8주차

15. 데이터 타입

컬럼의 데이터 타입과 길이를 선정할 때 주의해야 하는 사항

- 저장되는 값의 성격에 맞는 최적의 타입 선정
- 가변 길이 컬럼은 최적의 길이를 지정
- 조인 조건으로 사용되는 컬럼은 똑같은 데이터 타입으로 선정

▼ 1. 문자열(CHAR와 VARCHAR)

1.1 저장 공간

`CHAR`는 고정 길이로 실제 입력되는 컬럼 값에 따라 사용하는 저장 공간의 크기가 변하지 않는다. 실제 저장된 값의 유효 크기를 별도로 저장할 필요가 없어 추가 공간이 불필요하다.

`VARCHAR`는 가변 길이로 최대 저장할 수 있는 값의 길이는 제한되어 있지만 그 이하 크기의 값이 저장되면 그만큼 저장공간이 줄어든다. 하지만 저장된 값의 유효 크기가 얼마인지를 별도로 저장해야 하므로 1~2바이트의 추가 공간이 필요하다.

> **주의 ❗**
>
> MySQL에서는 하나의 레코드에서 `TEXT`와 `BLOB` 타입을 제외한 컬럼의 전체 크기가 64KB를 초과할 수 없다. 만약 `VARCHAR` 컬럼을 생성할 때 모두 합쳐 64KB를 넘어간다면 `VARCHAR` 타입이 `TEXT` 타입으로 자동 대체된다.
> 문자 집합에 따라 하나의 문자는 1~4바이트까지 공간을 사용할 수 있기 때문에 64KB 크기의 데이터가 65,536개의 글자를 저장할 수 있는 것은 아니다.

CHAR 타입과 VARCHAR 타입을 결정할 때 중요한 기준

- 저장되는 문자열의 길이가 대개 비슷한가?
- 컬럼의 값이 자주 변경되는가?

`VARCHAR(10)` 컬럼에 'ABCD'를 저장하면 길이 저장용 1바이트를 포함 총 5바이트(물론 문자 집합에 따라 다를 수 있음) 공간을 차지하고, 이후에 다음 컬럼의 데이터가 저장된다. 여기서 'ABCDE'와 같은 길이가 더 큰 값으로 바꾸면 자리가 없기 때문에 레코드 자체를 다른 공간으로 옮겨(Row migration) 저장해야 한다.

`CAHR(10)`이나 `VARCHAR(10)`으로 컬럼을 정의하면 10바이트가 아닌 10글자를 저장할 수 있는 공간을 의미한다.

- 일반적으로 영어를 포함한 서구권 언어는 각 문자가 1바이트이므로 10바이트가 사용된다.
- 한국어나 일본어와 같은 아시아권 언어는 각 문자가 최대 2바이트이므로 20바이트가 사용된다.

- UTF-8과 같은 유니코드는 최대 4바이트이므로 40바이트까지 사용된다.

주의❗

MySQL 서버에는 `utf8`과 `utf8mb4` 문자 집합이 별도로 존재한다. `utf8`은 5.5 이전 버전까지 UTF-8 인코딩을 저장하기 위해 사용되었으며 `utf8mb4`는 5.5 부터 지원되기 시작했다. MySQL 서버의 `utf8` 문자 셋은 한 글자당 최대 3바이트까지만 지원되었다.

- 1바이트: 아스키(ASCII) 문자
- 2바이트: 추가 알파벳 문자
- 3바이트: BMP(Basic Multilingual Plane) 문자
- 4바이트: SMP(Supplementary Multilingual Plane) & SIP(Supplementary Ideographic Plane & ...

이모티콘은 4바이트를 사용해야 한다. 이를 위해 utf8mb4라는 새로운 문자 집합을 도입하였다. `utf8`을 사용하던 테이블을 `utf8mb4` 문자 집합으로 전환하는 것은 아무런 문제가 없다.

