

カラムの型、インデックス

1日目

2日目

3日目

4日目

5日目

6日目

7日目

8日目

9日目

10日目

11日目

12日目

13日目

14日目

15日目

16日目

17日目

18日目

19日目

20日目

21日目

NUMBER

STRING

データタイプの設計が出来ていない場合

- DBパフォーマンスが悪化する可能性がある
- 正確な値をDBに格納できないかもしれない

DATETIME

文字列型

(CHAR, VARCHAR, TEXT)

CHAR, VARCHAR

文字列を扱う代表的な型として、CHAR(固定長文字列)、VARCHAR(可変長文字列)がある。作成するさいには、格納できる文字列の最大のサイズを設定する。

使用例

```
column_name_1 VARCHAR(255)
```

```
column_name_2 CHAR(10)
```

CHAR型

- DBにデータを格納する際に、あらかじめ**格納するデータの領域が固定で確保される**
- 格納する値が指定した長さよりも短くても、**ディスク領域が埋められる**
- 格納できる最大文字列は**255文字**

VARCHAR型

- DBにデータを格納する際に、データに応じて**確保されるディスク領域を調整する**
- 格納できる最大文字列は**65535文字**

文字列の長さが**固定されている場合**（電話番号、性別、マイナンバー番号 等）には、**CHAR(固定長文字列)**を用いる。**主キーとして用いる**

それ以外は、**VARCHAR(可変長文字列)**を用いる

TEXT

基本的な使い方は、VARCHAR(可変長文字列)と同じだが、VARCHARで格納できないサイズの文字列を格納する場合に用いる

使用例

column_name TEXT

- 最大で65535バイト(文字)まで格納できる
- 実際に格納されるデータのサイズが65535バイトよりも小さい場合、**ディスク使用量は少なくなる**
- **インデックスを作成することができない**(VARCHARは作成することができる)

TINYTEXT	255バイト(文字)まで格納できる
TEXT	65535バイト(文字)まで格納できる
MEDIUMTEXT	16777216バイト(文字)まで格納できる
LONGTEXT	4294967296バイト(文字)まで格納できる

VARCHARとTEXTの使い分け

【VARCHARを使う場合】

- 文字列の長さが最大でもVARCHARで設定できる限界(65535)以下であることがわかっている場合
- WHEREでの絞込み、JOINでの結び付け、ORDER BYでの並び替え、GROUP BYでの集計などに用いる場合

【TEXTを使う場合】

- VARCHARでは扱えない**大容量のデータを格納する場合**
- **絞込みなどに利用せず、最終的な結果(SELECTの対象カラム)でしか利用しない場合**

数值型

(INT, FLOAT, DOUBLE, DECIMAL)

整数型 - INT

使用例

column_name_1 INT

column_name_2 INT UNSIGNED

型名	説明
TINYINT	非常に小さい整数。 -128~127 までを扱う。符号なしの場合は、 0~255
SMALLINT	小さい整数。 -32768~32767 までを扱う。符号なしの場合は、 0~65535
MEDIUMINT	中間サイズの整数。 -8388608~8388607 までを扱う。符号なしの場合は、 0~16777215
INT	普通サイズの整数。 -2147483648~2147483647 までを扱う。符号なしの場合は、 0~4294967295
BIGINT	大きい整数。 -9223372036854775808~9223372036854775807 までを扱う。符号なしの場合は、 0~18446744073709551615

浮動小数点型(概数値) - FLOAT, DOUBLE

実数を扱うデータ型。重量、高さ、長さ、速度などの連続的な値を扱う場合に用いる
あくまで、概数値(近似値)を扱われ、数値が大きくなると正確な値にならないことに注意する

使用例

column_name_1 FLOAT

column_name_2 DOUBLE

型名	説明
FLOAT	単精度小数点と呼ばれる方法で浮動小数点を格納し、 4バイト 用いる
DOUBLE	倍精度小数点と呼ばれる方法で浮動小数点を格納し、 8バイト 用いる

固定小数点型(真数値) - DECIMAL

正確な浮動小数点の値を利用する場合に用いる。整数と浮動小数点の値の長さを指定して、指定した範囲内で正確な値を格納する

DECIMAL(M, N)

M: 整数部と小数部を合わせた桁数(最大65、デフォルト10)

N: 小数部の桁数(最大30、デフォルト0)

使用例

`column_name DECIMAL(5, 2)` — 整数部と小数部を合わせて5桁、小数部が2桁の正確な浮動小数点数

DECIMALで使用するデータ容量

9桁の10進数に対して4バイト使用して格納される。整数部と小数部のそれぞれに対して、9桁の繰返しごとに領域が割り当てられ、余り桁がある場合は、4バイトのうちの一部が必要になる。

余りの桁	バイト数
0	0
1-2	1
3-4	2
5-6	3
7-9	4

DECIMAL(18, 9)の場合: 小数点の両側に9桁ある。**整数部と小数部に4バイト必要**

DECIMAL(20, 6)の場合: 小数部に6桁、整数部に14桁ある。**小数部には6桁に3バイト必要。整数部には、9桁に4バイト、余りの5桁に3バイトが必要。**合計で、 $3+4+3 = 10$ バイト必要になる

