わかりみ

SQL

カウプラン機関極東支部

Kauplan Press

■免責

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、®、© などのマークは省略しています。

はじめに

この本は、初心者が SQL を学ぶための本です。

と同時に、「なんでこういう説明をしてくれなかったんだ！」と過去の自分が感じた恨みの集大成でもあります。

♣ この本の目的

この本の目的は、初めて SQL をやってみる人が実用的な集計を SQL で行えるようになることです。「実用的な集計」とは、たとえば次のようなことです。

- テストの平均点を、クラス別科目別に集計する。
- 赤点（平均点より 20 点低い点数）をとった生徒を検索する。
- 今月の 1 日からの売上を日別に集計し、累計と前日比もつけて表示する。

♣ 対象読者

この本は次のような人を対象としています。

- SQL が初めての人
- SQL を勉強したけど挫折した人
- SQL を新人に教えることになった先輩

特に初心者がつまづきやすい点を厚く解説しているので、過去に SQL で挫折した人こそこの本がお勧めです。「最初からこういうふうに教えてくれたらよかったのに！」と言ってもらえたら、それがこの本にとって最高の褒め言葉です。

♣ この本の内容

この本は大きく 4 つに分かれています。

- 第 I 部「導入編」では、データベースの必要性や基本操作を説明します。必要性が理解できないと学習意欲が湧かないと思います。「導入編」ではデータベースの特長を Excel と比べながら解説するので、必要性を理解しましょう。
- 第 II 部「初級編」では、SQL に関する一通りの基礎知識を説明します。

初心者の人は、なんとかこの「初級編」までを勉強しましょう。
- 第 III 部「中級編」では、実用的な SQL を書くのに必要な知識を説明します。仕事で SQL を書く人は、この「中級編」までを勉強しましょう。
- 第 IV 部「修行編」では、実システムの SQL の問題を解いてみます。これがスラスラ解けるようになったら、転職して給料アップを狙いましょう。

詳しい内容は目次を見てください。また上級編はありませんが、かわりに「SQL 高速化 in PostgreSQL」[*1]をお勧めしておきます。

♣ この本で扱わないこと

次のような内容はこの本の範囲を超えるため、扱いません。

- テーブル設計や正規化
- 実行計画やパフォーマンスチューニング
- ビューやトリガーやストアドプロシージャ

特にテーブル設計と正規化を扱ってないことに注意してください。この本はあくまで SQL を身につけるための本なので、それ以外は別の本を参照するようお願いします。

♣ サポートサイト

正誤表などはサポートサイトで公開しています。

- `https://kauplan.org/books/wakarimisql/`

♣ 謝辞

この本の一部をゴリラ氏 (@gorilla0513) にレビューしていただきました。ありがとうございます。

なおこの本に何か不備や問題点があったとしても、その責任はあくまで著者にあり、氏にはありません。氏に迷惑のかかるような真似はしないようお願いします。

[*1] `https://kauplan.org/books/sqltuning/`

目次

第Ⅰ部

導入編

第1章
データベース

SQL はデータベースを検索・操作するための専用言語です。なので SQL を学ぶなら、データベースについてもある程度学ぶ必要があります。とはいえ、データベースがなぜ必要なのかを納得しないと、学習意欲は湧かないと思います。

この章では、データベースとは何か、どんな特長があるのか、なぜ必要なのかなどを説明します。データベースの特長を Excel と比較しながら説明するので、データベースの必要性を納得できることでしょう。