`utf8mb3` 문자 집합도 만들어졌으며 8.0부터 `utf8` 문자 집합은 `utf8mb3`를 가리키는 별칭의로 정의되어있다. 곧 `utf8mb3` 문자 집합은 제거될 예정이다. Deprecated

1.2 저장 공간과 스키마 변경(Online DDL)

VARCHAR 데이터 타입을 사용하는 컬럼의 길이를 늘리는 작업은 길이에 따라 매우 빠르게 처리될 수 있지만 어떤 경우는 테이블에 대해 읽기 잠금을 걸고 레코드를 복사하는 작업이 필요할 수 있다. ⇒ 길이가 256바이트 이상으로 넘어갈 때 (`VARCHAR(63)`: $63 * 4byte = 252byte$, `VARCHAR(64)`: $64 * 4byte = 256byte$)

길이 저장 공간의 크기가 1바이트에서 2바이트로 변경되어야 하기 때문이다.

1.3 문자 집합(Character Set)

MySQL 서버에서 테이블의 각 컬럼은 모두 서로 다른 문자 집합을 사용해 문자열 값을 저장할 수 있다. 편의를 위해 서버나 DB, 테이블 단위로 기본 문자 집합을 설정하는 기능을 제공한다. 문자 집합은 `CHAR`, `VARCHAR`, `TEXT` 타입 컬럼에만 설정할 수 있다.

`utf8mb4` 쓰자.

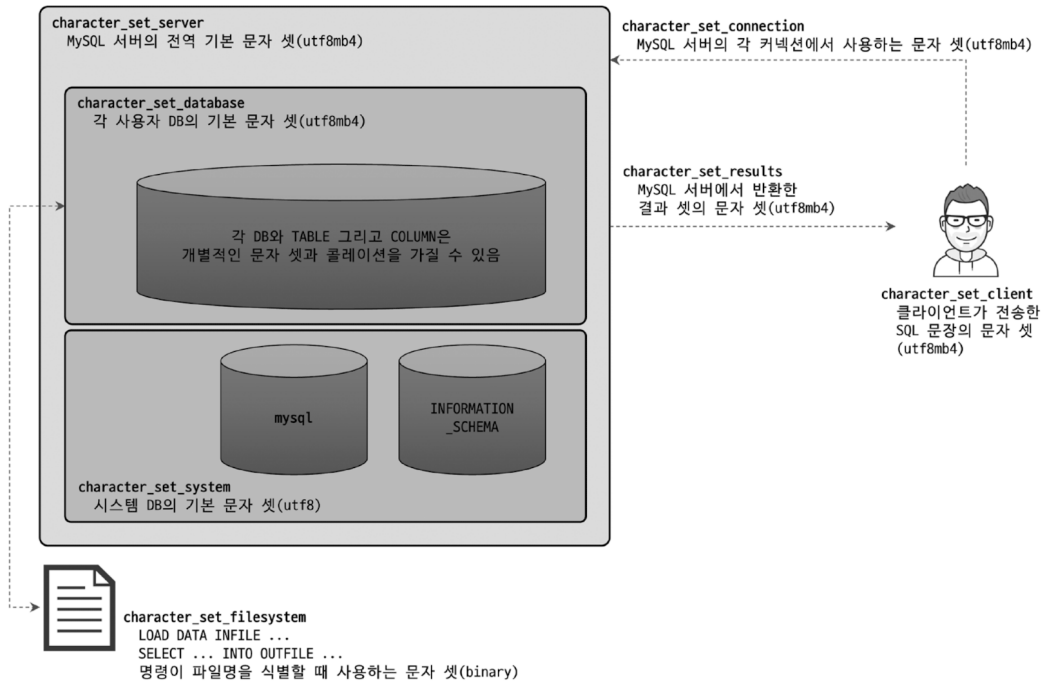
ANSI 표준에서 하나의 문자 집합만 기본으로 사용 가능한 DB에서 다국어 지원을 위해 NCHAR 또는 NATIONAL CHAR 같은 컬럼 타입을 정의하고 있다. MySQL에서도 NCHAR 타입을 지원하지만 컬럼 단위로 집합을 선택 가능하므로 NCHAR 타입을 사용할 필요는 없다. MySQL에서 NCHAR 타입을 사용하면 UTF-8 문자 집합을 사용하는 `CHAR` 타입으로 생성된다.

`SHOW CHARACTER SET` 명령어로 사용 가능한 문자 집합을 확인 가능하다. `Default collation` 컬럼은 해당 문자 집합의 기본 콜레이션(컬럼 콜레이션은 명시하지 않고 문자 집합만 지정했을 때 설정되는 콜레이션)에 대한 정보이다.

