[8주차] 15

태그	Done
날짜	@2024년 3월 19일 → 2024년 3월 26일
제목	데이터 타입

CHAR에서 빈 공간은 공백으로 처리되는데

만약 애초에 데이터로 공백을 넣으면 어떻게 될까?

주어진 값이 CHAR(4) 및 VARCHAR(4) 열에 저장된 경우, 검색 시 CHAR 열에서 후행 공백이 제거되므로 열에서 검색되는 값이 항상 동일하지는 않습니다. 다음 예제는 이 차이를 보여줍니다:

https://dev.mysql.com/doc/refman/8.3/en/char.html

📌 15장 데이터 타입

- 칼럼의 데이터 타입 선정은 물리 모델링에서 중요한 작업중 하나
- 다음과 같은 주의사항이 존재함
  - **저장되는 값의 성격에 맞는 최적의 타입을 선정**
  - **가변 길이 칼럼은 최적의 길이를 지정**

    - **조인 조건으로 사용되는 칼럼은 똑같은 데이터 타입으로 선정**
- 무분별한 데이터 타입 사용, 최대 길이 값을 사용하는건 서비스 도중에 스키마의 변경을 필요로하기도 함, 근데 읽기 전용 모드로의 전환 작업이 필요할 수 있습니다. (너무 넉넉해도, 부족해도 문제)

✅ 15.1 문자열(CHAR와 VARCHAR)

- 문자열 칼럼을 사용할때는 CHAR, VARCHAR 중 어떤 타입을 사용할지 결정해야 함

15.1.1 저장 공간

- 공통점
    - 문자열을 저장할 수 있는 데이터 타입
- 차이점
    - `고정 길이(CHAR)`
        - 실제 입력되는 칼럼값의 길이에 따라 사용하는 저장 공간의 크기가 변하지 않음
        - CHAR 타입은 이미 저장 공간의 크기가 고정적
        - 실제 저장된 값의 유효 크기가 얼마인지 별도로 저장할 필요가 없으므로 → 추가공간 필요 X
    - `가변 길이(VARCHAR)`
        - 최대로 저장할 수 있는 값의 길이는 제한돼 있음
        - 다만 이하 크기의 값이 저장되면 그만큼 저장공간이 줄어듭니다.
        - VARCHAR 타입은 유효 크기가 얼마인지에 대한 정보 저장을 위해 1 ~ 2 바이트의 저장공간이 추가로 필요함
- 1개의 글자를 저장할때 CHAR(1), VARCHAR(1)에서 실제 사용되는 저장 공간
    - `공통`
        - 사용하는 문자 집합에 따라 1 ~ 4바이트까지 실제 저장 공간을 사용할 수 있음
    - `CHAR`
        - 위 공간 제외 추가 공간이 필요하지 않음
    - `VARCHAR`
        - 문자열의 길이를 관리하기 위한 1~2바이트 공간을 추가로 더 사용합니다.

        - 최대 2바이트로 표현할 수 있는 수 길이만큼 VARCHAR를 지정할 수 있음
            - $pow(2, 16) = 65,536$ 따라서 VARCHAR는 최대 65,536 바이트까지 지정 가능

- MySQL에서 칼럼의 전체 크기 제한 64KB
    - TEXT, BLOB을 제외하고 칼럼의 전체 크기가 64KB를 초과할 수 없습니다.
        - **위 칼럼 제외 하나의 테이블에서 타입이 나누어 쓸 수 있는 크기제한은 64KB**
        - 만약 이미 64KB를 쓰고 있는데 새로운 VARCHAR 타입을 생성하면 에러 혹은 TEXT 타입으로의 자동 변환이 발생합니다. → 주의 요망
    - **문자열 타입의 저장은 1문자와 1바이트를 구분함**
        - 1문자는 실제 저장되는 문자 집합에 따라 1 ~ 4바이트까지 다양함
        - VARCHAR는 이로인해 정확히 65,536문자를 저장하지 못할 수 있음
            - 아시아권의 글자라면 절반으로, UTF-8사용하면 1/4로 줄어듦

- CHAR, VARCHAR 사용 판단 기준
    - 저장되는 문자열의 길이가 대개 비슷한가?
    - 칼럼의 값이 자주 변경되는가?

- 칼럼의 값이 자주 변경 되는게 왜 고려사항이 될까?
    - 아래 쿼리를 실행하면 그림과 같이 레코드가 저장됨

```
CREATE TABLE tb_test (
        fd1 INT NOT NULL,
        fd2 CHAR(10) NOT NULL,
        fd3 DATETIME NOT NULL
);

INSERT INTO tb_test (fd1, fd2, fd3) VALUES(1, 'ABCD', '
```

그림 15.1 CHAR 타입이 저장된 상태

    - fd1 칼럼은 INTEGER, fd3 칼럼은 DATETIME으로 각각 고정 길이로 4, 8바이트를 사용합니다.
    - fd2칼럼은 10 바이트를 사용하면서 앞쪽의 4바이트만 유효값으로 채우고 나머지는 공백문자로 채워집니다.(Space Character)
  - fd2가 VARCHAR(10)이라면

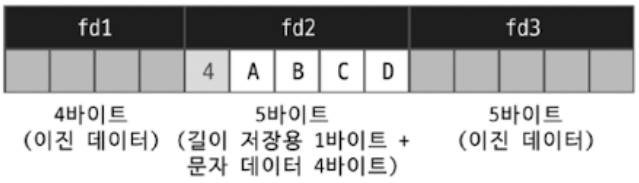

그림 15.2 VARCHAR 타입의 칼럼이 저장된 상태

    - 5바이트의 공간 차지
    - 첫번째 바이트에는 저장된 칼럼값의 유효한 바이트 수인 숫자 4가 저장됨
    - 2nd ~ 5th 바이트에는 실제 칼럼값이 저장됩니다.

- 위 결과에서 컬럼 값을 변경했을때 (ABCD → ABCDE)
  - `CHAR(10)`의 경우 10바이트가 준비돼 있으므로 그냥 변경되는 칼럼의 값을 업데이트 함
  - `VARCHAR(10)`의 경우 애초에 ABCD 4바이트만 저장할 수 없는 구조이므로 레코드 자체를 다른 공간으로 옮겨서 (Row migration) 저장해야 합니다.

