3. 데이터베이스 시작

#0.강의/2.데이터베이스로드맵/1.입문

데이터베이스 시작하기1

데이터베이스를 본격적으로 사용하기 전에 아직 MySQL이 설치되어 있지 않다면 다음을 참고해서 본인의 환경에 맞는 MySQL을 설치하고, MySQL 워크벤치도 실행하자.

1.강의 소개와 수업 자료

- 윈도우 - MySQL 설치 안내
- macOS - MySQL 설치 안내

이제 직접 해보자!

지금까지 우리는 왜 데이터베이스가 필요한지, 그리고 관계형 데이터베이스의 핵심 개념인 테이블, 기본 키, 외래 키에 대해 학습했다. 이론을 아는 것도 중요하지만, 진정한 이해는 우리 손으로 직접 무언가를 만들어 볼 때 찾아온다. "백문이 불여일타"다!

이제부터 우리는 데이터베이스와 직접 '대화'를 시작할 것이다. 이 대화를 위한 언어가 바로 **SQL(Structured Query Language)**이다. 처음에는 낯설고 어렵게 느껴질 수 있지만, 걱정할 필요 없다. 우리가 일상에서 사용하는 언어처럼 SQL도 명확한 목적과 문법이 있다. 그리고 무엇보다 쉽다! 이번 장에서는 우리 쇼핑몰의 데이터를 관리하기 위해 꼭 필요한 최소한의 SQL 명령어들을 맛보는 시간을 가질 것이다.

데이터의 집합, 데이터베이스 생성과 선택

가장 먼저 할 일은 데이터를 담을 거대한 저장 공간, 즉 '데이터베이스'를 만드는 것이다. 이것은 우리 쇼핑몰의 모든 데이터를 보관할 전용 창고를 짓는 것과 같다.

`CREATE DATABASE` : 데이터베이스 생성하기

`CREATE DATABASE` 는 말 그대로 데이터베이스라는 창고를 만드는 명령어다. 우리 쇼핑몰의 이름을 '마이샵(my_shop)'이라고 가정하고, 이 이름으로 데이터베이스를 생성해 보자.

```
CREATE DATABASE my_shop;
```

- 데이터베이스는 세미콜론(`;`) 단위로 하나의 SQL로 인식한다. 따라서 실행할 SQL의 마지막에 `;` 를 넣어주면 된다.
- 명령을 실행하면, MySQL 서버 안에 `my_shop` 이라는 이름의 독립된 데이터 공간이 만들어진다. 이제 이 공간 안에 고객, 상품, 주문 테이블들을 만들어 나갈 것이다.

MySQL 워크벤치 쿼리 실행 방법

MySQL 워크벤치에서 메뉴바 ➔ Query에 보면 다양한 쿼리 관련 기능이 제공된다.

- Execute (All or Selection): 영역을 지정하지 않으면 화면에 있는 모든 SQL을 다 실행한다. 영역을 지정하면 지정한 영역의 SQL만 실행한다.
- Execute Current Statement: 현재 마우스 커서가 있는 SQL만 실행한다.

쿼리를 실행하면 하단의 **Action Output**에 결과가 출력된다. 초록색은 성공, 빨간색은 실패다. 실패한 경우 메시지를 확인하면 어떤 이유로 실패했는지 알 수 있다.

> **대소문자 구분**
> `CREATE DATABASE` 와 같이 SQL이 제공하는 기본 키워드들은 대소문자를 구분하지 않는다.
> `my_shop` 같이 우리가 직접 만든 이름은 데이터베이스 종류나 환경 설정에 따라 대소문자를 구분하기도 한다.
> 따라서 다음과 같이 사용해도 된다. 더 자세한 내용은 뒤에서 설명하겠다.
> `create database my_shop;`

> **실무 팁**
> 데이터베이스 이름은 보통 프로젝트 이름이나 서비스의 특징을 나타내는 것으로 정한다. 소문자와 언더스코어(`_`)를 조합하여 만드는 것이 일반적인 관례다. (예: `my_shop` , `shopping_mall_db`)

`USE` : 작업할 데이터베이스 선택하기

서버에는 `my_shop` 데이터베이스 외에도 다른 여러 데이터베이스가 존재할 수 있다. 따라서 우리가 앞으로 수행할 모

든 작업(테이블 생성, 데이터 추가 등)이 정확히 어느 데이터베이스를 대상으로 하는지 알려줘야 한다.

`USE` 명령어는 비유하자면 여러 개의 엑셀 파일 중 앞으로 작업할 파일 하나를 더블 클릭해서 여는 행위와 같다.

```
USE my_shop;
```

이 명령을 실행하면, 지금부터 우리가 입력하는 모든 SQL 명령어는 `my_shop` 데이터베이스 안에서 실행된다. 데이터베이스를 생성한 후에는 항상 `USE` 명령어로 작업 공간을 선택하는 것을 잊지 말아야 한다.

> **MySQL 워크벤치를 다시 시작하면 반드시 `USE my_shop` 명령어를 다시 실행해서 데이터베이스를 선택해야 한다. 그렇지 않으면 데이터베이스가 선택되지 않았다는 다음과 같은 오류가 발생한다.**
>
> ```
> Error Code: 1046. No database selected Select the default DB to be
> used by double-clicking its name in the SCHEMAS list in the sidebar.
> ```

데이터를 담을 표, 테이블 생성

데이터베이스라는 창고를 지었으니, 이제 그 안에 물건을 정리할 선반, 즉 **테이블**을 만들 차례다.

`CREATE TABLE`: 테이블 설계하고 만들기

`CREATE TABLE`은 테이블의 구조를 정의하고 생성하는 명령어다. 어떤 열(Column)들로 구성될지, 각 열에는 어떤 종류의 데이터가 들어갈지를 명시해야 한다.

우선 다음 코드를 실행해보자.

```
CREATE TABLE sample (
    product_id INT PRIMARY KEY,
    name VARCHAR(100),
    price INT,
    stock_quantity INT,
    release_date DATE
);
```

이렇게 하면 `my_shop` 데이터베이스 안에 상품 정보를 저장할 `sample` 테이블이 만들어진다. 이렇게 테이블을 만드는 과정을 '테이블을 설계(또는 정의)한다'고 말한다.

위 SQL 코드를 해석해 보면 다음과 같다.
"`sample` 라는 이름의 테이블을 생성한다. 이 테이블은 `product_id`(정수), `name`(최대 100자 문자열), `price`(정수), `stock_quantity`(정수), `release_date`(날짜) 라는 5개의 열로 구성된다."