MySQL 문자 집합 설정 시스템 변수

- `character_set_system`: 식별자를 저장할 때 사용하는 문자 집합(기본 `utf8`, 사용자가 설정 및 변경할 필요 없음)
- `character_set_server`: 기본 문자 집합(기본 `utf8mb4`)
- `character_set_database`: DB의 기본 문자 집합(기본 `utf8mb4`)

- `character_set_filesystem`: `LOAD DATA INFILE …` 또는 `SELECT … INTO OUTFILE` 문장을 실행할 때 인자로 지정되는 파일 이름을 해석할 때 사용되는 문자 집합(기본 `binary`, 파일명이 제대로 인식이 안 되면 `utf8mb4`로 변경해보자)
- `character_set_client`: MySQL 클라이언트가 보낸 SQL 문장을 MySQL 서버로 전송할 때 인코딩할 문자 집합(기본 `utf8mb4`)
- `character_set_connection`: MySQL 서버가 클라이언트로부터 전달받은 SQL 문장을 처리하기 위해(숫자를 문자로 변환할 때에도) 인코딩할 문자 집합(기본 `utf8mb4`)
- `character_set_results`: MySQL 서버가 쿼리의 처리 결과를 클라이언트로 보낼 때 사용하는 문자 집합(기본 `utf8mb4`)



문자 집합의 적용 범위 및 클라이언트와 서버 간의 문자 집합 변환

서버와 클라이언트간의 문자 집합 변환은 동일한 문자 집합이라면 변환되지 않는다.

1. 클라이언트로부터 쿼리를 요청했을 때의 문자 집합 변환

MySQL 서버는 클라이언트로부터 받은 메시지(SQL 문장 및 변숫값)를 `character_set_connection` 시스템 변수에 정의된 문자 집합으로 변환한다.

SQL 문장에 별도의 문자 집합이 지정된 리터럴(문자열)은 변환하지 않는데, 이 설정 지정자를 '인트로듀서'라고 한다.

```
SELECT emp_no, first_name
FROM employees
WHERE first_name = _latin1'Matt';
```

`_latin1` ⇒ 인트로듀서 (문자열 리터럴 앞에 `_`와 문자 집합의 이름을 붙여 표현)

2. 처리 결과를 클라이언트로 전송할 때의 문자 집합 변환

MySQL 서버는 쿼리 결과(결과 셋이나 에러 메시지)를 `character_set_results` 변수에 설정된 문자 집합으로 변환해 클라이언트로 전송한다. 결과 셋에 포함된 컬럼의 값이나 컬럼명과 같은 메타데이터도 모두 인코딩한다.

```
// 개별 변경
SET character_set_client = 'utf8mb4';
SET character_set_results = 'utf8mb4';
SET character_set_connection = 'utf8mb4';

// 한번에 변경
SET NAMES utf8mb4; // 현재 접속된 커넥션에서만 유효
CHARSET utf8mb4; // MySQL 클라이언트 프로그램이 재시작될 때까지 유지
```

1.4 콜레이션(Collation)

콜레이션은 문자열 컬럼 값에 대한 비교나 정렬 순서를 위한 규칙을 의미한다. 비교나 정렬 작업에서 대소문자 구분 규칙을 정의하는 것이다.

테이블의 각 컬럼에 대해 독립적인 문자 집합과 콜레이션을 가진다. 각 문자열의 컬럼 값을 비교하거나 정렬할 때 문자 집합뿐 아니라 콜레이션의 일치 여부에 따라 결과가 달라지며 쿼리 성능 또한 영향을 받는다.

1. 콜레이션 이해

문자 집합은 2개 이상의 콜레이션을 가지고 있다. 하나의 문자 집합에 속한 콜레이션은 다른 문자 집합과 공유해 사용할 수 없다. 테이블 또는 컬럼에 문자 집합만 지정하면 해당 문자 집합의 기본 콜레이션이 지정된다. 반대로 콜레이션만 지정하면 해당 콜레이션이 소속된 문자 집합이 지정된다.

`SHWO COLLATION` 명령어로 콜레이션 목록을 확인할 수 있다.