- CHAR, VARCHAR 뒤에 인자로 전달되는 숫자 값의 의미를 알자
  - 바이트 크기가 아니라 문자의 수를 의미함 (바이트는 문자 집합에 따라 달라질 수 있음)
  - 즉, 동일한 10글자를 저장해도 바이트 수가 다를 수 있습니다.

- 일반적으로 영어를 포함한 서구권 언어는 각 문자가 1바이트를 사용하므로 10바이트를 사용한다.
- 한국어나 일본어와 같은 아시아권 언어는 각 문자가 최대 2바이트를 사용하므로 20바이트를 사용한다.
- UTF-8과 같은 유니코드는 최대 4바이트까지 사용하므로 40바이트까지 사용할 수 있다.

- 이모티콘과 같은 특수 문자의 발전으로 현재 mysql은 `utf8mb4` 라는 문자 집합을 사용합니다. 이는 한글자당 최대 4바이트까지 문자를 저장할 수 있습니다.

15.1.2 저장 공간과 스키마 변경(Online DDL)

- Online DDL을 통해 데이터 변경 도중에도 스키마를 변경할 수 있습니다.
- VARCHAR 타입은 사용하는 칼럼의 길이를 늘리는 작업은 길이에 따라 매우 빠르게 처리될 수 있지만, 어떤 경우에는 테이블에 대해 Shared Lock을 걸고 레코드를 복사하는 작업이 필요할 수 있습니다.
- 예제

```
CREATE TABLE test (
    id INT PRIMARY KEY,
    value VARCHAR(60)
) DEFAULT CHARSET=utf8mb4;
```

```
# 63으로 늘리는 경우는 아무 잠금 없이 매우 빠르게 변경됨
ALTER TABLE test MODIFY value VARCHAR(63), ALGORITHM=INPLA

# 64로 늘리면 INPLACE 알고리즘으로 스키마 변경이 허용되지 않습니다.
# 추가로 COPY 알고리즘은 Shared Lock도 필요로 합니다.
ALTER TABLE test MODIFY value VARCHAR(64), ALGORITHM=INPLA
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Re

# 가능
ALTER TABLE test MODIFY value VARCHAR(64), ALGORITHM=COPY,
```

  - utf8mb4는 한 글자가 최대 4바이트 이므로 60글자는 240바이트를 차지합니다.
  - 이때 `64글자는 256byte로 1byte로 표현할 수 있는 값이 범위를 벗어나기에 문자열 길이를 저장하는 공간의 크기를 2byte로 바꿔야 합니다.`

    - 따라서 문자열 길이를 저장하는 공간의 크기를 변경하는 동안 mysql 서버는 읽기 잠금을 걸어 아무도 데이터를 변경하지 못하도록 막고 테이블의 레코드를 복사합니다.
    - 이는 서비스를 점검 모드로 변경하거나, 가용성을 훼손합니다.
- VARCAHR 타입의 길이가 크게 변경될 것으로 예상되면 길이 저장 공간의 크기가 바뀌지 않도록 조금 크게 설계하는게 좋습니다.

15.1.3 문자 집합(캐릭터 셋)

- mysql 테이블의 칼럼은 모두 서로 다른 문자 집합을 사용해 문자열 값을 저장할 수 있음
    - 단, CHAR, VARCHAR, TEXT 타입의 칼럼에만 설정할 수 있음
- mysql은 최종적으론 칼럼 단위로 문자 집합을 관리함
    - 편의를 위해 mysql 서버, DB, 테이블 단위로 기본 문자 집합을 설정할 수 있는 기능도 있음
    - 기본적으로 테이블의 기본 문자 집합을 따라가지만, 별도로 칼럼에 대해 다른 문자 집합을 지정할 수 있음
- **최근 웹 서비스, 스마트폰 앱을 통한 여러 언어 동시 지원을 위해 주로 UTF-8(utf8mb4) 문자집합을 사용한느 추세입니다.**
- ANSI 표준은 다국어 지원을 위해 NCHAR or NATIONAL CHAR와 같은 칼럼 타입을 정의합니다.

- mysql 서버에서 사용 가능한 문자 집합은 `SHOW CHARACTER SET` 명령으로 확인 가능합니다.
    - `Default collation`으로 문자 집합의 기본 콜레이션이 무엇인지 보여줍니다.
    - 기본 콜레이션은 칼럼에 콜레이션을 명시하지 않고 문자 집합만 지정했을때 설정되는 콜레이션을 의미합니다.

- 문자 집합을 설정하는 시스템 변수

- character_set_system

  MySQL 서버가 식별자(Identifier, 테이블 명이나 칼럼 명 등)를 저장할 때 사용하는 문자 집합이다. 이 값은 기본적으로 utf8로 설정되며, 사용자가 설정하거나 변경할 필요가 없다.

- character_set_server

  MySQL 서버의 기본 문자 집합이다. DB나 테이블 또는 칼럼에 아무런 문자 집합이 설정되지 않을 때 이 시스템 변수에 명시된 문자 집합이 기본으로 사용된다. 이 시스템 변수의 기본값은 utf8mb4다.

- character_set_database

  MySQL DB의 기본 문자 집합이다. DB를 생성할 때 아무런 문자 집합이 명시되지 않았다면 이 시스템 변수에 명시된 문자 집합이 기본값으로 사용된다. 이 변수가 정의되지 않으면 character_set_server 시스템 변수에 명시된 문자 집합이 기본으로 사용된다. 이 시스템 변수의 기본값은 utf8mb4이다.

- character_set_filesystem

  LOAD DATA INFILE ... 또는 SELECT ... INTO OUTFILE 문장을 실행할 때 인자로 지정되는 파일의 이름을 해석할 때 사용되는 문자 집합이다. 여기서 주의해야 할 것은 데이터 파일의 내용을 읽을 때 사용하는 문자 집합이 아니라 파일의 이름을 찾을 때 사용하는 문자 집합이라는 점이다. 이 설정값은 각 커넥션에서 임의의 문자 집합으로 변경해서 사용할 수 있다. 기본값은 binary인데, LOAD DATA INFILE 명령이나 SELECT INTO OUTFILE 명령에서 파일명을 제대로 인식하지 못한다면 character_set_filesystem 시스템 변수를 utf8mb4로 변경하는 것이 좋다.