【この章の内容】

1.1 データベースと SQL

♣ データベースとは

データベース (Database) とは、さまざまなデータの検索や登録や変更や削除ができるようになっているシステムのことです。たとえば：

- 新幹線の座席予約システムでは、どの列車のどの座席が誰によって予約されているか、すべて記録されています。これが座席予約システムでのデータベースです。
- Amazon のような EC サイトには商品のデータベースがあり、商品コードや価格やメーカーなどが記録されています。またユーザの注文を記録した注文データベースがあり、ユーザが過去に購入した商品をもとに別の商品をお勧めしてきます。
- 小さな飲食店や美容院では「予約台帳」と呼ばれるノートがあり、日時や顧客名や連絡先のような予約情報が書き込まれます。デジタル化されていないものの、これも立派なデータベースです。
- 子供の保護者に回覧される緊急連絡網は、データベースというよりは名簿リストと呼んだほうがいいでしょう。

予約台帳のような紙のままのデータベースもありますが、この本ではデジタル化されたデータベースを扱います。

♣ リレーショナルデータベース

コンピュータで扱えるデータベースにもいくつか種類がありますが、現実のシステムでいちばん使われているのは「リレーショナルデータベース」というものです。

リレーショナルデータベース (Relational Database)[*1] とは、データを種類ごとにテーブル（表）の形で保持しておき、必要に応じてテーブルを連結したり加工できるデータベースです（詳しくは後述します）。

[*1] 略して「RDB」と呼ばれることもあります。

♣ リレーショナルデータベース製品

リレーショナルデータベースというのはあくまでデータベースの種類であり、それを使うためのソフトウェア製品[*2]が必要です。オープンソース[*3]の製品では「MySQL」や「PostgreSQL」がよく使われています[*4]。

そんな説明じゃよく分からんからゲームに例えて！

仕方ないなあ。ちょっとだけよ？

分類

- データベースにはいくつか種類がある。「リレーショナルデータベース」はそのひとつ。
- ゲームにはいろんなジャンルがある。「ロールプレイングゲーム」(RPG) はそのひとつ。

製品

- 「リレーショナルデータベース」は種類の名前。使うためには「PostgreSQL」や「MySQL」のようなソフトウェア製品が必要。
- 「ロールプレイングゲーム」はジャンルの名前。遊ぶためには「ドラゴンクエスト」や「ファイナルファンタジー」のようなゲームソフトが必要。

つまり、PostgreSQL や MySQL はドラクエや FF と同じ！

♣ SQL とは

「SQL」とは、リレーショナルデータベースからデータを検索したり加工するための専用言語です。慣れれば非常に強力ですが、慣れるまでがすごく大変で、毎年たくさんの脱落者を生み出しています。

[*2] リレーショナルデータベース製品は「RDBMS」(Relational Database Management System) と呼ぶことが多いです。

[*3] 広義ではソースコードが公開されているソフトウェアのことです。狭義ではオープンソースライセンスのもとで公開されているソフトウェアのことです。

[*4] 商用製品では「Oracle Database」(Oracle 社) と「SQL Server」(Microsoft 社) のシェアが高いです。

次が SQL の例です。これは赤点をとった生徒とそのテスト結果を検索し、一覧表示する SQL です。

▼赤点（平均点よりも 20 点低い）の生徒とそのテスト結果を検索

```
select s.*, t.*              ← 生徒の情報とテスト結果を表示
from test_scores t
  join (
      select x.subject
           , avg(x.score) as avg_score  ← 教科ごとの平均点
      from test_scores x
      group by x.subject               ← 教科ごとにグループ化
  ) agg on t.subject = agg.subject
  join students s on t.student_id = s.id
where t.score < agg.avg_score - 20  ← 平均点よりも20点低ければ
order by s.id, t.subject;
```

このような SQL がスラスラ書けると、SQL はとても強力な武器になります。

そしてこの本は、まさにこのような SQL がスラスラ書けるようになることを目標としています。

SQL の勉強に PostgreSQL を使う理由

この本では「PostgreSQL」を使って SQL を説明し、MySQL については必要に応じて都度言及します。

PostgreSQL を使う理由は、SQL の動作が素直で理解しやすいからです。MySQL は独特な仕様が多い[*5]ので、初心者が SQL を学習するには少し不向きです。