3개의 파트로 구성된 콜레이션 이름

- 1번째 파트: 문자 집합의 이름
- 2번째 파트: 문자 집합의 하위 분류
- 3번째 파트: 대문자나 소문자 구분 여부(`ci`(Case Insensitive) ⇒ 구분 안 함, `cs`(Case Sensitive) ⇒ 구분)

2개의 파트로 구성된 콜레이션 이름

- 1번째 파트: 문자 집합의 이름
- 2번째 파트: 항상 `bin` 키워드가 사용(binary). 이진 데이터로 관리되는 문자열 컬럼은 별도의 콜레이션을 가지지 않음. 실제 문자 데이터의 바이트 값을 기준으로 수행

`utf8mb4` 문자 집합의 콜레이션 이름은 복잡해졌다. `utf8mb4_0900`으로 시작하는 콜레이션에서 0900은 UCA(Unicode Collation Algorithm)(문자 비교 규칙 정도로 이해하면 됨)의 버전을 의미한다. UCA의 버전이 올라갈수록 문자 간의 정렬 순서와 비교 규칙은 더 정교해지고 언어별 특성이 더 잘 반영된 버전이라고 볼 수 있다.

`utf8mb4_0900_ai_ci`와 같이 `ai` 또는 `as`를 포함하는 경우는 액센트를 가진 문자(Accent Sensitive)와 그렇지 않은 문자(Accent Insensitive)들을 정렬 순서상 동일 문자로 판단할지 여부를 나타낸다.

> **주의 ❗**
>
> MySQL 서버는 인코딩된 상태로 저장된 문자열을 가져와 각 인코딩된 바이트 값에 해당하는 콜레이션 값으로 매칭시킨 다음 비교를 수행하게 된다. 즉, 저장된 문자열의 바이트 값은 직접 비교 대상이 아니다.

`utf8mb4` 문자 집합이나 `euckr`과 같이 별도 `cs` 계열의 콜레이션을 가지지 않는 문자 집합을 비교할 때 대소문자를 구분하고 싶다면 `utf8mb4_bin` 또는 `euckr_bin` 처럼 `bin` 계열의 콜레이션을 사용하면 된다.

문자열 컬럼의 타입, 길이, 문자 집합, 콜레이션까지 일치해야 앞에서 배운 조인이나 WHERE 조건이 인덱스를 효율적으로 사용할 수 있다. 주의하자.

테이블 생성 시 문자 집합 및 콜레이션 적용 방법

```
CREATE DATABASE db_test CHARACTER SET = utf8mb4;

CREATE TABLE tb_member
(
    member_id    VARCHAR(20)  NOT NULL COLLATE latin1_general_cs,
    member_name  VARCHAR(20)  NOT NULL COLLATE utf8mb3_bin,
    member_email VARCHAR(100) NOT NULL
);
```

`WHERE` 조건 검색은 대소문자를 구분하지 않고 실행하되 정렬은 대소문자를 구분해야 하는 경우 검색과 정렬 작업 중 하나는 인덱스 이용을 포기해야 한다. 이때는 주로 컬럼의 콜레이션을 `ci`로 만들어 검색은 인덱스를 충분히 이용할 수 있도록 하고, 정렬 작업은 인덱스를 사용하지 않는 명시적인 정렬(Using filesort) 형태로 처리하는 것이 일반적이다.

`SHOW CREATE TABLE` 명령으로 테이블 구조를 확인하면 기본 문자 집합이나 콜레이션을 사용하는 컬럼은 별도로 표시되지 않는다. 더 자세히 확인하려면 `information_schema`의 `COLUMNS` 뷰를 확인하면 된다.

```
SELECT TABLE_NAME, COLUMN_NAME, COLUMN_TYPE, CHARACTER_SET_NAME, COLLATION_NA
FROM information_schema.COLUMNS
WHERE TABLE_SCHEMA = 'employees'
  AND TABLE_NAME = 'tb_collate';
```

2. utf8mb4 문자 집합의 콜레이션

콜레이션 이름의 숫자는 비교 알고리즘 버전이다. 별도로 명시되지 않은 콜레이션은 UCA 4.0.0 버전이다.

Collation	UCA version
`utf8_unicode_ci`	4.0.0
`utf8_unicode_520_ci`	5.2.0
`utf8mb4_unicode_520_ci`	5.2.0
`utf8mb4_0900_ci`	9.0.0

콜레이션 이름에 Locale이 포함되어 있는지 여부로 언어에 종속적인 콜레이션과 언어에 비종속적인 콜레이션으로 구분 가능하다.