- character_set_client

  MySQL 클라이언트가 보낸 SQL 문장은 character_set_client에 설정된 문자 집합으로 인코딩해서 MySQL 서버로 전송한다. 이 값은 각 커넥션에서 임의의 문자 집합으로 변경해서 사용할 수 있다. 기본값은 utf8mb4이다. SQL 문장에서 문자열 리터럴에 대해 인트로듀서("_utf8mb4 'string_value'" 형태로 문자열 리터럴에 문자 셋을 설정하는 방법)가 사용된 경우에는 character_set_client 시스템 변수와 무관하게 개별적으로 설정된 문자 셋이 적용된다.

- character_set_connection

  MySQL 서버가 클라이언트로부터 전달받은 SQL 문장을 처리하기 위해 character_set_connection의 문자 집합으로 변환한다. 또한 클라이언트로부터 전달받은 숫자 값을 문자열로 변환할 때도 character_set_connection에 설정된 문자 집합이 사용된다. 이 변숫값 또한 각 커넥션에서 임의의 문자 집합으로 변경해서 사용할 수 있다. 기본값은 utf8mb4다.

- character_set_results

  MySQL 서버가 쿼리의 처리 결과를 클라이언트로 보낼 때 사용하는 문자 집합을 설정하는 시스템 변수다. 이 시스템 변수도 각 커넥션에서 임의의 문자 집합으로 변경해서 사용할 수 있다. 기본값은 utf8mb4다.

- `위 시스템 변수의 문자 집합의 적용 범위 및 클라이언트 서버간 문자 집합 변환` → 모르면 기본값 건드리지 않는편이 좋아보임

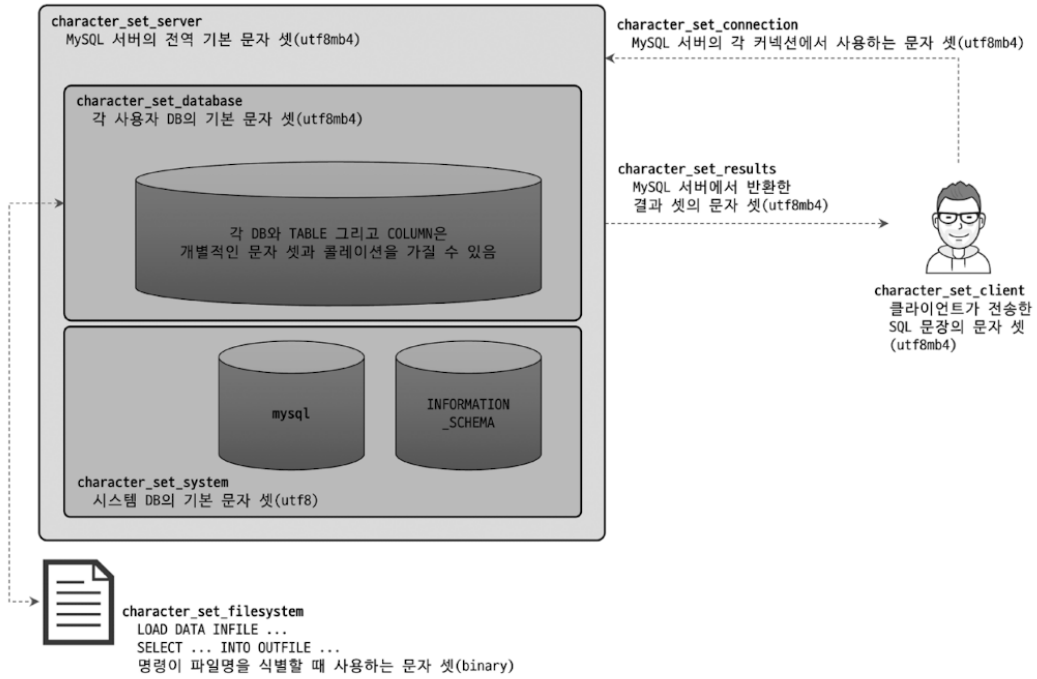

그림 15.3 문자 집합의 적용 범위 및 클라이언트와 서버 간의 문자 집합 변환

15.1.3.1 클라이언트로부터 쿼리를 요청했을 때의 문자 집합 변환

- mysql 서버는 `character_set_client`에 지정된 문자 집합으로 인코딩돼 있다고 판단합니다.
  이후 받은 문자열 데이터를
  `character_set_connection`에 정의된 문자 집합으로 변환합니다.
- 반면 SQL 문자에 별도의 문자 집합이 지정된 리터럴(문자열)은 변환 대상에 포함하지 않습니다.
  SQL 문장에서 별도로 설정하는 문자 집합을
  `인트로듀서`라고 합니다.

```
# character_set_connection으로 문자 집합이 변환된 후 처리됨
SELECT emp_no, first_name FROM employees WHERE first_name=

# latin1 문자 집합으로 first_name 칼럼 값과의 비교가 실행됩니다.
SELECT emp_no, first_name FROM employees WHERE first_name
```

  - 앞에 언더스코어("_")와 문자 집합의 이름을 붙여 사용

15.1.3.2 처리 결과를 클라이언트로 전송할 때의 문자 집합 변환

- 위 처리 이후 쿼리의 결과를 `character_set_results`의 문자 집합으로 변환해 클라이언트로 전송합니다.
  - 결과 셋에 포함된 칼럼의 값, 칼럼명과 같은 메타데이터도 모두 포함됨
  - 만약 변환 전의 문자 집합과 변환해야 할 문자 집합이 똑같다면 변환 작업은 모두 생략합니다.
    (그림 15.3 참조)
    - 다만 문자 집합은 물론이고 콜레이션까지 포함해서 같은지 다른지를 비교합니다.
- `character_set_client`, `_results`, `_connection` 3개의 시스템 변수는 클라이언트 GUI 도구에서 마음대로 변경 가능합니다. → 세션 변수이며 동적 변수임

```
mysql> SET character_set_client = 'utf8mb4';
mysql> SET character_set_results = 'utf8mb4';
mysql> SET character_set_connection = 'utf8mb4';

mysql> SET NAMES utf8mb4;
mysql> CHARSET utf8mb4;
```

  - 각각 설정(처음3개)하거나 한번에(마지막 2개) 변경할 수 있음
  - SET NAMES는 현재 접속된 커넥션에서만, CHARSET은 mysql 클라이언트 프로그램이 재시작되지 않은 상태에서 재접속시에도 유효하게 만듦

15.1.4 콜레이션(Collation)