데이터베이스 테이블에는 반드시 기본 키(PRIMARY KEY)가 있어야 한다.
기본 키란, 테이블에 있는 모든 행(Row)들 중에서 특정 행 하나를 유일하게 식별할 수 있는 열(Column) 또는 열들의 조합을 말한다.
여기서 기본 키(`PRIMARY KEY`)는 `product_id` 컬럼을 지정했다. 따라서 `product_id`에는 항상 다른 값이 입력되어야 한다. 우리는 `product_id`를 통해서 이곳에 입력되는 값을 명확히 구분할 수 있다.

데이터 타입 간단 소개
각 열에 저장될 데이터의 종류를 알려주는 '데이터 타입(Data Type)' 몇 가지만 간단히 알아보자.

- `INT`: 정수(Integer)를 의미한다. `1`, `100`, `-50`과 같은 소수점이 없는 숫자를 저장할 때 사용한다. 상품의 가격, 재고 수량, 고객 ID 등에 적합하다.
- `VARCHAR(n)`: 문자열(Variable Character)을 의미한다. `n`은 저장할 수 있는 최대 글자 수를 나타낸다. 예를 들어 `VARCHAR(100)`은 최대 100글자까지의 텍스트를 저장할 수 있다는 뜻이다. 고객 이름, 상품명처럼 길이가 다양한 글자를 저장할 때 사용한다.
- `DATE`: 날짜(YYYY-MM-DD 형식)를 저장할 때 사용한다. 주문일, 가입일, 상품 출시일 등을 기록하기에 적합하다.

이 외에도 수많은 데이터 타입이 있지만, 지금은 이 세 가지만 알아도 충분하다. 데이터 타입에 대한 자세한 내용은 뒤에서 설명한다.

구조 확인과 삭제

테이블을 만들었으니, 의도한 대로 잘 만들어졌는지 확인해보고, 혹시 잘못 만들었다면 삭제하는 방법도 알아보자.

`DESCRIBE`: 테이블 구조 확인하기
`DESCRIBE` 명령어는 테이블이 어떤 열들로, 어떤 데이터 타입으로 구성되어 있는지 그 구조를 보여준다. `DESCRIBE`는 `DESC`로 줄여서 사용할 수 있다.

```
DESC sample; -- 또는 DESCRIBE sample;
```

> **주석**
> SQL 앞에 `--`와 공백을 넣으면 주석이 된다. 주석은 실행되지 않는다. 주석은 컴퓨터가 아닌 사람이 이해할 용도의 메모다.

이 명령을 실행하면 방금 우리가 만든 `sample` 테이블의 설계도(열 이름, 데이터 타입 등)를 표 형태로 깔끔하게 보여준다.

[실행 결과]

Field	Type	Null	Key	Default	Extra
product_id	int	NO	PRI	NULL	
name	varchar(100)	YES		NULL	
price	int	YES		NULL	
stock_quantity	int	YES		NULL	
release_date	date	YES		NULL	

`SHOW DATABASES`, `SHOW TABLES`: 목록 보기

현재 DBMS 서버에 어떤 데이터베이스들이 있는지, 그리고 현재 선택된 데이터베이스(`my_shop`) 안에 어떤 테이블들이 있는지 목록을 보고 싶을 때 사용한다.

현재 서버에 있는 모든 데이터베이스 목록을 보여준다. my_shop도 보일 것이다.

```
SHOW DATABASES;
```

[실행 결과]

Database
information_schema

my_shop
mysql
performance_schema
sys
...

- 실행 결과는 환경에 따라 조금씩 다를 수 있다. `my_shop`은 꼭 있어야 한다.
- 나머지는 DBMS 운영을 위해 내부에서 사용하는 기본 데이터베이스들이다.

이번에는 `my_shop` 데이터베이스에 있는 테이블 목록을 확인해보자.

```
SHOW TABLES;
```

- 반드시 `USE my_shop`을 먼저 실행한 후에 `SHOW TABLES` SQL을 실행해야 한다.
- `my_shop` 안에 있는 테이블 목록을 보여준다. 앞서 우리가 만든 `sample` 테이블이 보인다.

[실행 결과]

Tables_in_my_shop
sample

`DROP TABLE`, `DROP DATABASE`: **삭제하기**

때로는 실수로 만들거나 더 이상 필요 없는 테이블이나 데이터베이스를 삭제해야 할 때가 있다. `DROP` 명령어는 구조 자체를 완전히 삭제하므로 사용에 매우 신중해야 한다.

sample 테이블을 그 안의 모든 데이터와 함께 영구적으로 삭제한다.

```
DROP TABLE sample;
```

`SHOW TABLES;` 로 사라진 테이블을 확인해보자. 아무런 데이터도 조회되지 않을 것이다.

[실행 결과]

Tables_in_my_shop
없음(비어있음)

이번에는 `my_shop` 데이터베이스를 삭제해보자.
그 안의 모든 테이블, 데이터와 함께 영구적으로 삭제한다.

```
DROP DATABASE my_shop;
```

`SHOW DATABASES;`로 사라진 데이터베이스를 확인하자.

[실행 결과]

Database
information_schema
mysql
performance_schema
sys

- `my_shop` 데이터베이스가 제거된 것을 확인할 수 있다.

> ⚠ **주의: 실제 운영 중인 서비스에서 `DROP` 명령어, 특히 `DROP DATABASE`는 절대 함부로 사용해서는 안 된다. 모든 데이터가 사라지는 돌이킬 수 없는 재앙을 초래할 수 있다. 삭제 전에는 항상 백업이 되어 있는지, 그리고 정말 삭제해야 하는 대상이 맞는지 두 번, 세 번 확인하는 습관을 들여야 한다.**

이렇게 데이터베이스와 테이블을 만들고 삭제하는 방법을 모두 알아보았다.

앞으로의 예제를 위해 앞서 만든 데이터베이스와 테이블을 다시 생성하자.

```
-- 데이터베이스 생성
CREATE DATABASE my_shop;
```

```sql
-- 데이터베이스 선택
USE my_shop;

-- 테이블 생성
CREATE TABLE sample (
    product_id INT PRIMARY KEY,
    name VARCHAR(100),
    price INT,
    stock_quantity INT,
    release_date DATE
);
```

- 메뉴바 ➔ Query ➔ Execute(All or Selection)을 선택하면 모든 쿼리를 한 번에 실행할 수 있다.
  - 이때 마우스로 범위를 지정하면 지정한 범위의 쿼리만 실행된다. 마우스로 범위를 지정하지 말고 실행하자.