市場シェアは MySQL のほうがずっと高いので、最初から MySQL でやりたい！ という気持ちは分からなくもないです。しかし MySQL の独特な仕様のせいで初心者が SQL を理解できなくなる可能性を考えれば、やはり初心者は動作が素直な PostgreSQL で SQL を勉強するのがお勧めです。

[*5] グループ化のキーではない列名を select 句に指定できる、references 句がエラーなしで無視される、相関ではないサブクエリが相関サブクエリのように動作する、など。なお Oracle 社に買収されてからは少しずつ素直な仕様に変わりつつあります。

1.2　データベースの特長

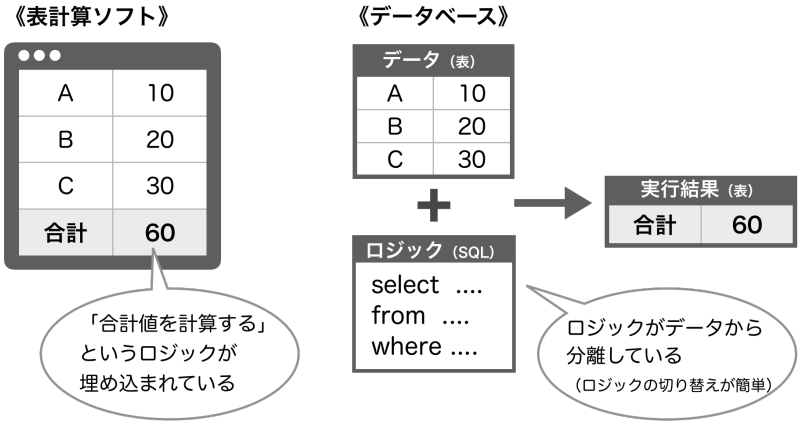
実はリレーショナルデータベースでも、Excel と同じように「テーブル（表）」を扱います。そのため両者はよく似ていると思われがちですが、もちろん違う点はたくさんあります。ここでは Excel のような表計算ソフトと比較しながら、リレーショナルデータベースの特長を紹介します。

♣ データとロジックが分離している

Excel では、データとロジックが同じシートに載っています（図 1.1 左）。たとえば、生徒のテストの点数を変更したらクラスの合計点が自動的に更新されるのも、1 つのシートにデータとロジックが載っているからです。

これに対しリレーショナルデータベースでは、データとロジックが分離しています（図 1.1 右）。たとえば、データ（テーブル）には生徒のテストの点数だけが載っており、クラスの合計点が必要になったらそのときに集計ロジック（SQL）を実行します。そのため、**同じテーブルでもロジックを切り替えれば違う集計ができます**（平均点、個人別総合得点、赤点の生徒の一覧など）。

このように、リレーショナルデータベースではデータとロジックが分離しているため、様々な集計や検索が切り替えやすくなっています。



▲図 1.1　リレーショナルデータベースではデータとロジックが分離

♣ データを柔軟に組み合わせられる

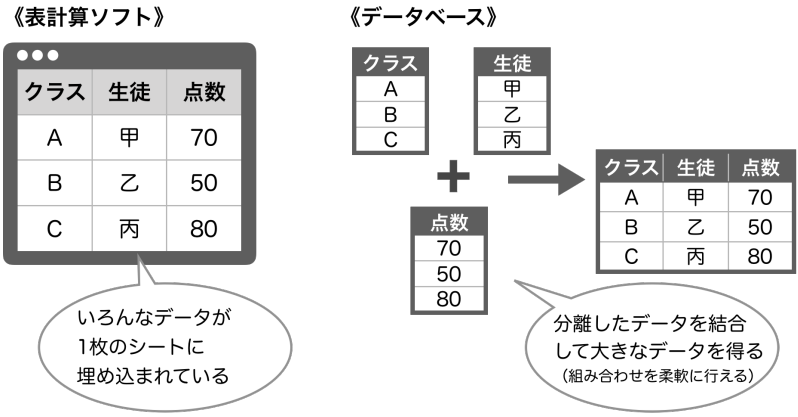
Excel では、1 つのシートやテーブルにいろんな種類のデータを入れることがあります。たとえば、1 つのシートにクラスのデータと生徒のデータとテストの得点データが入っているのが普通です（図 1.2 左）。