콜레이션	언어	표기
utf8mb4_0900_ai_ci	N/A	없음
utf8mb4_zh_0900_as_cs	중국어	zh
utf8mb4_la_0900_ai_ci	클래식 라틴	la 또는 roman
utf8mb4_de_pb_0900_ai_ci	독일 전화번호 안내 책자 순서	de_pb 또는 german2
utf8mb4_ja_0900_as_cs	일본어	ja
utf8mb4_ro_0900_ai_ci	로마어	ro 또는 romanian
utf8mb4_ru_0900_ai_ci	러시아어	ru
utf8mb4_es_0900_ai_ci	현대 스페인어	es 또는 spanish
utf8mb4_vi_0900_ai_ci	베트남	vi 또는 vietnamese

언어 비종속적 콜레이션은 문자 셋의 기본 정렬 순서에 의해 정렬 및 비교가 수행되며, 언어 종속적 콜레이션은 해당 언어에서 정의한 정렬 순서에 의해 정렬 및 비교가 수행된다.

`utf8mb4_general_ci`와 `utf8mb4_0900_ai_ci` 콜레이션의 성능 차이가 있지만 성능을 보고 선택하지 말고 콜레이션의 필요에 따라 선택하도록 하자.

기존 UCA 9.0.0 버전 이전의 콜레이션을 사용 중이었다면 충분히 테스트를 진행한 후에 9.0.0으로 변경하고, 새로운 서비스를 개발한다면 성능과 관련없이 9.0.0 콜레이션 사용을 권장한다.

5.7 버전에서는 `utf8mb4` 문자 집합의 기본 콜레이션이 `utf8mb4_general_ci` 였는데 8.0 부터는 `utf8mb4_0900_ai_ci` 로 변경되었다. 5.7부터 존재하던 테이블은 8.0에서 생성한 테이블과 콜레이션이 달라 두 테이블을 조인할 때 에러가 발생하거나 성능이 심각하게 저하될 수 있다. 이 문제를 해결하기 위해 `default_collation_for_utf8mb4` 시스템 변수를 제공한다. 이 시스템 변수에 `utf8mb4_general_ci` 를 설정하면 문자 집합이 `utf8mb4`로 설정될 경우 콜레이션들도 `utf8mb4_general_ci`로 초기화된다. 하지만 일시적으로 제공되는 기능이므로 영구적으로 사용하기에는 안정적이지 않을 수 있다.

MySQL 서버에 접속하는 애플리케이션의 Connection String에도 connectionCollation 파라미터를 추가하는 것을 권장한다.

```
jdbc:mysql://dbms_server:3306/DB?connectionCollation=utf8mb4_general_ci
```

8.0으로 새로 시작하는 애플리케이션이라면 `utf8mb4_0900_ai_ci` 콜레이션을 기본으로 사용하는 것이 좋다.

1.5 비교 방식

MySQL에서 문자열 컬럼을 비교하는 방식은 `CHAR`와 `VARCHAR`가 거의 동일하다. `CHAR` 타입 컬럼에 `SELECT`를 실행했을 때 사용되지 않는 공간에 공백 문자가 채워져 나오지 않는다. MySQL 서버에서 지원하는 대부분의 문자 집합과 콜레이션에서 `CHAR` 타입이나 `VARCHAR` 타입을 비교할 때 공백 문자를 뒤에 붙여 두 문자열의 길이를 동일하게 만든 후 비교를 수행한다.

기존에는 문자열 뒤에 있는 공백이 비교 결과에 영향을 미치지 않았다.(`SELECT 'ABC' = 'ABC  '` ⇒ `true`) 하지만 `utf8mb4` 문자 집합이 UCA 9.0.0을 지원하면서 문자열 뒤에 붙은 공백 문자들에 대한 비교 방식이 달라졌다. 문자열 뒤의 공백이 비교 결과에 영향을 미친다.

문자열 뒤 공백이 비교 결과에 영향을 미치는지에 대한 여부는 `information_schema` 데이터베이스의 `COLLATIONS` 뷰에서 `PAD_ATTRIBUTE` 컬럼의 값으로 판단할 수 있다.

```
SELECT COLLATION_NAME, PAD_ATTRIBUTE
FROM information_schema.COLLATIONS
WHERE COLLATION_NAME LIKE 'utf8mb4%';
```

`PAD SPACE`라고 표시된 콜레이션은 비교 대상 문자열의 길이가 같아지도록 문자열 뒤에 공백을 추가해 비교를 수행하고 `NO PAD`로 표시된 콜레이션은 그대로 비교한다. 대부분 `PAD SPACE`이고 `utf8mb4_0900`으로 시작하는 콜레이션만 `NO PAD`이다.

