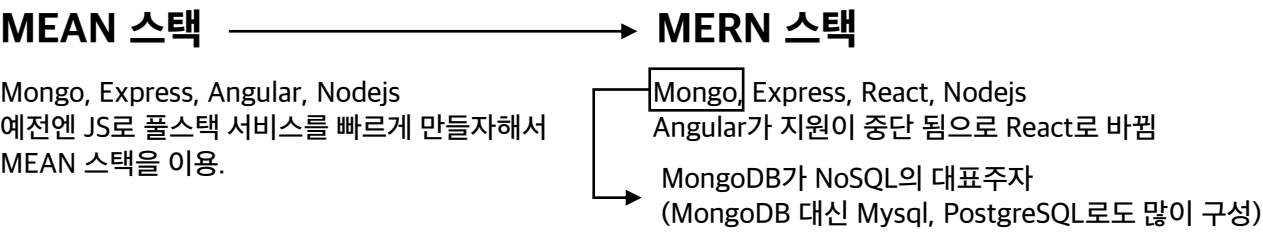
Master에서 Write가 발생하면 Replica로 내려보냄 (Sync)
Replica는 Read만 담당 (Replica는 여러 개 가질 수 있다)

Master가 고장나면?



Replica 하나를 Master로 승격시키고 새로운 Replica를 띄운다.

NoSQL(MongoDB, Redis)



NoSQL 특징

쿼리는 있지만 관계형DB가 아니다.
즉, 정규화 1:1 1:N M:N이 없다.

| Id | Title | content | Owner |
|----|-------|---------|--|
| 1 | ~~~~~ | ~~~~~ | { id : 1 name : onecho email : ~~~ } |
| 2 | ~~~~~ | ~~~~~ | { id : 2 name : twocho email : ~~~ } |

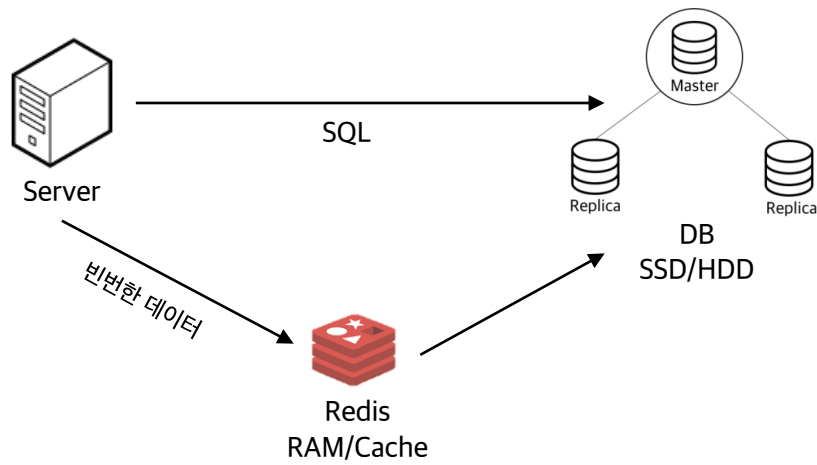
NoSQL은 외래키로 조인을 안하고 Owner에 객체를 통째로 집어넣는다.
단점: 공간낭비
장점: 어느 레코드부터 갑자기 컬럼을 추가했다가 없었다가 자유롭게 가능 → 빅데이터에 유리

Redis

빠른 오픈 소스 인 메모리 키 값 데이터 구조 스토어

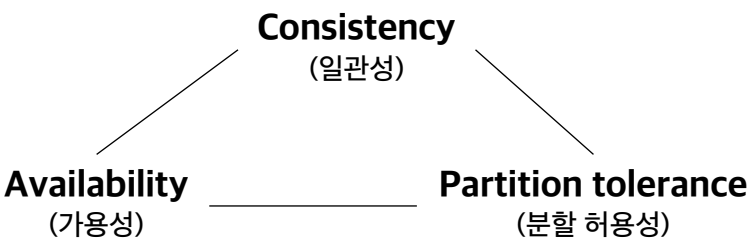
Redis { Memory DB
자료구조 DB

DB를 메모리(램)에다가 저장하므로 접근이 매우 빠르다. (단, 용량이 작음)
자료구조를 DB에 저장할 수 있다.
일반적으로 빈번하게 접근하는 데이터는 Redis를 이용하고, Redis에 없는 데이터만 DB에서 가져오는 방식을 쓴다.
Redis에 있는 데이터와 DB에 있는 데이터가 불일치 할 수 있으므로 서버 단에서 일치할 수 있게 만들어야 한다.
데이터 불일치로 인한 시차를 예방하기 위해 Redis에는 데이터의 유효기간도 넣어놓으며 유효기간이 지나면 DB에서 가져온다.

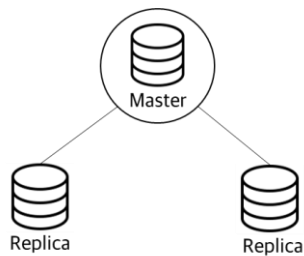


CAP, BASE, ACID

CAP 이론 (분산 시스템) CAP Theorem



CAP 이론은 Consistency, Availability, Partition tolerance의 앞글자를 딴 것이며, 분산 시스템(Replication) 관계에서 발생하는 문제를 다룬 것이다.



- Consistency: DB들은 셋다 똑같은 데이터를 갖고 있어야 한다.
- Availability: 모든 DB들은 항상 사용이 가능해야 한다.
- Partition tolerance: DB들간의 연결고리 하나가 끊겼다 하더라도 전체적으로는 정상 작동해야 한다.

CAP는 동시에 만족시킬 수 없다.

Master의 수정된 사항을 Replica들한테 전달하되는 동안 시차가 발생하게 되는데, 시차 동안에는 Replica는 수정되기 이전 데이터를 갖고 있으므로 Replica에 접근하면 안된다. 게다가 시차로 인해 Replica가 많아질 수록 Consistency를 지키려면 시간이 필요하므로 지키기가 어려워진다. 그러므로 Consistency를 지키려면 Availability를 포기해야 하고, Availability를 지키려면 Consistency를 포기해야 하는 문제가 발생하므로 CP, AP 둘다 문제가 있다. 결론적으로 CAP이론에서는 CP와 AP만 존재하고 CA와 CAP는 불가능하다. (P가 안되는 상황은 애초에 분산 시스템이 아니다)

↓
PACELC Theorem 탄생

BASE 이론/모델 (NoSQL에 주로 적용)

- Basically Available: 완벽하게 Consistency하지 않아도 기본적으로는 항상 Availability 하다.
- Soft State: Consistency가 조금 깨질 수 있지만 개발자들이 극복을 하자.
- Eventually consistent: 최종적으로는 시간이 지나면 데이터들을 Consistency(일관성)있게 맞춘다.

BASE 모델을 따르는 곳은 가용성이 굉장히 높다.

ACID 이론/모델 (BASE와 반대)

TRANSACTION을 통해서 Consistency를 최대한 맞추려고 하는 것이 ACID 모델이다. 단, LOCK과 ISOLATION 때문에 가용성이 떨어진다.

ACID 모델을 따르는 곳은 일관성이 굉장히 높다.