これに対しリレーショナルデータベースでは、種類ごとに細かくテーブルを分けます。そして必要に応じてテーブルを結合することで、望みの結果を得ます（図 1.2 右）。

たとえばクラスのデータと生徒のデータとテストの得点データは、それぞれ別のテーブルとして格納されます。そして必要に応じてこれらを結合することで、クラスと生徒とテストの得点の結果を得ます。

こうすることで、たとえばクラスと生徒だけを結合する、生徒とテストの得点だけを結合する、あるいは別のデータのテーブルと結合する、というようにデータの組み合わせを柔軟に変えられます。

一般的には、**結合されたデータを分離するよりも、分離されたデータを結合するほうがずっと簡単**です。リレーショナルデータベースでは、扱いやすい形でデータを保持しておき、必要に応じて組み合わせるので、柔軟な操作や検索ができます。



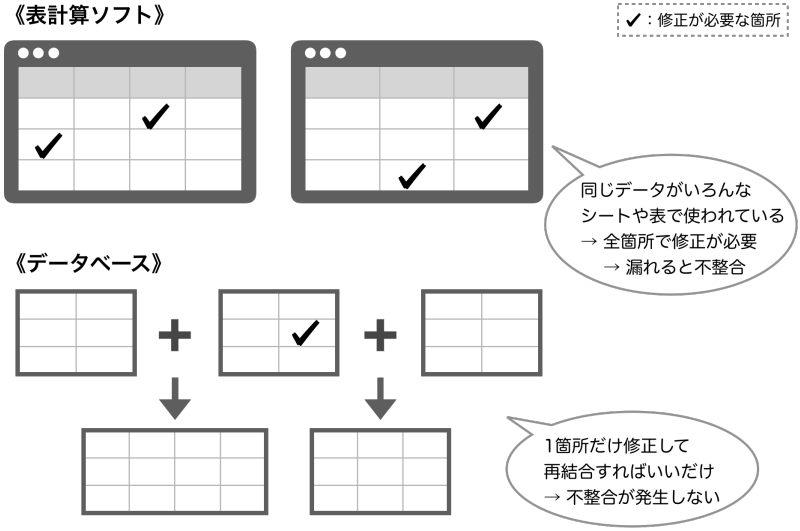
▲図 1.2　種類ごとにデータを分離しておき、必要に応じて結合

♣ データの整合性を保つ

Excel では、あるデータが複数の箇所に散らばっていることがよくあります。そのため、たとえば社員が結婚して名前が変わった場合、その社員が登場している箇所をすべて修正しなければいけません。もし修正し忘れた箇所があれば、その社員は結婚前と結婚後の両方の名前がシステムで使われてしまうため、データの矛盾が発生します（図 1.3 左）。

これに対しリレーショナルデータベースでは、データは 1 箇所にのみ保持され、必要に応じてテーブルを連結します。そのため、たとえば社員が結婚して名前が変わったとしても **1 箇所だけ修正して再連結すればよく、いろんな箇所を探しまわって修正する必要はありません**（図 1.3 右）。

