

# 【SQLite】入門

## はじめに

### ■特徴

- ▶ ※ SQLiteはRDBMS。
- ▶ ※ サーバにインストールするのではなく、アプリケーションに組み込んで使用。サーバ不要でかつ軽量のため、手軽に導入できる。

### ■用語

- table（ここでは表と表記してゆく）  
record（1行1行のデータのこと。ここでは録と表記してゆく）  
column（ここでは列と表記してゆく）
- view（ここでは視と表記してゆく）：既存の表（実表、基底表）を参照して作ることができ、その表の変更に伴い自らも自動更新する、表に似た存在（導出表）。
- クエリ（命令のこと）

### ■初歩的注意

- ▶ ※ 1データベースが1ファイルになる。
- ▶ ※ SQLコマンドは大文字・小文字を区別しない。
- ▶ ※ 重要なコマンドは `.` から始まる（ドットコマンド）。しかも最後に `;` が要らない。
- ▶ ※ データ型は、列に対するあらかじめの指定によらず、データが格納された瞬間に動的に決められることになっている。
- ▶ ※ **文字列は `''` で囲む。**
- ▶ ※ 命名規則はスネークケース (hoge\_hoge\_hoge)。
- ▶ ※ 以下の構文における「表」の部分は、普通に考えて「視」に置き換えても問題なさそうなものは置き換えられると思われる。（だって、表に似た存在だもの）
- ▶ ※ 以下で「計算式」という場合、それには「値を別の値に変換する行為」すべてが含まれる。
- ▶ ※ コマンドの途中の空白は改行にしてもよい。

## 環境編

### ■環境構築

# 【SQLite】入門

## はじめに

### ■特徴

- ▶ ※ SQLiteはRDBMS。
- ▶ ※ サーバにインストールするのではなく、アプリケーションに組み込んで使用。サーバ不要でかつ軽量のため、手軽に導入できる。

### ■用語

- table（ここでは表と表記してゆく）  
record（1行1行のデータのこと。ここでは録と表記してゆく）  
column（ここでは列と表記してゆく）
- view（ここでは視と表記してゆく）：既存の表（実表、基底表）を参照して作ることができ、その表の変更に伴い自らも自動更新する、表に似た存在（導出表）。
- クエリ（命令のこと）

### ■初歩的注意

- ▶ ※ 1データベースが1ファイルになる。
- ▶ ※ SQLコマンドは大文字・小文字を区別しない。
- ▶ ※ 重要なコマンドは `.` から始まる（ドットコマンド）。しかも最後に `;` が要らない。
- ▶ ※ データ型は、列に対するあらかじめの指定によらず、データが格納された瞬間に動的に決められることになっている。
- ▶ ※ **文字列は `''` で囲む。**
- ▶ ※ 命名規則はスネークケース (hoge\_hoge\_hoge)。
- ▶ ※ 以下の構文における「表」の部分は、普通に考えて「視」に置き換えても問題なさそうなものは置き換えられると思われる。（だって、表に似た存在だもの）
- ▶ ※ 以下で「計算式」という場合、それには「値を別の値に変換する行為」すべてが含まれる。
- ▶ ※ コマンドの途中の空白は改行にしてもよい。

## 環境編

### ■環境構築

▶ バージョンを確認

## 基礎編

### ■データ型の種類

▶ 整数	
▶ 浮動小数点数	
▶ 文字列	
▶ 任意のバイナリデータ	
▶ 空(から)	

### ■基本

▶ 1行のスクリプトを実行	
▶ DBに接続しコンソールを起動	
▶ コンソールを終了	
▶ コメントのしかた	
▶ 1文が長くなる時	
▶ 外部のスクリプトファを実行	

### ■出力の設定を変更

▶ 出力先を標準出力に	
▶ 出力先をファイルに	
▶ 見出しも出力する設定に	
▶ 見出しを出力しない "	
▶ カンマ区切りで "	
▶ 区切り文字で区切って "	
▶ 列ごとに左揃えで "	
▶ HTMLのTABLE形式で "	
▶ INSERT文として "	
▶ 各列ごとに行を分けて "	
▶ SQLリテラルで "	
▶ タブ区切りで "	

▶ バージョンを確認

```
$ sqlite3 -version
```

## 基礎編

### ■データ型の種類

▶ 整数	integer	5	-20
▶ 浮動小数点数	real	4.3	-1.8
▶ 文字列	text	'hello'	'世界'
▶ 任意のバイナリデータ	blob		
▶ 空(から)	null	null	