`SHOW TABLES;` 로 테이블이 잘 생성되었는지 다시 확인해보자.

[실행 결과]

Tables_in_my_shop
sample

데이터베이스 시작하기2

데이터베이스의 기본, CRUD 맛보기

이제 테이블이라는 틀을 만들었으니, 그 안에 실제 데이터를 넣고(Create), 읽고(Read), 수정하고(Update), 삭제하는(Delete) 방법을 알아볼 차례다. 이 네 가지 기본 작업을 묶어서 **CRUD**라고 부르며, 데이터 관리의 가장 핵심적인 기능이다.

`INSERT`: 데이터 넣기 (Create)

`INSERT`는 테이블에 새로운 행(Row)을 추가하는 명령어다. `sample` 테이블에 신상품을 하나 등록해 보자.

```sql

```

```
INSERT INTO sample (product_id, name, price, stock_quantity, release_date)
VALUES (1, '프리미엄 청바지', 59900, 100, '2025-06-11');
```

이 구문은 "`sample` 테이블의 `product_id`, `name`, `price`, `stock_quantity`, `release_date` 열에 각각 `1`, `'프리미엄 청바지'`, `59900`, `100`, `'2025-06-11'` 값을 넣어서 새로운 행을 추가해라" 라는 의미다.

숫자는 그냥 사용하면 되고, 문자열과 날짜는 작은따옴표(') 로 감싸주어야 한다.

[실행 결과]

```
1 row affected (0.00 sec)
```

- 1개의 행이 저장되었다는 뜻이다.

`SELECT` : 데이터 꺼내보기 (Read)

`SELECT`는 테이블에 저장된 데이터를 조회하는, 아마도 가장 많이 사용하게 될 명령어다.

`sample` 테이블에 있는 모든 데이터를 보고 싶다면 `*`(별표, asterisk)를 사용한다. `*`는 '모든 열(컬럼)'을 의미한다.

```
SELECT * FROM sample;
```

- `SELECT` : 원하는 열(컬럼)을 선택한다.
- `FROM` : 원하는 테이블을 선택한다.

[실행 결과]

product_id	name	price	stock_quantity	release_date
1	프리미엄 청바지	59900	100	2025-06-11

이 명령은 `sample` 테이블의 모든 열과 모든 행을 보여준다. 방금 우리가 `INSERT`한 '프리미엄 청바지' 데이터가 보일 것이다.
만약 상품의 이름과 가격만 보고 싶다면, 보고 싶은 열의 이름을 `SELECT` 다음에 직접 지정하면 된다. 이때 쉼표(,)로 각각의 열(컬럼)을 구분한다.

```
```

```
SELECT name, price FROM sample;
```

[실행 결과]

name	price
프리미엄 청바지	59900

`UPDATE` : 데이터 바꾸기 (Update)

`UPDATE` 는 이미 저장된 데이터의 내용을 수정하는 명령어다. '프리미엄 청바지'의 가격을 낮추어서 40000원으로 변경해 보자.

```
UPDATE sample
SET price = 40000
WHERE product_id = 1;
```

이 구문은 "`sample` 테이블에서, `product_id` 가 `1` 인 행을 찾아서, 그 행의 `price` 값을 `40000` 으로 변경해라" 라는 의미다.

- `UPDATE` : 데이터를 변경할 테이블을 지정한다.
- `SET` : 변경할 열(컬럼)과 그 값을 지정한다.
- `WHERE` : 변경할 행(로우)을 선택한다.

[실행 결과]

```
1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0
```

- WHERE 조건에 맞는 1개 행(Rows matched: 1)을 찾아 그중 1개 행의 데이터를 변경(Changed: 1)했음을 의미한다.

`SELECT * FROM sample;` 로 다시 조회해 보면 가격이 40000원으로 변경된 것을 확인할 수 있다.

[실행 결과]

product_id	name	price	stock_quantity	release_date
1	프리미엄 청바지	40000	100	2025-06-11

`DELETE` : 데이터 지우기 (Delete)

`DELETE` 는 테이블에서 특정 행을 삭제하는 명령어다. '프리미엄 청바지' 상품을 단종시켜서 삭제해 보자.

```
DELETE FROM sample
WHERE product_id = 1;
```

이 구문은 "`sample` 테이블에서, `product_id` 가 `1` 인 행을 삭제해라" 라는 의미다.

- `DELETE FROM` : 삭제할 테이블을 지정한다.
- `WHERE` : 삭제할 행(로우)을 선택한다.

[실행 결과]

```
1 row affected (0.00 sec)
```

- 1개의 행이 성공적으로 삭제되었음을 의미한다.

[삭제 확인]

`SELECT * FROM sample;` 로 다시 조회해 보면 데이터가 사라진 것을 확인할 수 있다.

```
0 row(s) returned
```

지금까지 우리는 데이터베이스와 테이블을 만들고, 그 안에 데이터를 CRUD하는 최소한의 과정을 모두 경험했다. 이것만으로도 여러분은 데이터베이스와 소통하는 첫걸음을 성공적으로 뗀 것이다. 다음 장에서는 우리가 사용한 이 SQL이라는 언어의 정체에 대해 조금 더 깊이 알아보겠다.

참고: 데이터베이스의 대소문자 구분

`SELECT` , `FROM` , `CREATE` 같은 SQL 키워드들은 대소문자를 구분하지 않는다. 따라서 `select` , `SELECT` , `SelecT` 는 모두 같은 뜻이다.
하지만 우리가 직접 지정한 테이블 명, 컬럼 명은 DBMS 종류나 서버 환경, 또는 설정에 따라 대소문자를 구분하기도 한다. 따라서 `sample` , `SAMPLE` 이 다른 결과를 보일 수 있다.

이러한 혼란을 피하기 위해 실무에서는 보통 같은 규칙을 따른다.