このように、リレーショナルデータベースではデータの整合性を保ち、不整合を起こさないための機能を備えています[*6]。



▲図 1.3　修正箇所が 1 箇所で済むので、データの不整合が発生しない

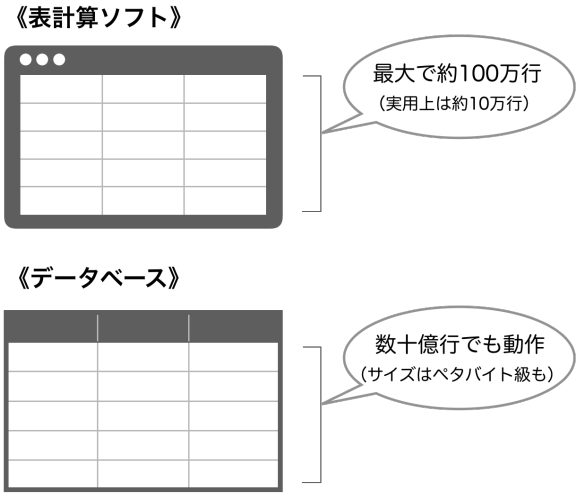
[*6] ただし機能を正しく使う必要があります。正しく使わないと不整合は発生します。

♣ 大量のデータを扱える

Excel は大量のデータを扱えません。1 シートあたりの行数は、最大でせいぜい 100 万行です[*7]。また最大で約 100 万行といってもそれは仕様上の上限であって、実際には 10 万行も使えばパソコンが固まってしまいます（図 1.4 左）。

これに対しリレーショナルデータベース製品は、大量のデータを扱うことに長けています[*8]。たとえば**テーブルの行数が数十億単位であっても問題なく動作します**（図 1.4 右）。さらにコンピュータの台数を増やして性能を上げる[*9]ことも可能です。

このように、大量のデータを扱うなら Excel ではなくデータベース製品を使うべきです。



▲図 1.4　Excel では無理でも、データベースなら大量のデータを扱える

*7 MS Office 365 の場合。なお Office 2003 までは最大で 65,536 行でした。とてもデータベース製品には敵いません。

*8 これはリレーショナルデータベースだからそうなったというわけではなく、各製品が改善を積み重ねた結果、大量のデータを扱えるようになりました。

*9 複数のコンピュータをあたかも 1 台かのように扱う技術を「クラスタリング」といいます。データベースのサーバをクラスタリングにすると、性能が上がるだけでなく、1 台が故障しても別のコンピュータが動けるので故障にも強くなります。

♣ 同時変更による競合を考慮している

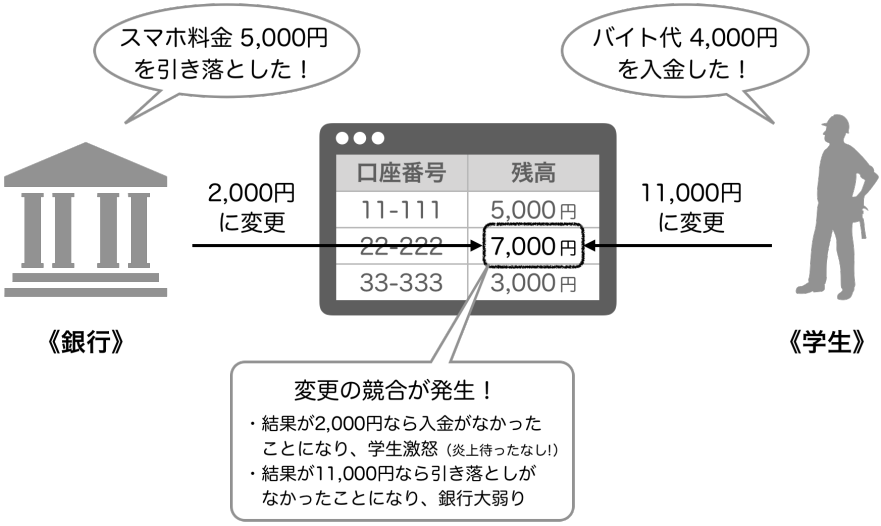
Excel では複数の人が同時に 1 つのファイルを編集できますが、同じ箇所を同時に変更したときの競合は考慮してくれません。