FLOAT, DOUBLE, DECIMALの比較

	メリット	デメリット
FLOAT	使用する容量が少なくて済む(4バイト)。	精度が低い
DOUBLE	DECIMALに比較して一般的に容量は少ない(8バイト)	FLOATよりも精度は高いが、DECIMALよりも低い
DECIMAL	精度の高いデータの格納と計算ができる	容量が多くなる。けた数を指定する必要がある

- 金融系システムなどで、正確な値を格納する場合は、DECIMALを利用する
- WHEREに=などの比較を行ってデータを絞り込む対象となるカラムにはDECIMALを利用する
- 概算値でもよくデータ容量を抑えたい場合、FLOAT, DOUBLEを利用する

論理型 - BOOLEAN

真か偽の値を格納する。MySQL(ver.8)には存在しないため、TINYINTが利用される
例) アクティブか非アクティブ、管理者か一般ユーザーかなど

0の場合は偽、0以外は真を表す

使用例

column_name BOOLEAN — 実行するとTINY INT型でカラムが作成される

INSERT INTO table_name VALUES(true) -- 1が格納される

INSERT INTO table_name VALUES(false) -- 0が格納される

日時型

(DATE, TIME, DATETIME, TIMESTAMP)

DATE, TIME

DATE: 日付を扱う型。YYYY-MM-DDのフォーマット。誕生日や入社日など、**時刻のない日付のみの**データを挿入したい場合に用いる

TIME: 時刻を扱う型。HH:MM:SSのフォーマット。日時処理の起動時刻など、**日付のない時刻のみの**データを挿入したい場合に用いる。()で数字を入れると **小数点以下まで値が入れられる**

使用例

birthday **DATE** — **DATE型でカラムを作成**

start_at **TIME** — **TIME型でカラムを作成**

INSERT INTO table_name(birthday) VALUES('2019-09-01') — **2019-09-01を挿入**

INSERT INTO table_name(birthday) VALUES('20190801') — **2019-08-01を挿入**

INSERT INTO table_name(birthday) VALUES('2020/01/01') — **2020-01-01を挿入**

INSERT INTO table_name(start_at) VALUES('19:08:01') — **19:08:01を挿入**

INSERT INTO table_name(start_at) VALUES('190801') — **19:08:01を挿入**

start_at **TIME(3)** — **TIME型で小数点3桁まで残りは四捨五入する**

DATETIME, TIMESTAMP

DATETIME: 日付と時刻を扱う型。YYYY-MM-DD HH:MM:SSのフォーマット。**正確な日付と時刻**のデータを挿入したい場合に用いる。()で数字を入れると**小数点以下まで値が入れられる**

TIMESTAMP: 日付と時刻を扱う型。YYYY-MM-DD HH:MM:SSのフォーマット。DATETIMEより少しデータ使用量が少ない。DB実行の**タイムゾーン**から**UTC時刻に変換されて**データが格納され、**UTC時刻からDBのタイムゾーンに変換されて**データが取り出される。UTCで**1970-01-01 00:00:01から2038-01-19 03:14:07.9999**までしか扱えない**(2038年問題)**。**主に、レコードの挿入時刻や更新時刻を記述するカラム(create_at, update_at)に使用される。**()で数字を入れると**小数点以下まで値が入れられる**

使用例

schedule_at **DATETIME** — **DATETIME型でカラムを作成**

create_at **TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP** — **TIMESTAMP型でカラムを作成(デフォルトで現在時刻、行が更新されたときも現在時刻に自動更新される)**

インデックスについて

インデックスとは

テーブルの特定のカラムに対する索引。インデックスを用いることで、特定のレコードへのアクセスを高速で行うことができるようになる

テーブル

インデックス

インデックスを使って、特定のレコードへのアクセスを高速化する

使用例

`CREATE INDEX index_name ON table_name(column1, column2, ...)` # インデックス作成

`DROP INDEX index_name` # インデックス削除

`SHOW INDEX FROM table_name` # テーブルにあるインデックス一覧

関数インデックスの作成

関数に対するインデックス。関数の結果を用いた処理を高速化する場合に用いる

```
CREATE INDEX index_name ON table_name ((function_name(column1, column2, ...))) # 関数に対するインデックスを作成する(関数の前後に丸括弧を2つつける)
```

複数カラムに対するインデックスの作成

複数のカラムや、関数に対してインデックスを作成することもできる。

```
CREATE INDEX index_name ON table_name(column1, (function_name(column1, column2, ...)))  
# 関数に対するインデックスを作成する(関数の前後に丸括弧を2つつける)
```

ユニークインデックスの作成

カラム(カラムの組み合わせ)に対して、インデックス+ユニーク制約をつける

```
CREATE UNIQUE INDEX index_name ON table_name(column1, (function_name(column1,  
column2, ...)))
```