예외적으로 `LIKE` 조건으로 문자열 패턴을 비교할 경우 공백 문자가 유효 문자로 취급된다.

1.6 문자열 이스케이프 처리

`\`를 이용해 이스케이프 처리가 가능하다.(`\t`, `\n` 등)

> MySQL :: MySQL 8.3 Reference Manual :: 11.1.1 String Literals
> A string is a sequence of bytes or characters, enclosed within
> either single quote (') or double quote
> (") characters. Examples:
> https://dev.mysql.com/doc/refman/8.3/en/string-literals.html#character-escape-sequences

Escape Sequence	Character Represented by Sequence
`\0`	An ASCII NUL (`X'00'`) character
`\'`	A single quote (`'`) character
`\"`	A double quote (`"`) character
`\b`	A backspace character
`\n`	A newline (linefeed) character
`\r`	A carriage return character
`\t`	A tab character
`\Z`	ASCII 26 (Control+Z); see note following the table
`\\`	A backslash (`\`) character
`\%`	A `%` character; see note following the table
`\_`	A `_` character; see note following the table

마지막의 `\%`와 `\_`는 `LIKE`를 사용하는 패턴 검색 쿼리의 검색어에서만 사용 가능하다.

따옴표(`'`, `"`)의 경우에는 두 번 연속으로 표기해 이스케이프 처리가 가능하다. 홑따옴표로 문자열을 감쌀 때는 쌍따옴표를 한 번만 표기해도 입력된다.(반대도 마찬가지)

▼ 2. 숫자

숫자를 저장하는 타입은 값의 정확도에 따라 크게 참값(Exact value)과 근사값 타입으로 나눌 수 있다.

- 참값은 소수점 이하 값의 유무와 관계없이 정확히 그 값을 그대로 유지하는 것을 의미한다. 참값을 관리하는 데이터 타입으로는 `INTEGER`를 포함해 `INT`로 끝나는 타입과 `DECIMAL`이 있다.
- 근삿값은 흔히 부동 소수점이라고 불리는 값을 의미하며, 처음 컬럼에 저장한 값과 조회된 값이 정확하게 일치하지 않고 최대한 비슷한 값으로 관리하는 것을 의미한다. 근삿값을 관리하는 타입으로는 `FLOAT`와 `DOUBLE`이 있다.

값 저장 포맷에 따라 십진 표기법(`DECIMAL`)과 이진 표기법으로 나눌 수 있다.

- **이진 표기법**: 흔히 프로그래밍 언어에서 사용하는 정수나 실수 타입을 의미
  - 한 바이트로 한 자리나 두 자리 숫자만 저장하는 것이 아니라 256까지의 숫자(양수만 저장한다고 가정)를 표현할 수 있기 때문에 숫자 값을 적은 메모리나 디스크 공간에 저장 가능
  - MySQL의 `INTEGER`나 `BIGINT` 등 대부분 숫자 타입은 모두 이진 표기법 사용
- **십진 표기법**(`DECIMAL`): 숫자 값으 각 자릿값을 표현하기 위해 4비트나 한 바이트를 사용해 표기
  - MySQL의 십진 표기법을 사용하는 타입은 `DECIMAL`뿐
  - 금액처럼 정확하게 소수점까지 관리되어야 하는 값을 저장할 때 사용
  - 65자리 숫자까지 표현할 수 있으므로 `BIGINT`로도 저장할 수 없는 값을 저장 가능

근삿값은 저장할 때와 조회할 때의 값이 정확히 일치하지 않고, 유효 자릿수를 넘어가는 소수점 이하의 값은 계속 바뀔 수 있다. 또한 `STATEMENT` 포맷을 사용하는 복제에서는 소스 서버와 레플리카 서버 간 데이터 차이가 발생할 수 있다.

MySQL에서 `FLOAT`나 `DOUBLE`과 같은 부동 소수점 타입을 잘 사용하지 않는다. 십진 표기법을 사용하는 `DECIMAL` 타입은 이진 표기법을 사용하는 타입보다 저장 공간을 2배 이상 필요로 한다. 매우 큰 수나 고정 소수점을 저장해야 하는 것이 아니라면 일반적으로 `INTEGER`나 `BIGINT` 타입을 자주 사용한다.