- 문자열 칼럼의 값에 대한 비교나 정렬 순서를 위한 규칙을 의미함
- 비교, 정렬시 영문 대소문자를 같은 것을 처리할지, 아니면 더 크거나 작은 것으로 판단할지에 대한 규칙 정의
- 모든 문자열 타입 칼럼은 독립적인 문자 집합 및 콜레이션을 가짐
  - 지정하지 않으면 mysql 서버나 DB의 기본 문자 집합과 콜레이션이 자동 설정됨
- `콜레이션의 특성 때문에 문자열 비교, 정렬시에는 문자집합 + 콜레이션의 일치 여부에 따라 결과가 달라지고 쿼리 성능 또한 상당한 영향을 받습니다.`

15.1.4.1 콜레이션 이해

- 문자 집합은 2개 이상의 콜레이션을 가지고 있고, 하나의 문자 집합에 속한 콜레이션은 다른 문자 집합과 공유해서 사용할 수 없습니다.
- 테이블, 칼럼에 문자 집합만 지정해도 디폴트 콜레이션이 설정됩니다.
  (반대로 콜레이션만 지정하면 묵시적으로 해당 콜레이션이 소속된 문자 집합이 사용됨)

- `SHOW COLLATION`을 통해 사용가능한 목록을 확인할 수 있음
  - 콜레이션의 이름은 2개, 3개 파트로 구분돼 있는데 다음과 같은 의미를 가집니다.

    ```
    utf16_bin
    utf16_croatian_ci
    ```

  - 3개 파트
    - 첫 번째 파트는 문자 집합의 이름
    - 두 번째 파트는 해당 문자 집합의 하위 분류
    - 세 번째 파트는 대문자, 소문자의 구분 여부를 나타냄 `ci = 대소문자구분X`, `cs = 대소문자 별도 구분` (Case Sensitive)
  - 2개 파트
    - 첫 번째 파트는 문자 집합의 이름
    - 두 번째는 항상 bin이 사용됨 bin은 이진 데이터를 의미하며, 이진 데이터로 관리되기에 문자열 칼럼은 별도의 콜레이션을 갖지 않습니다. 비교 및 정렬 또한 문자 데이터의 바이트 값을 기준으로 수행합니다.

- `utf8mb4 문자 집합의 콜레이션은 좀 더 복잡`
  - utf8mb4_0900: 0900은 Unicode Collation Algorithm 버전을 의미함 (문자 비교 규칙)
  - utf8mb4_unicode_520_ci: 5.2.0 버전을 의미하며 버전 높을수록 문자 간 정렬, 비교 규칙이 정교하고 언어별 특성이 더 잘 반영됨
  - utf8mb4_0900_ai_ci: `ai` or `as`는 액센트를 가진 문자와 그렇지 않은 문자를 정렬 순서상 동일 문자로 취급할지 여부를 나타냅니다. (정렬 뿐만 아니라 동일 문자인지 아닌지의 검색 결과도 영향을 미침)

- 콜레이션이 대소문자, 액센트를 구별하지 않아도 저장될때는 원형 그대로 저장됨
- `mysql 서버는 인코딩된 상태로 저장된 문자열을 가져와 바이트 값에 해당하는 콜레이션 값으로 매칭시킨 후 비교를 수행합니다. 문자열의 바이트 값은 직접적인 비교 대상이 아닙니다.`

- 콜레이션 없이 문자 집합만 가질 수 없음
- `동일한 문자열 타입의 구분은 데이터 타입 + 문자 집합 + 콜레이션까지 일치해야 합니다.`
  `그래야 WHERE 조건이 인덱스를 효율적으로 이용할 수 있음`
- 테이블 생성시 문자 집합, 콜레이션 적용하기

```
# 디폴트 콜레이션은 utf8mb4_0900_ai_ci
# DB내에서 테이블, 칼럼에 문자 집합, 콜레이션 지정이 없으면 자동으로 0
CREATE DATABASE db_test CHARACTER SET=utf8mb4;

USE db_test

CREATE TABLE tb_member (
        # 대소문자 구분함
        member_id VARCHAR(20) NOT NULL COLLATE latin1_gene
        # 비교 및 정렬은 실제 문자 데이터의 바이트 값을 기준으로 함
        # 바이트 값으로 저장하므로 대소문자에서 차이가 발생하므로 대소
        member_name VARCHAR(20) NOT NULL COLLATE utf8_bin,
        # DB의 콜레이션을 따라감
        member_email VARCHAR(100) NOT NULL,
        ...
);
```

- 대표적인 latin 계열 문자의 _ci, _cs, _bin 정렬 규칙 테스트

```
CREATE TABLE tb_collate (
        fd_latin1_general_ci VARCHAR(10) COLLATE latin1_ge
        fd_latin1_general_cs VARCHAR(10) COLLATE latin1_ge
        fd_latin1_bin VARCHAR(10) COLLATE latin1_bin,
        fd_latin7_general_ci VARCHAR(10) COLLATE latin7_ge
);

INSERT INTO tb_collate VALUES
```

```
('a','a','a','a'), ('A','A','A','A'), ('b','b','b','b'), (
('_','_','_','_') , ('-','-','-','-'), ('.','.','.','.') ,
```

```
SELECT fd_latin1_general_ci FROM tb_collate ORDER BY fd_la
+----------------------+
| fd_latin1_general_ci |
+----------------------+
| -                    |
| .                    |
| a                    |
| A                    |
| b                    |
| B                    |
| _                    |
| ~                    |
+----------------------+
# 대소문자의 구분 없이 정렬됨

SELECT fd_latin1_general_cs FROM tb_collate ORDER BY fd_la
+----------------------+
| fd_latin1_general_cs |
+----------------------+
| -                    |
| .                    |
| A                    |
| a                    |
| B                    |
| b                    |
| _                    |
| ~                    |
+----------------------+
# 대소문자 구분하여 정렬됨, 대문자가 더 먼저 정렬

SELECT fd_latin1_bin FROM tb_collate ORDER BY fd_latin1_bi
+---------------+
| fd_latin1_bin |
+---------------+
| -             |
```

```
| .              |
| A              |
| B              |
| _              |
| a              |
| b              |
| ~              |
+----------------+
# 대문자만 먼저 정렬되고 소문자가 정렬됨

SELECT fd_latin7_general_ci FROM tb_collate ORDER BY fd_la
+----------------------+
| fd_latin7_general_ci |
+----------------------+
| -                    |
| .                    |
| _                    |
| ~                    |
| a                    |
| A                    |
| b                    |
| B                    |
+----------------------+
# 특수문자만 먼저 정렬하고 알파벳을 그 다음으로 정렬됨
```

  - `WHERE 조건 검색시 대소문자 구분 없이 실행하되 정렬은 대소문자를 구분해야 할 때는 두 작업 중 하나는 인덱스 이용하기를 포기` 해야 합니다.
  - `주로 칼럼 콜레이션을 _ci로 만들어 검색은 인덱스를 충분히 이용하게 하고, 정렬은 인덱스 사용 없는 명시적인 정렬 Using filesort 형태로 처리하는것이 일반적임`
  - 인덱스를 모두 이용하고 싶다면 정렬을 위한 콜레이션을 사용하는 칼럼을 하나 추가하는 방법도 생각해볼 수 있습니다.