- SQL 키워드는 대문자로, 이름은 소문자로: `SELECT` , `FROM` , `CREATE` 같은 SQL 예약어는 대문자로 작성하고,

`products`, `customer_id`와 같이 우리가 직접 짓는 테이블/열 이름은 소문자로 작성하는 것이 일반적인 관례(convention)다.
    - 이 방식의 장점은 가독성이 좋다는 것이다.
    - 하지만 이렇게 하면 SQL을 작성할 때 대소문자를 중간에 계속 변환해야 하기 때문에 실무에서는 그냥 모두 소문자를 사용하는 경우도 많다.
- 이름은 소문자와 언더스코어(_) 조합으로: `my_shop`, `order_details`처럼 단어를 구분하는 언더스코어를 사용해 이름을 짓는다. 이렇게 하면 어떤 상황에도 일관되게 동작하는 코드를 작성할 수 있다.
- 데이터 자체는 대소문자를 구분한다.
    - `SELECT * FROM customers WHERE name = 'kim'` 과 `SELECT * FROM customers WHERE name = 'KIM'` 은 다른 결과가 나온다.

결론적으로, "어떤 환경에서든 동일하게 작동하는 코드를 만들기 위해, 데이터베이스와 테이블, 열 이름은 항상 소문자로 작성하는 습관을 들이는 것이 가장 좋다." 이 원칙만 지키면 대소문자 문제로 골치 아플 일은 없을 것이다.

SQL이란?

SQL(Structured Query Language)란?

우리는 앞서 `CREATE`, `INSERT`, `SELECT` 같은 몇 가지 영어 단어로 데이터베이스를 다뤄봤다. 이 명령어들의 집합이 바로 **SQL(Structured Query Language)**, 우리말로는 '구조화된 질의 언어'다.

여기서 또다시 질문을 던져보자. **왜 우리는 SQL이라는 새로운 언어를 배워야만 하는가?**
MySQL, Oracle, PostgreSQL, Microsoft SQL Server 등 세상에는 다양한 종류의 관계형 데이터베이스가 존재한다.
예를 들어서 현재 팀에서 MySQL을 사용해서 열심히 배웠는데, 다른 팀으로 이동하거나 다른 회사로 이직했을 때 Oracle을 사용한다고 가정해보자. 만약 이 데이터베이스들이 각자 다른 명령어를 사용한다면, 우리는 데이터베이스를 바꿀 때마다 새로운 언어를 처음부터 배워야 하는 끔찍한 상황에 놓일 것이다.

다행히도, SQL은 국제 표준 기구(ISO/ANSI)에 의해 표준이 정해진 **'관계형 데이터베이스의 표준 언어'**다. 마치 전 세계 공용어처럼, 대부분의 관계형 데이터베이스는 이 표준 SQL을 지원한다. 물론 각 데이터베이스마다 자신들만의 추가 기능이나 약간의 문법 차이는 있지만, `CREATE`, `SELECT`, `INSERT`와 같은 핵심적인 문법은 거의 동일하다. 따라서 한번 SQL을 제대로 배워두면, 어떤 관계형 데이터베이스를 만나더라도 금방 적응할 수 있다. 이것이 우리가 SQL을 배우는 가장 큰 이유다.

> **데이터베이스 방언(Dialect)**
> SQL 표준을 넘어서 각 DBMS는 자신들만의 추가 기능이나 추가 문법을 제공하는데, 이것을 방언(사투리, Dialect)라고 한다.

SQL의 또 다른 중요한 특징은 '선언적 언어(Declarative Language)'라는 점이다.
이게 무슨 말이냐 하면, 우리는 데이터베이스에게 **'어떻게(How)'** 데이터를 가져올지 지시하는 것이 아니라, **'무엇을(What)'** 원하는지만 선언적으로 명시한다는 뜻이다.

예를 들어, "서울에 사는 고객 명단을 찾아줘" 라고 SQL로 요청하면, 데이터베이스 관리 시스템(DBMS)이 내부적으로 가장 효율적인 방법을 스스로 찾아서 결과를 가져다준다. 따라서 우리는 그 복잡한 내부 과정을 알 필요가 없다.

우리는 그저 SQL로 무엇을 원하는지만 이야기하기만 하면 된다.

```sql
SELECT name FROM customers WHERE address = '서울';
```

이러한 특징 덕분에 우리는 데이터 처리의 본질에만 집중할 수 있다.

SQL 명령어의 4가지 종류

SQL의 모든 명령어는 그 목적과 기능에 따라 크게 4가지 그룹으로 나눌 수 있다. 이 4가지 분류를 이해하는 것만으로도 SQL의 전체적인 그림을 파악하고 앞으로 배울 내용을 체계적으로 정리하는 데 큰 도움이 된다. **이 4가지 종류만 알아도 SQL의 절반은 이해한 것이나 다름없다.**

우리가 쇼핑몰 창고를 관리하는 상황에 비유해 보자.

1. **DDL (Data Definition Language, 데이터 정의어)**
    - **목적**: 데이터의 '구조'를 정의하고 관리하는 언어다. 데이터 그 자체가 아니라, 데이터를 담을 그릇(테이블)이나 창고(데이터베이스)의 설계도를 만들고, 수정하고, 제거하는 역할을 한다.
    - **창고 비유**: 창고 건물을 짓고(`CREATE`), 내부 선반 구조를 바꾸거나(`ALTER`), 창고를 철거하는(`DROP`) 작업에 해당한다. 선반에 놓인 물건(데이터)은 건드리지 않고, 구조 자체에만 관여한다.
    - **주요 명령어**:
        - `CREATE`: 데이터베이스, 테이블 등의 구조를 생성한다.
        - `ALTER`: 이미 만들어진 테이블의 구조를 변경한다. (이후에 학습한다.)
        - `DROP`: 데이터베이스, 테이블을 완전히 삭제한다.

2. **DML (Data Manipulation Language, 데이터 조작어)**
    - **목적**: 테이블 안에 들어있는 실제 '데이터'를 직접 조작(추가, 조회, 수정, 삭제)하는 언어다. SQL에서 가장 빈번하게 사용되는 명령어들이 여기에 속한다.
    - **창고 비유**: 선반에 새 물건을 진열하고(`INSERT`), 진열된 물건의 목록을 확인하며(`SELECT`), 물건의 가격표를 바꾸고(`UPDATE`), 팔린 물건을 선반에서 치우는(`DELETE`) 작업에 해당한다.
    - **주요 명령어**:
        - `INSERT`: 테이블에 새로운 데이터를 추가한다.
        - `SELECT`: 테이블에서 데이터를 조회(검색)한다.
        - `UPDATE`: 기존 데이터를 수정한다.
        - `DELETE`: 기존 데이터를 삭제한다.