2.1 정수

`TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, `BIGINT`(`DECIMAL` 제외)

> MySQL :: MySQL 8.3 Reference Manual :: 13.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT,
> MySQL supports the SQL standard integer types
> INTEGER (or INT) and
> SMALLINT. As an extension to the standard,
> https://dev.mysql.com/doc/refman/8.3/en/integer-types.html

- `SIGNED`: 음수와 양수 표현 가능
- `UNSIGNED`: 0보다 큰 양의 정수만 저장

`AUTO_INCREMENT` 컬럼과 같이 음수가 저장되지 않는 컬럼에 `UNSIGNED` 옵션을 명시하면 작은 데이터 공간으로 더 큰 값을 저장 가능하다.

`UNSIGNED` 옵션은 인덱스 사용 여부에 영향을 미치지 않는다.

2.2 부동 소수점

`FLOAT`, `DOUBLE`

> MySQL :: MySQL 8.3 Reference Manual :: 13.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE
> The FLOAT and DOUBLE types
> represent approximate numeric data values. MySQL uses four bytes
> for single-precision values and eight bytes for double-precision
> https://dev.mysql.com/doc/refman/8.3/en/floating-point-types.html

`FLOAT`는 정밀도를 명시하지 않으면 4바이트를 사용해 유효 자릿수 8개까지 유지하며, 정밀도를 명시하면 최대 8바이트까지 저장 공간을 사용한다. `DOUBLE`은 8바이트의 저장 공간을 사용해 최대 유효 자릿수 16개를 유지한다.

유효 소수점의 자릿수만큼 10을 곱해 정수로 만들어 그 값을 정수 타입의 컬럼에 저장하는 방법도 있다.

2.3 DECIMAL

고정 소수점

> MySQL :: MySQL 8.3 Reference Manual :: 13.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC
> The DECIMAL and NUMERIC
> types store exact numeric data values. These types are used when
> it is important to preserve exact precision, for example with
> https://dev.mysql.com/doc/refman/8.3/en/fixed-point-types.html

숫자 하나를 저장하는데 1/2바이트가 필요하다.

곱셈 연산은 DECIMAL 타입보다 BIGINT 타입이 조금 더 빠르다. ⇒ 소수가 아닌 정수를 저장하기 위해 DECIMAL 타입을 사용하는 것은 성능 + 공간 사용면에서 좋지 않다.

2.4 정수 타입 컬럼 생성 시 주의사항

`DECIMAL(20, 5)` 라고 정의하면 정수부를 15자리, 소수부를 5자리까지 저장할 수 있다. `DECIMAL(20)`은 정수부만 20자리까지 저장할 수 있다. `FLOAT`나 `DOUBLE` 타입은 저장 공간의 크기가 고정이므로 정밀도를 조절한다고 해서 저장 공간의 크기가 바뀌는 것은 아니다. `DECIMAL`은 저장 공간의 크기가 가변적인 데이터 타입이어서 `DECIMAL`의 정밀도는 저장 가능 자릿수를 결정함과 동시에 저장 공간의 크기까지 제한한다.

버전 5.7까지는 부동 소수점이나 고정 소수점이 아닌 정수 타입을 생성할 때에도 똑같이 BIGINT(10)과 같이 괄호로 값의 크기를 명시하는 문법이 지원되었다. 이는 저장되는 값의 길이를 지정하는 것이 아닌 표시할 자릿수를 의미할 뿐이다.(크기 고정임) 8.0부터는 Deprecated

2.5 자동 증가(AUTO_INCREMENT) 옵션 사용

MySQL 서버의 `auto_increment_increment`와 `auto_increment_offset` 시스템 변수를 이용해 자동 증가값을 설정할 수 있다. `..._offset` 값을 5, `..._increment` 값을 10으로 설정하면 5, 15, 25, 35, ...와 같이 증가한다.

`AUTO_INCREMENT` 옵션을 사용한 컬럼은 반드시 그 테이블의 PK나 유니크 키 일부로 정의해야 한다. PK나 유니크 키가 여러 개의 컬럼으로 구성되면 `AUTO_INCREMENT` 속성의 컬럼값이 증가하는 패턴이 MyISAM과 InnoDB 스토리지 엔진에서 각각 달라진다.