- SHOW CREATE TABLE로 구조를 보면 디폴트 문자 집합, 콜레이션을 사용한 칼럼을 별도 표시하지 않습니다. 이때는 information_schema DB의 COLUMNS 뷰를 확인할 수 있습니다.

```
SELECT table_name, column_name,
            column_type, character_set_name, collation_nam
```

```
FROM information_schema.columns
WHERE table_schema='test' AND table_name='tb_collate';
+------------+----------------------+-------------+-------
| TABLE_NAME | COLUMN_NAME          | COLUMN_TYPE | CHARAC
+------------+----------------------+-------------+-------
| tb_collate | fd_latin1_bin        | varchar(10) | latin1
| tb_collate | fd_latin1_general_ci | varchar(10) | latin1
| tb_collate | fd_latin1_general_cs | varchar(10) | latin1
| tb_collate | fd_latin7_general_ci | varchar(10) | latin7
+------------+----------------------+-------------+-------
```

15.1.4.2 utf8mb4 문자 집합의 콜레이션

- 숫자 값이 포함된 콜레이션은 콜레이션의 비교 알고리즘 버전입니다.

콜레이션	UCA 버전
uf8_unicode_ci	4.0.0
utf8_unicode_520_ci	5.2.0
utf8mb4_unicode_520_ci	5.2.0
utf8mb4_0900_ai_ci	9.0.0

- 또한 Locale 값 여부로 언어에 종속적인지 아닌지를 알 수 있습니다.

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

    - 언어에 종속적인 정렬 순서 필요하면 고려해볼만함
- UCA 9.0.0 버전
    - 모든 문자를 `NO PAD` 옵션으로 문자 공백을 유효하게 생각하여 비교작업이 처리됩니다.
    - 특정 케이스에는 UCA 9.0.0 버전이 이전 버전보다 더 빠르게 작동된다고 하지만 큰 영향은 없습니다.
        - 다만 성능적인 이유로 콜레이션을 결정하는 방식보다는 콜레이션의 필요에 따라 결정해야 합니다.
    - 9.0.0은 NO PAD 옵션이 있으므로 이전 버전에서 9.0.0으로 변경하는 작업은 위험할 수 있으니 테스트를 통해 검사 이후 변경하는것이 좋고, 새로운 서비스 개발이라면 성능과 관계없이 UCA 9.0.0 사용을 권장합니다.
    - 5.7 → 8.0 업그레이드 할 때 my.cnf 설정 파일에 콜레이션 설정 없이 `문자 집합`에 대한 설정만 추가된 경우 콜레이션을 `utf8mb4_0900_ai_ci`로 초기화 합니다.

```
mysql> show global variables like '%character%';
+--------------------------------------------------+----
| Variable_name                                    | Val
+--------------------------------------------------+----
| character_set_client                             | utf
| character_set_connection                         | utf
| character_set_database                           | utf
| character_set_filesystem                         | bin
| character_set_results                            | utf
| character_set_server                             | utf
| character_set_system                             | utf
```

```
| character_sets_dir                                    | /op
| validate_password.changed_characters_percentage | 0
+-------------------------------------------------------+----

show global variables like 'collation%';
+----------------------+--------------------+
| Variable_name        | Value              |
+----------------------+--------------------+
| collation_connection | utf8mb4_0900_ai_ci |
| collation_database   | utf8mb4_0900_ai_ci |
| collation_server     | utf8mb4_0900_ai_ci |
+----------------------+--------------------+
```

- 기본 콜레이션 변경이 완료되면 새로 생성되는 DB, 테이블은 모두 해당 콜레이션을 따라갑니다.(`utf8mb4_0900_ai_ci`)
- 하지만 5.7 버전에서 생성한 테이블은 `utf8mb4_general_ci`를 사용하고 있었기 때문에 조인할때 에러가 발생하거나 심각한 성능 저하가 발생합니다.

- 위와 같은 문제 해결을 위해 `default_collation_for_utf8mb4` 시스템 변수를 제공합니다.
  - 해당 변수에 기존 콜레이션을 저장하면 8.0버전으로 올리고 문자 집합이 utf8mb4로 지정했을 경우 시스템 변수의 콜레이션을 다라갑니다
  - 하지만 일시적인 기능이므로, 당분간 콜레이션의 변경 예정이 없다면 my.cnf 파일에 콜레이션 관련 시스템 변수를 기존 콜레이션으로 고정하는게 좋습니다.
  - 또한 mysql 서버에 접속하는 응용 프로그램의 connection string에도 connectionCollation 파라미터를 추가하는 것을 권장합니다.