3. **DCL (Data Control Language, 데이터 제어어)**
    - **목적**: 데이터에 대한 접근 권한을 부여(`GRANT`)하거나 회수(`REVOKE`)하는 등, 데이터의 보안과 관련된 권한을 제어한다.
    - **창고 비유**: 창고 관리자가 직원들에게 창고 출입 키를 나눠주거나 특정 구역만 들어갈 수 있는 권한을 주는 것에 해당한다. 마케팅팀에게는 재고 조회 권한만 주고, 배송팀에게는 주소 조회 및 수정 권한을 주는 식이다.
    - **주요 명령어**:
        - `GRANT`: 특정 사용자에게 특정 작업에 대한 수행 권한을 부여한다.
        - `REVOKE`: 특정 사용자에게서 이미 부여한 권한을 회수한다.

> **DCL은 혼자 개발하고 공부할 때는 잘 사용하지 않지만, 여러 명의 개발자와 사용자가 함께 시스템을 사용하는 실무 환경에서는 보안을 위해 중요한 역할을 한다. 강의에서도 DCL은 다루지 않는다.**

4. **TCL (Transaction Control Language, 트랜잭션 제어어)**
    - **목적**: DML에 의해 수행된 데이터 변경 작업들을 하나의 '거래(Transaction)' 단위로 묶어서 관리하는 언어다. 작업의 일관성을 보장하여 데이터가 잘못되는 것을 방지한다.
    - **은행 계좌 이체 비유**: 당신이 친구에게 5만 원을 계좌 이체하는 상황을 생각해 보자. 이 작업은 사실 두 단계로 이루어져 있다.
        1. 내 계좌에서 5만 원이 빠져나간다 (UPDATE).
        2. 친구 계좌에 5만 원이 들어온다 (UPDATE).

        만약 1번 작업만 성공하고 시스템이 멈춰버린다면, 5만 원은 공중으로 증발해 버린다. 이런 사태를 막기 위해, 두 작업을 하나의 덩어리(트랜잭션)로 묶는다. 이 덩어리는 **두 작업이 모두 성공해야만 최종적으로 저장(`COMMIT`)되고, 둘 중 하나라도 실패하면 모든 작업을 없었던 일로 되돌린다(`ROLLBACK`)**.
    - **주요 명령어**:

- `COMMIT`: 트랜잭션의 모든 작업을 최종적으로 데이터베이스에 확정, 저장한다.
- `ROLLBACK`: 트랜잭션의 모든 작업을 취소하고 이전 상태로 되돌린다.

> **트랜잭션(TCL)은 실무에서 매우 중요하다.**
> 트랜잭션 관련 내용은 **실전 데이터베이스 기본** 강의에서 다룬다.

이처럼 SQL은 단순히 데이터를 조회하는 언어를 넘어, 데이터의 구조를 정의하고(DDL), 실제 데이터를 조작하며(DML), 접근 권한을 제어하고(DCL), 작업의 안정성을 보장하는(TCL) 매우 체계적이고 강력한 언어 체계다.

이제 우리는 이 분류를 머릿속에 넣고, 다음 장부터 쇼핑몰의 뼈대를 만드는 DDL을 시작으로 각 명령어들을 본격적으로, 그리고 깊이 있게 파고들어 갈 것이다.

데이터 타입

지금까지 우리는 DDL, DML과 같은 개념들을 배웠다. 이제 그중 DDL(데이터 정의어)을 사용해 우리 쇼핑몰의 핵심 데이터 구조, 즉 테이블들을 본격적으로 설계하고 구축할 것이다. 엉성하게 지은 건물은 작은 충격에도 무너지듯, 잘못 설계된 데이터베이스는 수많은 문제를 일으킨다. 기초를 단단히 다져보자.

앞서 맛보기로 `CREATE TABLE`을 사용해 봤다. 이제는 쇼핑몰 운영에 실제로 필요한 `고객(customers)`, `상품(products)`, `주문(orders)` 테이블을 제대로 설계해 볼 것이다. 이를 위해 더 다양한 데이터 타입과, 데이터의 규칙을 강제하는 '제약 조건(Constraints)'에 대해 알아야 한다. 순서대로 데이터 타입과 제약 조건을 알아보자.

앞서 맛보기로 `CREATE TABLE`을 사용하면서 문자, 숫자 같은 다양한 종류의 데이터 타입을 사용해보았다. 이번 시간에는 데이터 타입을 자세히 알아보자.

데이터 타입은 단순히 '숫자'나 '글자' 정도로만 구분되지 않는다. 어떤 종류의 숫자인지, 글자의 길이는 어떻게 되는지에 따라 타입을 세분화해서 선택해야 저장 공간을 효율적으로 사용하고 데이터의 정확성을 높일 수 있다. **왜 이렇게 다양한 데이터 타입이 필요할까?** 답은 **'효율성'**과 **'정확성'**에 있다. 알맞은 크기의 옷을 입어야 편한 것처럼, 데이터에 딱 맞는 타입을 지정해야 저장 공간(디스크)과 메모리를 아끼고, 데이터의 성격을 명확히 하여 오류를 줄일 수 있다.

숫자 타입

다음 표를 보자. 왜 이렇게 숫자 타입이 많을까? 예를 들어, '나이'를 저장하는 열과 '국가 총인구'를 저장하는 열에 똑같

은 크기의 데이터 타입을 쓰는 것은 낭비다. '나이'는 기껏해야 0~150 사이일 텐데, 수십억까지 표현 가능한 타입을 쓰는 것은 작은 물건을 담으려고 거대한 컨테이너를 빌리는 것과 같다. 그래서 데이터의 예상 범위에 맞는 타입을 선택하는 것이 중요하다.

> **중요**
> 지금부터 설명하는 내용을 절대! 외우지 말자
> 개념만 이해하고 자세한 항목들은 필요할 때 찾아보면 충분하다!

정확한 숫자 타입 (Exact Value Numeric Types)
이 타입들은 소수점 이하까지 정확한 값을 저장해야 할 때 사용된다. 주로 정수나 고정 소수점 숫자를 다룰 때 유용하다.

타입	저장 공간 (Bytes)	설명
`INTEGER`, `INT`	4	가장 일반적으로 사용되는 정수 타입 (-2,147,483,648 ~ 2,147,483,647)
`TINYINT`	1	매우 작은 정수를 저장할 때 사용 (-128 ~ 127)
`SMALLINT`	2	작은 정수를 저장할 때 사용 (-32,768 ~ 32,767)
`MEDIUMINT`	3	중간 크기의 정수를 저장할 때 사용 (-8,388,608 ~ 8,388,607)
`BIGINT`	8	매우 큰 정수를 저장할 때 사용 (-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807)
`DECIMAL`, `NUMERIC`	가변	고정 소수점 숫자를 정확하게 저장할 때 사용. 금융 데이터와 같이 정확한 계산이 필요할 때 사용. `DECIMAL(M, D)` 형태로 사용하며, M은 총 자릿수, D는 소수점 이하 자릿수를 의미한다.
`BIT`	가변	하나 이상의 비트 값을 저장. `BIT(M)` 형태로 사용하며, M은 1에서 64까지의 비트 수를 지정한다.