- MyISAM: 자동 증가 옵션 컬럼이 PK나 유니크 키의 아무 위치에 사용될 수 있다.
- InnoDB: 자동 증가 옵션 컬럼으로 시작되는 인덱스를 생성해야 한다.

`AUTO_INCREMENT` 컬럼은 테이블당 하나만 사용 가능하다. `AUTO_INCREMENT`의 현재 증가 값은 테이블 메타 정보에 저장되어 있는데, `SHOW CREATE TABLE` 명령으로 조회 가능하다. 해당 명령어로 조회한 DDL 명령을 운영 DB에 실행하면 `AUTO_INCREMENT` 컬럼의 시작 값이 예상과 다를 수 있으므로 주의하자.

▼ 3. 날짜와 시간

Data Type	< MySQL 5.6.4	MySQL 5.6.4 ≤
`YEAR`	1byte	1byte
`DATE`	3byte	3byte
`TIME`	3byte	3byte + (ms 단위 저장 공간)
`DATETIME`	8byte	5byte + (ms 단위 저장 공간)
`TIMESTAMP`	4byte	4byte + (ms 단위 저장 공간)

ms 단위는 2자리당 1바이트씩 공간이 더 필요하다. 버전 8.0에서 `DATETIME(6)`은 5 + 3 = 8바이트를 사용한다. 1자리든 2자리든 1바이트를 사용한다.

날짜 타입 컬럼은 자체에 타임존 정보를 가지지 않으므로 `DATETIME`이나 `DATE` 타입은 현재 DBMS 커넥션의 타임존과 관계없이 클라이언트로부터 입력된 값을 그대로 저장하고 조회할 때도 변환 없이 출력한다. 하지만 `TIMESTAMP`는 항상 UTC 타임존으로 저장되므로 타임존이 달라져도 값이 자동으로 보정된다. `SET time_zone = …` 명령어로 타임존 설정

이미 데이터를 가지고 있는 MySQL 서버의 타임존(`system_time_zone` 변수)을 변경해야 한다면 타임존 설정뿐 아니라 `DATETIME` 타입 컬럼이 가지고 있는 값도 `CONVERT_TZ()` 함수를 이용해 변환해야 한다. `TIMESTAMP` 타입 값은 항상 UTC로 저장되므로 서버의 타임존을 변경한다고 해서 별도로 변환 작업이 필요하지는 않다.

`system_time_zone` 시스템 변수는 MySQL 서버의 타임존이며 일반적으로 운영체제의 타임존을 그대로 상속받는다. 시스템 타임존은 운영체제 계정 환경 변수를 변경하거나 `mysqld_safe`를 시작할 때 `--timezone` 옵션을 이용해 변경할 수 있다.

`time_zone` 시스템 변수는 MySQL서버로 연결하는 모든 클라이언트 커넥션의 기본 타임존을 의미한다. 물론 커넥션의 타임존은 응용 프로그램 코드에 의해 다른 값으로 언제든 변경될 수 있다. 아무것도 설정하지 않으면 SYSTEM으로 자동 설정되는데 이는 `system_time_zone` 시스템 변수 값을 그대로 사용한다는 의미이다.

MySQL 서버에 접속된 커넥션에서 시간 관련 처리(`NOW()` 함수나 `TIMESTAMP` 컬럼 초기화 등)를 할 때는 `time_zone` 시스템 변수의 영향만 받는다.

3.1 자동 업데이트

5.6 버전 이전까지는 `TIMESTAMP` 타입의 컬럼은 레코드의 다른 컬럼 데이터가 변경될 때마다 시간이 자동 업데이트되고, `DATETIME`은 그렇지 않은 차이를 가지고 있었다. 하지만 5.6 버전부터는 `TIMESTAMP`와 `DATETIME` 모두 `INSERT`와 `UPDATE` 문장이 실행될 때마다 해당 시점으로 자동 업데이트되게 하려면 테이블 생성 시 옵션을 정의해야 한다.

```
CREATE TABLE tb_autoupdate
(
    id            BIGINT NOT NULL AUTO_INCREMENT,
    title         VARCHAR(20),
    created_at_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at_ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMES
    created_at_dt DATETIME  DEFAULT CURRENT_TIMESTAMP,
    updated_at_dt DATETIME  DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMES
);
```