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

  - 설정이 완료됐고, connection string에 문자 셋, 콜레이션 관련 파라미터를 제거하면 자동으로 클라이언트 드라이버가 mysql 서버에 설정된 문자 집합 및 콜레이션을 가져와 사용합니다.
- 새로운 서비스는 utf8mb4_0900_ai_ci를 권장, 만약 콜레이션이 불일치하면 에에러가 발생합니다.

```
Error Code: 1267. Illegal mix of collations
```

```
(utf8mb4_general_ci,IMPLICIT) and (utf8mb4_0900_ai_ci,I
```

15.1.5 비교 방식

- 문자열 칼럼 비교 방식은 `VARCHAR`, `CHAR`가 거의 같습니다.
- CHAR 타입 칼럼
  - SELECT를 실행했을 때 다른 DB처럼 사용되지 않는 공간에 공백 문자가 채워져서 나오지 않습니다.
- 과거 mysql 서버에서 지원하는 대부분의 문자 집합 및 콜레이션은 CHAR나 VARCHAR 타입을 비교할때 공백 문자를 뒤에 붙여서 두 문자열의 길이를 동일하게 만든 후 비교를 수행합니다.
- 8.0 이상의 UCA 9.0.0 버전은 뒤에 붙어 있는 공백이 비교 결과에 영향을 미칩니다.

```
show global variables like 'collation%';
+----------------------+--------------------+
| Variable_name        | Value              |
+----------------------+--------------------+
| collation_connection | utf8mb4_0900_ai_ci |
| collation_database   | utf8mb4_0900_ai_ci |
| collation_server     | utf8mb4_0900_ai_ci |
+----------------------+--------------------+

SELECT 'ABC'='ABC   ' AS is_equal;
+----------+
| is_equal |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)

SELECT 'ABC'='   ABC' AS is_equal;
+----------+
| is_equal |
+----------+
```

```
|         0 |
+-----------+
```

- 문자열 뒤의 공백이 비교 결과에 영향을 미치는지 확인하는 방법은 information_schema의 COLLATION 뷰의 PAD_ATTRIBUTE 값으로 확인가능

```
SELECT collation_name, pad_attribute
FROM information_schema.COLLATIONS
WHERE collation_name LIKE 'utf8mb4%';

+-----------------------------+---------------+
| collation_name              | pad_attribute |
+-----------------------------+---------------+
| utf8mb4_0900_ai_ci          | NO PAD        |
| utf8mb4_0900_as_ci          | NO PAD        |
| utf8mb4_0900_as_cs          | NO PAD        |
| utf8mb4_0900_bin            | NO PAD        |
| utf8mb4_bg_0900_ai_ci       | NO PAD        |
| utf8mb4_bg_0900_as_cs       | NO PAD        |
| utf8mb4_bin                 | PAD SPACE     |
| utf8mb4_bs_0900_ai_ci       | NO PAD        |
| utf8mb4_bs_0900_as_cs       | NO PAD        |
| utf8mb4_croatian_ci         | PAD SPACE     |
...

// 0900은 모두 NO PAD
// PAD SPACE는 비교 대상 문자열의 길이가 같아지도록 문자열 뒤에 공백을
```

  - 위와 같은 이유로 0900으로 시작하는 콜레이션은 비교 대상 문자열 길이 차이가 많이 나도 더 빠른 비교 성능을 보입니다.

- 반면 LIKE 연산의 패턴 비교에선 공백 문자가 유효 문자로 취급됩니다.

15.1.6 문자열 이스케이프 처리

- 프로그래밍 언어처럼 `'\'` 를 이용해 이스케이프 처리가 가능함

이스케이프 표기	의미
\0	아스키(ASCII) NULL 문자(0x00)
\'	홑따옴표(')
\"	쌍따옴표(")
\b	백스페이스 문자
\n	개행문자(라인 피드)
\r	캐리지 리턴 문자 유닉스 계열 운영체제에서는 "\n"만 개행문자로 사용하며, 윈도우 계열 운영체제에서는 "\r\n"의 조합으로 개행문자를 사용한다.
\t	탭 문자
\\	백 슬래시(\) 문자

이스케이프 표기	의미
\%	퍼센트(%) 문자(LIKE의 패턴에서만 사용함)
_	언더 스코어(_) 문자(LIKE의 패턴에서만 사용함)

- 서로 다른 따옴표를 사용하면 감싸진 따옴표는 이스케이프 처리가 아니라 문자 그 자체로 인식합니다.

```
mysql> CREATE TABLE tb_char_escape (fd1 VARCHAR(100));

mysql> INSERT INTO tb_char_escape VALUES ('ab''ba');
mysql> INSERT INTO tb_char_escape VALUES ("ab""ba");
mysql> INSERT INTO tb_char_escape VALUES ("ab\'ba");
mysql> INSERT INTO tb_char_escape VALUES ('ab\"ba');
mysql> INSERT INTO tb_char_escape VALUES ('ab""ba');
mysql> INSERT INTO tb_char_escape VALUES ("ab''ba");

mysql> SELECT * FROM tb_char_escape;
+--------+
|    fd1 |
+--------+
| ab'ba  |
| ab"ba  |
| ab'ba  |
| ab"ba  |
| ab""ba |
| ab''ba |
+--------+
```

✅ 15.2 숫자

- 숫자를 저장하는 타입은 값의 정확도에 따라 `참값(Exact Value) 타입`과 `근삿값 타입`으로 나눠집니다.
  - `참값 타입`
    - 소수점 이하 값의 유무와 관계없이 정확히 그 값을 그대로 유지함
    - INTEGER를 포함해, INT로 끝나는 타입, DECIMAL이 있음
  - `근삿값 타입`
    - 부동 소수점이라고 불리는 값을 의미

        - 처음 칼럼에 저장한 값과 조회된 값이 정확하게 일치하지 않고 최대한 비슷한 값으로 관리하는 것을 의미합니다.
        - FLOAT, DOUBLE이 해당됨
- 값이 저장되는 포맷에 따라 십진 표기법과 이진 표기법으로 나뉩니다.
    - `이진 표기법`
        - 프로그래밍 언어에서 사용하는 정수, 실수 타입을 의미함
        - 한 바이트로 256까지의 숫자(양수만 저장했을때)를 표현할 수 있기에 메모리, 디스크 공간을 절약합니다.
        - NTEGER, BIGINT 등 대부분 숫자 타입은 모두 이진 표기법을 사용합니다.
    - `십진 표기법`
        - 숫자 값의 각 자릿값을 표현하기 위해 4비트나 한 바이트를 사용해서 표기함
        - 십진수가 아니라 디스크나 메모리에 십진 표기법으로 저장됨을 의미합니다.
        - 금액(돈)처럼 정확하게 소수점까지 관리돼야 하는 값을 저장할때 사용합니다.
        - DECIMAL만 해당됩니다.
            - 65자리 숫자까지 표현할 수 있으므로 BIGINT로도 저장할 수 없는 값을 저장할때 사용합니다.