참고: `UNSIGNED` 속성을 추가하면, 각 정수 타입은 0부터 시작하여 양수 범위만 저장하게 되며 저장할 수 있는 양수 값의 범위가 두 배로 늘어난다. 예를 들어 `UNSIGNED INT`는 0부터 4,294,967,295까지 저장할 수 있다.

근사치 숫자 타입 (Approximate Value Numeric Types)
이 타입들은 부동 소수점 숫자를 저장하며, 매우 크거나 매우 작은 숫자를 표현할 수 있지만 약간의 오차가 발생할 수 있

다.

타입	저장 공간 (Bytes)	설명
`FLOAT`	4	단정밀도 부동 소수점 숫자. 정밀도가 아주 중요하지 않은 실수 값에 사용된다.
`DOUBLE`	8	배정밀도 부동 소수점 숫자. `FLOAT` 보다 더 큰 범위와 높은 정밀도를 가진다. 과학 계산이나 공학 데이터에 주로 사용된다.

문자열 타입

`VARCHAR(n)` : 최대 `n` 글자까지 저장되는 **가변 길이** 문자열.

- 이름, 주소처럼 길이가 제각각인 데이터에 사용하면 저장 공간을 효율적으로 쓸 수 있다.
- '홍길동'(3글자)을 `VARCHAR(10)` 에 저장하면 실제 데이터 길이인 3글자만큼의 공간만 차지한다.
- '김네이트'(4글자)을 `VARCHAR(10)` 에 저장하면 실제 데이터 길이인 4글자만큼의 공간만 차지한다.

`CHAR(n)` : 항상 `n` 글자 길이를 차지하는 **고정 길이** 문자열.

- '남' (1글자)을 `CHAR(2)` 에 저장하면, 나머지 1글자는 공백으로 채워져 무조건 2글자의 공간을 차지한다.
- 성별 코드('M'/'F')나 우편번호, 국가 코드('KR'/'US')처럼 **항상 길이가 정해져 있는 데이터**에 사용하면, `VARCHAR` 보다 아주 약간의 이점을 가질 수 있다. 길이가 항상 같으니, 데이터를 찾기 위해 길이를 확인할 필요가 없기 때문이다.

`TEXT` : 매우 긴 텍스트를 저장할 때 사용한다.

- `TEXT` 는 주로 상품의 상세 설명, 장문의 리뷰, 블로그 게시글 본문처럼 긴 글 작성에 적합하다.
  - 자주 사용하지는 않지만 더 긴 텍스트를 보관할 수 있는 `MEDIUMTEXT` , `LONGTEXT` 등도 있다.

VARCHAR vs CHAR 핵심 차이점

구분	`VARCHAR` (Variable Character)	`CHAR` (Character)
저장 공간	**가변 길이**로, 입력된 데이터의 실제 크기만큼만 공간을 차지한다. (데이터의 길이 확인을 위해 추가로 1~2바이트의 길이 정보 저장 공간 필요)	**고정 길이**로, 선언한 길이만큼의 공간을 항상 차지한다. 입력된 데이터가 선언된 길이보다 짧으면 나머지를 공백으로 채운다.

최대 길이	65,535자 (UTF-8과 같은 멀티바이트 문자셋 사용 시 실제 저장 가능 글자 수는 줄어듦)	255자
공간 효율성	데이터 길이가 다양할 때 공간을 효율적으로 사용한다.	데이터 길이가 모두 동일할 때 효율적이며, 가변 길이를 저장하면 공간 낭비가 발생한다.
성능	데이터 길이가 자주 변경되면, 행의 크기가 변해 데이터 단편화(fragmentation)나 페이지 분할이 발생하여 성능 저하의 원인이 될 수 있다.	데이터 길이가 고정되어 있어 업데이트 시에도 행의 크기가 변하지 않으므로, 데이터 수정이 빈번할 때 VARCHAR보다 유리할 수 있다.

실무에서의 선택 방향: `VARCHAR`의 보편적 사용

과거에는 "길이가 고정적이면 `CHAR`, 가변적이면 `VARCHAR`"라는 원칙이 강조되었지만, 현대 데이터베이스 환경에서는 다음과 같은 이유로 `VARCHAR`**를 우선적으로 고려하는 경우가 많다.**

`VARCHAR`를 선호하는 이유

1. **뛰어난 공간 효율성**: 대부분의 문자열 데이터는 길이가 가변적이다. 사용자 이름, 이메일 주소, 게시물 제목, 주소 등은 그 길이가 예측하기 어렵고 다양하므로 `VARCHAR`를 사용하는 것이 저장 공간을 크게 절약할 수 있다.
2. **유연성**: 초기에 예측한 길이보다 긴 데이터가 들어올 가능성에 대비하기 용이하다. `CHAR`는 고정 길이의 데이터가 아니면 불필요한 공간 낭비가 심해져 잘 사용하지 않게 된다.

그럼에도 `CHAR`를 고려하는 경우

`VARCHAR`가 보편적으로 사용되지만, **데이터 길이가 완벽하게 고정된 경우**, 예를 들어 주민등록번호(13자리), 국가 코드(2~3자리), 우편번호, MD5 해시값(32자리)처럼 모든 값이 예외 없이 동일한 길이를 갖는 경우에 `CHAR` 사용을 고려할 수 있다.

정리

실무에서는 데이터의 특성과 애플리케이션의 요구사항을 종합적으로 판단하여 결정하지만, 특별한 이유가 없다면 대부분의 상황에서 `VARCHAR`로 시작하는 것이 일반적인 접근 방식이다.

날짜와 시간 타입

- `DATE` : 날짜 정보('YYYY-MM-DD')만 저장한다.
- `DATETIME` : 날짜와 시간 정보('YYYY-MM-DD HH:MM:SS')를 함께 저장한다. 입력된 값을 그대로 저장한다.
- `TIMESTAMP` : `DATETIME`과 유사하게 날짜와 시간을 저장하지만, 특별한 기능이 있다.