### ■基本

▶ 1行のスクリプトを実行	\$ sqlite3 <i>dbFilePath</i> "スクリプト"	※ 'スクリプト' でも可
▶ DBに接続しコンソールを起動	\$ sqlite3 <i>databaseFilePath</i> ※ <sup>1</sup>	※'拡張子は <code>.db</code> が主
▶ コンソールを終了	.exit か {Ctrl + D}	
▶ コメントのしかた	-- で行末まで、あるいは /* */ で囲めば改行可能。	
▶ 1文が長くなる時	トークン同士のあいだではいくらかでも改行可能	
▶ 外部のスクリプトファを実行	.read <i>scriptFillePath</i> か \$ sqlite3 <i>dbFilePath</i> < <i>scriptFillePath</i>	

### ■出力の設定を変更

▶ 出力先を標準出力に	.output stdout (デフォルト)
▶ 出力先をファイルに	.output <i>filePath</i>
▶ 見出しも出力する設定に	.headers on
▶ 見出しを出力しない "	.headers off (デフォルト)
▶ カンマ区切りで "	.mode csv
▶ 区切り文字で区切って "	.mode list (デフォルト)
▶ 列ごとに左揃えで "	.mode column ※これ、いいじゃん！！
▶ HTMLのTABLE形式で "	.mode html
▶ INSERT文として "	.mode insert
▶ 各列ごとに行を分けて "	.mode line
▶ SQLリテラルで "	.mode quote
▶ タブ区切りで "	.mode tabs

- ▶ TCLのlist形式で `"` `mode tcl`
- ▶ ※ コンソール起動中や、外部スクリプトファイルのなかで、これら設定変更のコマンドを実行すれば、以降コンソールを起動しているあいだはその設定が継続される。そして、コンソールを抜けた段階でデフォルトに戻される。

#### ■入力

- ▶ CSVファを表として読み込み `mode csv` `import csvPath` 表 ※要注意

#### ■出力・表の読み取り

- ▶ ヘルプを表示 `.help`
- ▶ ある表の構造を確認 `.schema` 表
- ▶ すべての表の構造を確認 `.schema` ※ 覗(ビュー)、トリガーの構造も一緒に
- ▶ すべての表の名前一覧 `.tables` ※ `"`
- ▶ 直近に挿入した録の主キー値 `last_insert_rowid()`

- ▶ すべての列を出力 `select * from 表;`
- ▶ 特定の列を出力 `select 列1, 列2, ... from 表;`
- ▶ 条件に合う録のみに `where 条件式`
- ▶ ある列で並び替え `order by 列1 desc, 列2, ...` ※ `desc` つけると降順に
- ▶ シャッフル `order by random()` ※limit節つければランダム抽出も可能に
- ▶ 最大 $n$ 件の録のみに `limit n`
- ▶  $m$ 件目以降で `"` `limit n offset m` または `limit m, n`
- ▶ 計算して出力 `select 計算式 as 好きな見出し名 from 表;`
- ▶ UNIQUEして出力 `select distinct 列 from 表;`
- ▶ グル化し集計して出力 `select 列1, 列2への集計処理 from 表 group by 列1;`
- ▶ 条件に合うグルのみに `group by 列1 having 条件式`
- ▶ 内部結合して出力 `select ··· from 表1 join 表2 on 表1.列1 = 表2.列2;`
- ▶ 左外部結合して出力 `select ··· from 表1 left join 表2 on 表1.列1 = 表2.列2;`
- ▶ 交差結合して出力 `select ··· from 表1 cross join 表2;`

- ▶ 表をスクリプトファとして保存 `output scriptFilePath` `.dump` 表
- ▶ CSVファとして保存 `mode csv` `.output csvPath` `select ···`; ※要注意

#### ■表の作成

- ▶ 表を作る `create table if not exists 表名 (列名1, 列名2, ...);`

- ▶ TCLのlist形式で `"` `mode tcl`
- ▶ ※ コンソール起動中や、外部スクリプトファイルのなかで、これら設定変更のコマンドを実行すれば、以降コンソールを起動しているあいだはその設定が継続される。そして、コンソールを抜けた段階でデフォルトに戻される。

#### ■入力

- ▶ CSVファを表として読み込み `mode csv` `import csvPath` 表 ※要注意

#### ■出力・表の読み取り

- ▶ ヘルプを表示 `.help`
- ▶ ある表の構造を確認 `.schema` 表
- ▶ すべての表の構造を確認 `.schema` ※ 覗(ビュー)、トリガーの構造も一緒に
- ▶ すべての表の名前一覧 `.tables` ※ `"`
- ▶ 直近に挿入した録の主キー値 `last_insert_rowid()`