- 근삿값은 조회시 값이 정확히 일치하지 않고 유효 자릿수를 넘어서면 소수점 이하의 값이 계속 바뀔 수 있습니다. STATEMENT 포맷을 사용하는 복제에선 소스 서버와 레플리카 서버간 데이터 차이가 발생 할 수 있기에 잘 사용하지 않습니다.
- 십진 표기법(DECIMAL)도 이진 표기법보다 저장 공간을 2배이상 필요로 하기에 매우 큰 숫자, 고정 소수점을 저장해야 하는 것이 아니라면 일반적으론 INTEGER, BIGINT를 사용합니다.

15.2.1 정수

- DECIMAL 타입을 제외하고 5가지의 정수를 저장할 수 있는 데이터 타입이 존재
- 저장 가능한 숫자 값의 범위 차위 제외 다른 차이는 없습니다.
- 직관적이기에 입력 가능한 수의 범위 내에서 최대한 적은 공간을 사용하는 타입을 선택하면 됩니다.

데이터 타입	저장공간 (Bytes)	최솟값 (Signed)	최솟값 (Unsigned)	최댓값 (Signed)	최댓값 (Unsigned)
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-263	0	263-1	264-1

- BIGINT는 8바이트로 2의 보수를 고려하면 `-2^63 ~ 2^63 - 1` 범위임
- UNSINGED를 붙이면 음수가 사라지므로 양수만 표현할 수 있게되어 양수의 범위가 2배 커집니다.
    - UNSINGED, SIGINED여부는 인덱스 사용 여부에 영향을 주지 않습니다. 하지만, 값의 범위가 다르므로 외래키 칼럼이나 조인 조건이 되는 칼럼은 일치시키는게 좋습니다.

15.2.2 부동 소수점

- 소수점의 위치가 고정적이지 않다는 의미 (Floating Point)
- 숫자 값의 길이에 따라 유효 범위의 소수점 자릿수가 바뀝니다. 따라서 정확한 유효 소수점 값을 식별하기 어렵기에 크다 작다 비교가 쉽지 않습니다.
- 근삿값을 저장하기에 동등 비교는 사용 불가
- `FLOAT`
    - 정밀도를 명시하지 않으면 4바이트를 사용해 유효 자릿수를 8개까지 유지함
    - 정밀도를 명시하면 최대 8바이트까지 저장 공간 사용 가능
- `DOUBLE`
    - 8바이트의 저장 공간을 필요 하며 최대 유효 자릿수를 16개까지 유지할 수 있음
- 복제에 참여하는 mysql 서버에서 부동 소수점을 사용하고 바이너리 로그 포맷이 STATEMENT 타입이라면 복제, 레플리카 서버간 데이터가 달라질 수 있음
    - 유효 정수부나 소수부는 달라지지 않겠지만 눈으로 판별하기 쉽지 않습니다.
- 부동 소수점 값을 저장해야 한다면 유효 소수점의 자릿수만큼 10을 곱해서 정수로 만들어 정수 타입 칼럼에 저장하는 방법도 생각해볼 수 있습니다.

```
mysql> CREATE TABLE tb_location (
         latitude INT UNSIGNED,
         longitude INT UNSIGNED,
         ...
       );

mysql> INSERT INTO tb_location (latitude, longitude, ..) VALUES
         (37.1422 * 10000, 131.5208 * 10000, ..);

mysql> SELECT latitude/10000 AS latitude, longitude/10000 AS longitude
       FROM tb_location
         WHERE latitude=37.1422 * 10000 AND longitude=131.5208 * 10000;
```

15.2.3 DECIMAL

- 금액, 대출, 이자 등과 같이 고정된 소수점을 정확히 관리해야 한다면 위치가 가변적이지 않은 DECIMAL 타입을 사용할 수 있습니다.
- DECIMAL 타입은 숫자를 저장하는 1/2 바이트가 필요합니다. (숫자의 자릿수 / 2 의 올림 만큼 필요)
- 정수 타입 BIGINT와 비교해보면 연산 속도에서도 DECIMAL이 조금 느린 모습을 보입니다.
- 따라서 단순히 소수가 아니라 정수를 관리하고 한다면 INTEGER, BIGINT를 사용하는 게 좋습니다.

15.2.4 정수 타입의 칼럼을 생성할 때의 주의사항

- 부동 소수점, DECIMAL을 이용해 칼럼을 정의할때는 괄호로 정밀도를 표시하는게 일반적임
  - DECIMAL(20, 5) 정수부를 15자리, 소수부를 5자리 저장할 수 있는 타입이 됨
  - DECIMAL(20) 정수부만 20자리까지 저장할 수 있는 타입
  - FLOAT, DOUBLE과 달리 DECIMAL은 저장 공간의 크기가 가변적인 데이터 타입이므로 정밀도를 지정하면 자릿수와 동시에 저장 공간의 크기까지 제한합니다.
- 5.7까지는 정수 타입에서도 길이를 명시할 수 있었지만 이는 단순히 화면에 표시할 값의 자릿수를 의미했습니다만 8.0에 deprecated 됐습니다.

15.2.5 자동 증가(AUTO_INCREMENT) 옵션 사용

- 숫자 타입의 칼럼에 자동 증가 옵션을 사용해 인조 키(Artificial Key)를 생성할 수 있습니다.
- MySQL 서버의 `auto_increment_increment(증가량)`와 `auto_increment_offset(초기 오프셋)` 시스템 설정을 이용해 AUTO_INCREMENT 칼럼의 자동 증가값이 얼마가 될지 변경할 수 있습니다.
  - 기본값은 1로 사용 됩니다.
- AI를 사용한 칼럼은 반드시 해당 테이블에서 PK나 유니크 키의 일부로 정의해야 합니다.
- PK, UK가 여러 개의 칼럼으로 구성되면 AI 속성의 칼럼값이 증가하는 패턴이 MyISAM과 InnoDB에서 각각 달라집니다.

```
-- // AUTO_INCREMENT 칼럼을 프라이머리 키의 뒤쪽에 배치해 테이블 생성 시 오류 발생
mysql> CREATE TABLE tb_autoinc_innodb (
         fd_pk1 INT NOT NULL DEFAULT '0',
         fd_pk2 INT NOT NULL AUTO_INCREMENT,
         PRIMARY KEY (fd_pk1, fd_pk2)
       ) ENGINE=INNODB;
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column ai
be defined as a key

-- // AUTO_INCREMENT 칼럼을 프라이머리 키의 뒤쪽에 배치했으나
-- // 유니크 키에서 제일 선두에 위치하여 정상적으로 테이블이 생성됨
mysql> CREATE TABLE tb_autoinc_innodb (
         fd_pk1 INT NOT NULL DEFAULT '0',
         fd_pk2 INT NOT NULL AUTO_INCREMENT,
         PRIMARY KEY (fd_pk1, fd_pk2),
         UNIQUE KEY ux_fdpk2 (fd_pk2)
       ) ENGINE=INNODB;
Query OK, 0 rows affected (0.01 sec)
```

  - MyISAM 테이블은 자동 증가 옵션이 사용된 칼럼이 PK나 유니크 키 아무 위치에서 사용될 수 있음
  - InnoDB 테이블은 PK나 유니크 인덱스에서 AI 칼럼으로 시작되는 인덱스를 생성해야 합니다.
- AI 칼럼은 테이블당 하나만 사용할 수 있습니다.