- **타임존(Timezone) 자동 변환**: `TIMESTAMP`는 데이터를 저장할 때 현재 서버의 타임존을 기준으로 UTC(협정 세계시)로 변환하여 저장하고, 데이터를 조회할 때는 현재 서버의 타임존에 맞춰 다시 변환해서 보여준다.

날짜 및 시간 타입

타입	저장 형식	저장 공간	값의 범위	주요 특징 및 사용 사례
`DATE`	`YYYY-MM-DD`	3 바이트	`'1000-01-01'` ~ `'9999-12-31'`	**날짜만 저장한다.** 시간 정보가 필요 없을 때 사용한다. **사용 사례:** 생년월일, 제품 출시일, 기념일
`TIME`	`HH:MM:SS`	3 바이트	'-838:59:59' ~ '838:59:59'	**시간 정보 또는 경과 시간을 저장한다.** 매우 넓은 범위를 지원하여 단순한 시각 외에 시간의 양을 저장할 수 있다. **사용 사례:** 영업 시작/종료 시각, 비디오 재생 시간
`DATETIME`	YYYY-MM-DD HH:MM:SS	5 바이트	'1000-01-01 00:00:00' ~ '9999-12-31 23:59:59'	**날짜와 시간을 함께 저장한다.** 지정된 값을 그대로 저장하며, 타임존(Time Zone) 정보의 영향을 받지 않는다. **사용 사례:** 이벤트 예약 시간, 특정 시점의 로그
`TIMESTAMP`	YYYY-MM-DD HH:MM:SS	4 바이트	1970-01-01 00:00:01 UTC ~ '2038-01-19 03:14:07' UTC	**날짜와 시간을 함께 저장하며 타임존을 자동 변환한다.** 저장 시 서버 타임존을 기준으로 UTC로 변환하여 저장하고, 조회 시 현재 세션의 타임존에 맞춰 다시 변환해 준다. **사용 사례:** 게시물 작성일(`created_at`), 최종 수정일(`updated_at`)

`YEAR`	`YYYY`	1 바이트	`1901` ~ `2155`	**연도만 저장한다.** 4자리 연도를 저장하는 데 사용된다. **사용 사례:** 졸업 연도, 제조 연도

`DATETIME` **vs** `TIMESTAMP`

두 타입은 날짜와 시간을 모두 저장하지만, 결정적인 차이가 있어 구분해서 사용해야 한다.

구분	`DATETIME`	`TIMESTAMP`
타임존(Time Zone) 처리	**처리 안 함** 입력된 값을 그대로 저장하고 그대로 보여준다. 서버의 타임존 설정이 바뀌어도 저장된 값은 변하지 않는다.	**자동 변환 처리** 값을 UTC(협정 세계시)로 변환하여 저장한다. 조회하는 사용자의 타임존 설정에 따라 시간이 자동으로 변환되어 보인다.
저장 가능 범위	`1000년 ~ 9999년` (넓음)	`1970년 ~ 2038년` (제한적)
저장 공간	5 바이트 + α	4 바이트 + α
자동 초기화/업데이트	기능이 없었으나 MySQL 5.6.5부터 지원한다.	`DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP` 기능을 통해 행 생성/수정 시 자동으로 현재 시간을 기록할 수 있다.

현대의 애플리케이션 개발에서는 `TIMESTAMP` 보다는 `DATETIME` 사용을 권장한다.

- `TIMESTAMP`는 2038년까지의 데이터만 보관할 수 있는 문제가 있다.
- `TIMESTAMP`의 행의 생성/수정 시 자동으로 현재 시간을 기록해주는 기능을 이제는 `DATETIME`도 지원한다.
- `TIMESTAMP`의 저장공간이 약간 작지만 현대의 하드웨어와 DB에서는 미미한 차이
- 글로벌 서비스를 운영하는 경우 애플리케이션에서 `DATETIME`에 데이터를 보관할 때 UTC 시간을 기준으로 변경해서 보관하면 된다.

기타 타입

- `BLOB` : 이미지, 오디오, 비디오 같은 **이진(Binary) 대용량** 데이터를 저장한다.
- `ENUM` : 단일 선택 타입. 미리 정의한 값 목록 중 하나만 선택하여 저장할 수 있다.
- `SET` : 다중 선택 타입. 미리 정의한 값 목록 중 여러 개를 동시에 선택하여 저장할 수 있다.

> **실무 팁**
> 실무에서 ENUM, SET 타입은 잘 사용하지 않는다.

제약 조건

제약 조건은 테이블에 데이터를 저장할 때, 특정 규칙을 지키도록 강제하는 장치다. 이 규칙 덕분에 우리는 잘못되거나 일관성 없는 데이터가 입력되는 것을 원천적으로 차단할 수 있다. 데이터의 '무결성', 즉 데이터에 결점이 없는 상태를 유지하는 것이 제약 조건의 핵심 목표다. 여기서는 간략히 알아보겠다.

`NOT NULL` : **필수 입력 항목 지정**

- 이 제약 조건이 걸린 열에는 `NULL` 값(값이 없음)을 허용하지 않는다. 즉, 데이터를 `INSERT` 할 때 이 열의 값은 반드시 입력되어야 한다. 고객의 이름, 아이디, 상품의 가격 등은 비어있으면 안 되는 핵심 정보이므로 `NOT NULL` 을 설정해야 한다.

`UNIQUE` : **중복 불가 항목 지정**

- 이 제약 조건이 걸린 열의 값은 테이블 내에서 항상 고유해야 한다. 즉, 중복된 값을 허용하지 않는다. 고객의 아이디(ID), 이메일 주소, 사업자 등록번호 등은 다른 사람과 중복되면 안 되므로 `UNIQUE` 제약 조건을 사용한다.
- `PRIMARY KEY` 와의 차이점: `PRIMARY KEY` 는 테이블 당 단 하나만 존재할 수 있지만, `UNIQUE` 는 여러 열에 설정할 수 있다. `PRIMARY KEY` 는 `UNIQUE` 와 `NOT NULL` 속성을 모두 포함하는, 더 강력한 개념이다.