- ▶ すべての列を出力 `select * from 表;`
- ▶ 特定の列を出力 `select 列1, 列2, ... from 表;`
- ▶ 条件に合う録のみに `where 条件式`
- ▶ ある列で並び替え `order by 列1 desc, 列2, ...` ※ `desc` つけると降順に
- ▶ シャッフル `order by random()` ※limit節つければランダム抽出も可能に
- ▶ 最大 $n$ 件の録のみに `limit n`
- ▶  $m$ 件目以降で `"` `limit n offset m` または `limit m, n`
- ▶ 計算して出力 `select 計算式 as 好きな見出し名 from 表;`
- ▶ UNIQUEして出力 `select distinct 列 from 表;`
- ▶ グル化し集計して出力 `select 列1, 列2への集計処理 from 表 group by 列1;`
- ▶ 条件に合うグルのみに `group by 列1 having 条件式`
- ▶ 内部結合して出力 `select ··· from 表1 join 表2 on 表1.列1 = 表2.列2;`
- ▶ 左外部結合して出力 `select ··· from 表1 left join 表2 on 表1.列1 = 表2.列2;`
- ▶ 交差結合して出力 `select ··· from 表1 cross join 表2;`

- ▶ 表をスクリプトファとして保存 `output scriptFilePath` `.dump` 表
- ▶ CSVファとして保存 `mode csv` `.output csvPath` `select ···`; ※要注意

#### ■表の作成

- ▶ 表を作る `create table if not exists 表名 (列名1, 列名2, ...);`

▶ ある列に型を指定	
▶ ある列で空値を禁止	
▶ ある列で値の重複を禁止する	
▶ ある列に初期値を設定	
▶ ある列に値の制限かける	
▶ ある列を主キー (自動) に	
▶ ある列を作成日時 (自動) に	
▶ ☆ ある列を更新日時 (自動) に	
▶ 出力を新視として作成	

#### ■表の更新

▶ 表を削除	
▶ 表名を変更	
▶ 列を追加	
▶ ※ 列名の変更はできない！	
▶ ※ 列の削除はできない！	

▶ 録を追加	
▶ 録の値を変える	
▶ 録に計算を加える	
▶ 全録を削除	
▶ 条件に合う録を削除	
▶ 別の表から録を読み込み	

#### ■条件式・条件分岐

▶ 比較演算子	
▶ ※ 文字列と比較するときは <b>大小文字を区別する</b> ことに注意！	
▶ 論理演算子	
▶ $m$ 以上 $n$ 以下	
▶ $a, b, c$ どれかに一致	
▶ ワイルドカード	

▶ ある列に型を指定	列名 型
▶ ある列で空値を禁止	列名 型 not null
▶ ある列で値の重複を禁止する	列名 型 unique
▶ ある列に初期値を設定	列名 型 default 初期値
▶ ある列に値の制限かける	列名 型 check (列名 >= 0 and 列名 <= 200) など
▶ ある列を主キー (自動) に	・列名 integer primary key ※削除済みの録と重複しうる ・列名 integer primary key autoincrement ※ " しない
▶ ある列を作成日時 (自動) に	列名 datetime default (datetime('now', 'localtime'))
▶ ☆ ある列を更新日時 (自動) に	
▶ 出力を新視として作成	create vie 視名 as 表の形で出力させるクエリ;

#### ■表の更新

▶ 表を削除	drop table if exists 表;
▶ 表名を変更	alter table 表 rename to 表名;
▶ 列を追加	alter table 表 add column 列名 ...;
▶ ※ 列名の変更はできない！	
▶ ※ 列の削除はできない！	

▶ 録を追加	insert into 表 (列1, 列2, ...) values (値1, 値2, ...);
▶ 録の値を変える	update 表 set <u>列 = 別の値</u> , ... where 条件式;
▶ 録に計算を加える	update 表 set <u>列 = 計算式</u> , ... where 条件式;
▶ 全録を削除	delete from 表;
▶ 条件に合う録を削除	delete from 表 where 条件式;
▶ 別の表から録を読み込み	insert into 表1 (列11, 列12, ...) select 列21, 列22, ... from 表2;