- 현재 증가값은 테이블 메타 정보에 있고, 다음 증가값은 SHOW CREATE TABLE 명령으로 조회 가능
    - 다만 해당 명령에 AI의 초기값이 같이 포함되므로 개발용에서 조회한 쿼리를 그대로 서비스용 mysql에서 사용하지 않도록 주의해야 합니다.

✅ 15.3 날짜와 시간

- 데이터 타입 소개

데이터 타입	MySQL 5.6.4 이전	MySQL 5.6.4부터
YEAR	1바이트	1바이트
DATE	3바이트	3바이트
TIME	3바이트	3바이트 + (밀리초 단위 저장 공간)
DATETIME	8바이트	5바이트 + (밀리초 단위 저장 공간)
TIMESTAMP	4바이트	4바이트 + (밀리초 단위 저장 공간)

    - mysql은 날짜만 저장하거나, 시간만 따로 저장할 수도 있습니다.
    - 주로 DATE, DATETIME이 많이 사용됩니다.
    - 밀리초 단위는 2자리당 1바이트의 공간이 더 필요합니다.
      (`DATETIME(6) == 8바이트(5바이트 + 3바이트)`)

밀리초 단위 자릿수	저장 공간
없음	0바이트
1, 2	1바이트
3, 4	2바이트
5, 6	3바이트

    - 밀리초 단위의 데이터 저장을 위해 DATETIME이나 TIME, TIMESTAMP 타입 뒤에 괄호와 함께 숫자를 표기하면 됩니다.
        - NOW()도 밀리초의 자릿수를 명시할 수 있음 (기본은 `NOW(0)`)

- 데이터 타입의 타임존
  - 칼럼 자체에 타임존 정보가 저장되지 않음
  - `DATETIME`, `DATE` 타입은 현재 DBMS 커넥션의 타임존과 관계없이 클라이언트로부터 입력된 값을 그대로 저장하고 조회할 때도 변환없이 그대로 출력함
  - 하지만 `TIMESTAMP`의 경우 항상 `UTC 타임존`으로 저장됨 → 타임존이 달라져도 값이 자동으로 보정됩니다.

- JDBC에선?
  - `DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306?serverTimezone=Asia/Seoul"`
    - 위 설정은 mysql 서버의 타임존을 지정합니다.
  - JDBC는 TIMESTAMP, DATETIME 관계없이 날짜 및 시간 정보를 getTimestam()를 통해 가져오기 때문에 둘 다 타임존 변환이 됩니다.
  - 하지만 다른 방식으로 가져왔을때는 타임존 변환이 안될 수 있습니다.
  - 저장또한 동일한 규칙이 적용됩니다.

- ORM과 같은 기술을 사용한다면
  - 내부적으로 어떤 JDBC API를 통해 DATETIME, TIMESTAMP 칼럼의 값을 가져오는지 타임존이 자동으로 변환하나는지 여부를 실제 테스트해볼 것을 권장합니다.

- 이미 데이터를 가지고 있는 mysql 서버의 타임존을 변경해야 한다면
  - 타임존 설정 및 칼럼이 가지고 있는 값도 CONVER_TZ() 같은 함수를 이용해 변환해야 합니다.
  - **TIMESTAMP의 경우**
    - 서버의 타임존에 의존하지 않고 UTC로 저장되므로 별도 변환이 필요하지 않음
  - MySQL 서버의 기본 타임존을 확인 및 변경하기 (DATETIME, TIMESTAMP)
    - `SET time_zone ...` 을 통해 서버의 기본 타임존을 변환할 수 있습니다.
    - `system_time_zone`
      - mysql 서버의 운영체제에 설정된 시스템의 시간대를 나타냅니다.

        - 운영체제 환경변수 변경, mysqld_safe 시작시 —timezone 옵션을 통해 변경할 수 있습니다.
    - `time_zone`
        - mysql 서버로 연결하는 모든 클라이언트 커넥션의 기본 커넥션을 의미합니다.
        - 아무런 설정이 없으면 `SYSTEM`으로 이는 system_time_zone을 그대로 사용함을 의미합니다.

- 두 변수는 서버 시작시 `--timezone`, `--default-time_zone` 명령 행 옵션으로 변경할 수 있음
    - 커넥션에서 시간 관련 처리를 하면 time_zone 시스템 변수의 영향만 받게 됩니다. (값이 `SYSTEM`값으로 설정되어 있따면 system_time_zone의 영향을 받음)

15.3.1 자동 업데이트

- 5.6이전까지 TIMESTAMP 타입의 칼럼은 레코드의 다른 칼럼 데이터가 변경될 때마다 시간이 자동 업데이트 되고 DATEMTIME은 그렇지 않음
- 5.6부터 INSERT, UPDATE마다 자동 업데이트를 하려면 테이블 생성시 옵션을 정의해야 함

```
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 (
        created_at_dt DATETIME DEFAULT CURRENT_TIMESTAMP,
        updated_at_dt DATETIME DEFAULT CURRENT_TIMESTAMP O
        PRIMARY KEY (id)
);
```

    - `DEFAULT CURRENT_TIMESTAMP`
        - `INSERT`될 때의 시점을 자동으로 업데이트 함
    - `ON UPDATE CURRENT_TIMESTAMP`

- `UPDATE`될 때의 시점을 자동으로 업데이트 해줌

이 기능들은 과거 버전에 비해 상당히 컴팩트해졌고 효과적입니다.

→ 5.6버전부터는 `DATETIME`, `TIMESTAMP` 타입 사이에 time_zone 시스템 변수의 타임존으로 저장하는지 UTC로 저장하는지의 차이를 제외하고 동일해졌습니다.