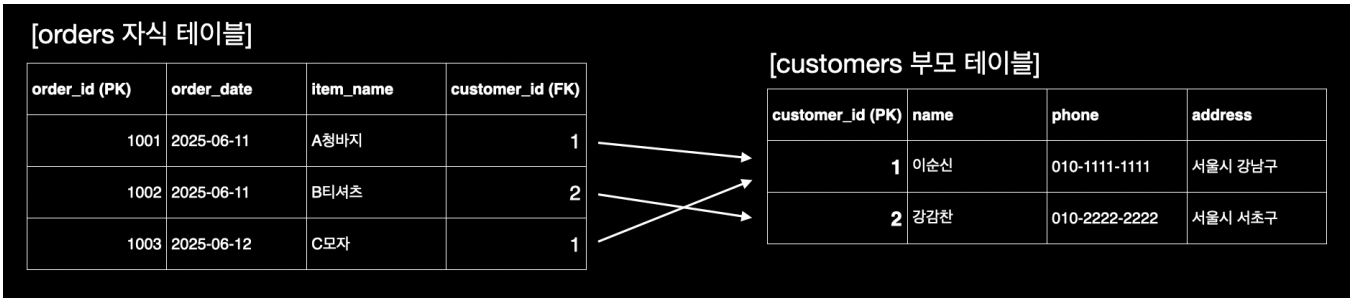
`PRIMARY KEY (PK)` : **테이블의 대표 식별자**

- 테이블의 모든 행을 유일하게 식별하는 열. `NOT NULL` 과 `UNIQUE` 의 특징을 모두 가진다. 모든 테이블에는 반드시 `PRIMARY KEY` 가 있어야 한다.
- `AUTO_INCREMENT` : MySQL에서 `PRIMARY KEY` 에 자주 사용하는 옵션이다. 정수 타입의 `PRIMARY KEY` 열에 이 옵션을 설정하면, 새로운 데이터가 추가될 때마다 1씩 자동으로 증가하는 번호를 할당해 준다. 우리가 직접 ID를 고민해서 넣을 필요가 없어 매우 편리하다. (뒤에서 실습한다.)

`FOREIGN KEY (FK)` : **테이블 간의 관계 설정**

- 한 테이블을 다른 테이블과 연결하는 관계의 고리. 참조하는 열의 값은 반드시 참조되는 테이블의 `PRIMARY`

`KEY` 값 중 하나여야 한다는 '참조 무결성'을 강제한다.



- 예를 들어 주문 테이블에 `customer_id` (FK) 값이 99가 입력된다고 가정해보자. `customers` 테이블에는 99번 고객 번호가 없다. 이런 경우 데이터베이스는 데이터의 입력을 막는다. (오류가 발생한다.)

`DEFAULT` : **기본값 설정**

- 데이터를 `INSERT` 할 때 특정 열의 값을 명시하지 않으면, 자동으로 설정된 기본값이 입력된다. 예를 들어, 회원 가입 시 포인트를 따로 주지 않으면 자동으로 0점이 들어가게 하거나, 상품 등록 시 '판매 상태'를 기본적으로 '판매중'으로 설정할 수 있다. 상품이 등록된 시간을 저장할 수 도 있다.

`CHECK` : 컬럼에 입력되는 값이 특정 조건을 만족하는지 검사

- 설명: 조건에 맞지 않는 데이터의 입력을 막는다.
- 예를 들어
  - 10보다 작은 숫자가 입력되지 않도록 막는다.
  - 18세 미만의 회원이 저장되지 않도록 막는다.

정리

데이터베이스 시작하기1

- `CREATE DATABASE` 로 데이터베이스를 만들고 `USE` 로 작업할 데이터베이스를 선택한다.
- `CREATE TABLE` 로 테이블을 설계하며 이때 각 열의 데이터 타입과 기본 키를 지정한다.
- `DESC` 로 테이블 구조를 확인하고 `SHOW` 명령어로 데이터베이스나 테이블 목록을 본다.
- `DROP` 명령어로 테이블이나 데이터베이스를 완전히 삭제할 수 있으므로 사용에 주의해야 한다.

데이터베이스 시작하기2

- CRUD는 데이터 생성(Create) 조회(Read) 수정(Update) 삭제(Delete)를 의미한다.
- `INSERT` 로 데이터를 추가하고 `SELECT` 로 데이터를 조회한다.
- `UPDATE` 로 기존 데이터를 수정하며 `DELETE` 로 데이터를 삭제한다.

- `UPDATE`와 `DELETE` 사용 시 `WHERE` 절로 특정 데이터를 지정하는 것이 중요하다.

SQL이란?

- SQL은 관계형 데이터베이스의 표준 언어다.
- 특정 데이터베이스에 종속되지 않고 대부분의 관계형 데이터베이스에서 사용 가능하다.
- SQL은 '무엇을' 원하는지 선언하므로 복잡한 내부 처리 과정을 알 필요 없다.
- SQL 명령어는 목적에 따라 DDL DML DCL TCL 네 가지로 분류된다.
- DDL은 데이터 구조를 정의하고 DML은 데이터를 조작한다.
- DCL은 데이터 접근 권한을 제어하고 TCL은 트랜잭션을 제어하여 작업의 일관성을 보장한다.

데이터 타입

- 데이터 타입은 저장 공간 효율성과 데이터 정확성을 위해 필요하다.
- 데이터의 예상 범위와 성격에 맞는 타입을 선택해야 한다.
- 주요 타입으로 숫자 문자열 날짜와 시간 타입이 있다.
- 숫자 타입에는 정수를 저장하는 `INT`와 정확한 소수를 저장하는 `DECIMAL` 등이 있다.
- 문자열 타입에는 가변 길이 `VARCHAR`와 고정 길이 `CHAR`가 있으며 대부분 `VARCHAR`를 사용한다.
- 날짜와 시간 타입에는 `DATE` `DATETIME` `TIMESTAMP`가 있고 타임존 처리 여부와 저장 범위에 차이가 있다.
- `TIMESTAMP` 보다는 `DATETIME`을 권장한다.

제약 조건

- 제약 조건은 데이터의 무결성을 보장하기 위해 테이블에 특정 규칙을 강제하는 장치다.
- `NOT NULL`은 값이 비어있는 것을 허용하지 않는다.
- `UNIQUE`는 중복된 값을 허용하지 않는다.
- `PRIMARY KEY`는 테이블의 각 행을 유일하게 식별하며 `NOT NULL`과 `UNIQUE` 특징을 모두 갖는다.
- `FOREIGN KEY`는 테이블 간의 관계를 설정하여 참조 무결성을 강제한다.
- `DEFAULT`는 값 미지정 시 기본값을 설정한다.
- `CHECK`는 특정 조건을 만족하는 값만 입력 및 변경을 허용한다.