#### ■条件式・条件分岐

▶ 比較演算子	> < >= <= = == <> != ※ <b>列 &gt; 値</b> のように使う
▶ ※ 文字列と比較するときは <b>大小文字を区別する</b> ことに注意！	
▶ 論理演算子	and or not
▶ $m$ 以上 $n$ 以下	列 between $m$ and $n$
▶ $a, b, c$ どれかに一致	列 in ( $a, b, c$ )
▶ ワイルドカード	like節なら % : 0文字以上の任意の文字列 _ : 任意の1文字 glob節なら * : " ? : " [abc] や [a-c] というのも可能

▶ パターンマッチング	
▶ ワイルドカードをエスケープ	
▶ null かどうか	

▶ SWITCH関数的な	
--------------	--

■数値	
▶ 算術演算子	
▶ 集計処理	
▶ 乱数( $c \mathbb{R}$ )の生成	

■文字列	
▶ 文字列を表現	
▶ 特殊な文字を表現	
▶ 改行を表現	
▶ 文字列の結合	
▶ MID	
▶ 文字数	
▶ 大文字に	

■日時	
▶ ※ SQLiteに日時を扱うデータ型はないので、実際には日時を表す文字列や数値を管理しておいて、それに <code>datetime</code> のような関数を組み合わせて扱う。	
▶ 現在日時	
▶ 日時・日付・時刻を表現	

## 応用編

■トランザクション	
▶ トランザクションを設定	
▶ トラに入る前の状態を回復	

▶ パターンマッチング	大小文字を区別しないなら 列 like パターン文字列 " を区別する 列 glob パターン文字列
▶ ワイルドカードをエスケープ	like節なら 'str' escape 'char' で任意の文字をエスケープ文字に glob節なら [] で囲む
▶ null かどうか	列 is null 列 is not null

▶ SWITCH関数的な	case <u>when 条件式1 then 値1</u> ... else 値n end
--------------	---

■数値	
▶ 算術演算子	+ - * / %
▶ 集計処理	count(列) avg(列) max(列) min(列)
▶ 乱数( $c \mathbb{R}$ )の生成	random()

■文字列	
▶ 文字列を表現	'文字列'
▶ 特殊な文字を表現	" (シングルクォートに)
▶ 改行を表現	そのまま ' ' のなかで改行
▶ 文字列の結合	
▶ MID	substr(列, start※ <sup>1</sup> , len※ <sup>2</sup> ) ※ <sup>1</sup> 負列 ※ <sup>2</sup> 略すと最後まで
▶ 文字数	length(列)
▶ 大文字に	upper(列)

■日時	
▶ ※ SQLiteに日時を扱うデータ型はないので、実際には日時を表す文字列や数値を管理しておいて、それに <code>datetime</code> のような関数を組み合わせて扱う。	
▶ 現在日時	datetime('now', 'localtime') ※ <code>'localtime'</code> 略すとUTCに
▶ 日時・日付・時刻を表現	datetime(timeValue, modifier1, ...)・date(")・time(")

## 応用編

■トランザクション	
▶ トランザクションを設定	begin transaction; 一連処理 commit;
▶ トラに入る前の状態を回復	rollback;

■トリガー

- ▶ トリガーを設定
- ▶ 既存のトリガーの構造を出力
- ▶ トリガーを削除

■インデックス

- ▶ ※ インデックスを設定すると、検索が早くなる反面、挿入や更新をするときには処理が重くなってしまうので、状況を見ながらバランスよく設定していくとよい。
- ▶ ある列に索引を設定
- ▶ ある列にユニーク索引を設定
- ▶ ※ 表作成時に `unique` 制約や `integer primary key` をしている列は初めからユニーク索引を設定しているのと同じなので、改めてユニーク索引を設定する必要はない。
- ▶ 索引を削除
- ▶ ある表の索引の一覧を出力
- ▶ ある表の索引の構造を出力

■トリガー

- ▶ トリガーを設定
- ▶ 既存のトリガーの構造を出力
- ▶ トリガーを削除

■インデックス

- ▶ ※ インデックスを設定すると、検索が早くなる反面、挿入や更新をするときには処理が重くなってしまうので、状況を見ながらバランスよく設定していくとよい。
- ▶ ある列に索引を設定
- ▶ ある列にユニーク索引を設定
- ▶ ※ 表作成時に `unique` 制約や `integer primary key` をしている列は初めからユニーク索引を設定しているのと同じなので、改めてユニーク索引を設定する必要はない。
- ▶ 索引を削除
- ▶ ある表の索引の一覧を出力
- ▶ ある表の索引の構造を出力