複数人が同時に変更を行うと、たとえば図 1.5 のような問題が起こります。

- 学生はバイト代 4,000 円を入金した。現在の残高が 7,000 円なので、口座残高を 11,000 円に変更。
- 銀行は学生のスマホ料金 5,000 円を引き落とした。現在の残高が 7,000 円なので、口座残高を 2,000 円に変更。

これらの変更が同時に行われた場合、口座残高が 11,000 円になるのか 2,000 円になるのか、分かりません。そのうえ、どちらの結果になっても問題なのです。

これに対しデータベース製品では、**変更の競合を検出したり、整合性が取れなくなったら変更をキャンセルできます**（トランザクション機能）。複数人が同じデータを同時に変更するなら、必ずデータベース製品を使ってください。



▲図 1.5　口座残高を同時に更新すると、問題が起こる

1.3 データベースが苦手なこと

Excel では簡単にできるのに、リレーショナルデータベースではできないこともあります。ここではリレーショナルデータベースが苦手なことを紹介します。

♣ グラフ表示

Excel には強力なグラフ作成機能があります。棒グラフ、円グラフ、折れ線グラフなど、多彩できれいなグラフが作成できます。

リレーショナルデータベースには、そのような機能はありません。データベースではデータの検索や集計だけを行い、それをグラフにするのは別のアプリケーションで行います。

♣ フォーム作成

Excel にはフォーム作成機能があります。たとえば氏名と性別と生年月日を入力するようなフォームを作成し、その Excel ファイルを相手にメールで送信し、入力してもらった Excel ファイルを送り返してもらう、という作業を経験したことのある人は多いでしょう。

リレーショナルデータベースには、そのような機能はありません。フォーム入力を受けつける Web アプリケーションを作って、そこで入力されたデータをデータベースに保存するのが一般的です。

♣ 帳票機能

Excel を使うと、注文書や請求書などを作成できます。特に日本人は罫線が大好きなので、罫線を使った帳票を作れる Excel は重宝されています。

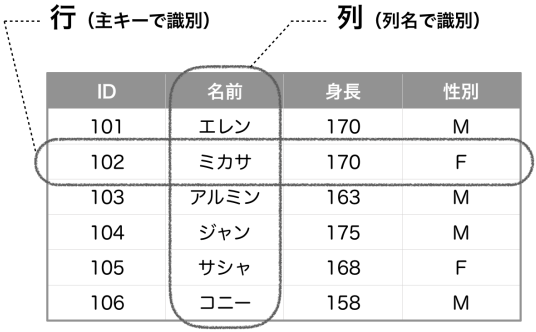
リレーショナルデータベースには、そのような機能はありません。グラフ機能やフォーム機能もそうですが、見た目を整えるような機能はリレーショナルデータベースにはありません。

1.4 テーブルについて

リレーショナルデータベースでは、データをテーブル（表）の形で保持しています。ここではテーブルについて詳しく見てみます。

♣ テーブルの構造

テーブル (Table) は、行 (Row) と列 (Column) から構成されます。



▲図 1.6　テーブルは行と列から構成される

リレーショナルデータベースでは、列を識別するには列名を使います。たとえば上の図だと、「ID」列と「名前」列と「身長」列と「性別」列があるので、この列名を指定すれば列を識別できます。

行には列名のような名前はなく、かわりに「主キー」[*10]と呼ばれる、行を特定するための列とその値で識別します。たとえば図 1.6 では「ID」列が主キーであり、ID=102 を指定すればミカサの行、ID=105 ならサシャの行となります。

このように、行と列とでは識別する方法が違います。**行を識別するのは（主キーの）「値」だけど列を識別するのは値でなく「名前」（列名）である**、というのがポイントです。覚えておきましょう。

また行と列は対等のように思うかもしれませんが、リレーショナルデータベースでは「テーブルは行の集まり」あるいは「行が集まったものがテーブル」です。決して「列が集まったのがテーブル」ではありませんので注意してください。

[*10] 主キーについては第 10.3 節「主キーと主キー制約」(p.188) で説明します。

♣ テーブルの種類

テーブルは、縦方向に伸びる/伸びない、横方向に伸びる/伸びない、の組み合わせがあります（2 × 2 = 4 種類）。

- 「縦方向に伸びる」とは、行が増えることです。
- 「横方向に伸びる」とは、列が増えることです。

図 1.7 は、横に伸びるタイプのテーブルの例です。横軸が試合番号なので、試合数が増えるたびに列が増えていく（つまり横に伸びる）ことが分かります。

	第1戦	第2戦	第3戦	第4戦	第5戦	第6戦	第7戦	第8戦
修		1			2	1		
遊真	6	3	2	1	3	3	1	
千佳							1	
ヒュース						2		

▲図 1.7　試合ごとの個人別獲得ポイント数（横軸が値）

これに対し図 1.8 は、横に伸びないタイプのテーブルの例です。横軸が能力パラメータ名なので、列が増えていかない（横に伸びない）ことが分かるでしょう。

	トリオン	攻撃	防御・援護	機動	技術	射程	指揮	特殊戦術
修	2	3	4	4	5	4	6	4
遊真	7	9	8	10	8	2	5	4
千佳	38	2	5	3	6	8	1	1
ヒュース	18	12	24	13	9	4	6	10

▲図 1.8　個人別の能力パラメータ値（横軸が名前）

この 2 つを見ると、横方向に伸びるタイプと伸びないタイプの違いは、**横軸が値か名前か**の違いであることが分かります。

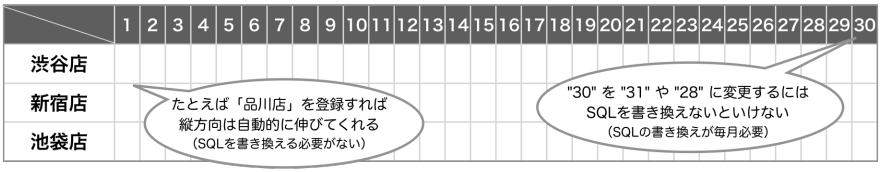
- 図 1.7 は横軸が試合番号なので試合ごとに増える値です。だから横方向に伸びます（列が増えます）。
- 図 1.8 は横軸が能力パラメータ名なので名前です。だから（能力パラメー

タの種類を増やさない限り[*11]）横方向には伸びません。

これは縦方向でも同じです。

表計算ソフトではどのタイプのテーブルでも扱えます。しかし、リレーショナルデータベースでは横に伸びない方のテーブルだけを扱います。なぜなら、リレーショナルデータベースでは列を識別するのに列名を使い、値を使わないからです（前の項で説明した通りです）[*12]。そのため、横方向に伸びる（つまり列を増やす）にはテーブルや SQL を書き換えなければならないので苦手です。それに対して縦方向に伸びる（つまり行を増やす）のは、テーブルや SQL を書き換える必要がなくデータに応じて自動的に伸びるので、得意です。

たとえば横軸に今月の日付（1〜31）を使った集計をすると、月によって列の数が 28 から 31 まで変化するので、そのたびに SQL を書き換えないといけません（図 1.9）。こういうのは SQL はあまり得意ではないです[*13]。



▲図 1.9　横軸に今月の日付（1〜31）を使った集計

テーブルは二次元なのか？

縦にも横にも伸びるテーブルは、「2 次元のテーブル」と言えます。これに対し、リレーショナルデータベースが扱うのは横に伸びないので、2 次元のテーブルではありません。なので、たとえば「リレーショナルデータベースは 2 次元のテーブルの集まりです」という表現は間違いです。

しかし多くの人はそんな厳密な意味は考えておらず、単に「平面的な」という意味で「2 次元」と言っているだけです。厳密な意味を必要として

[*11] 試合数が増えることと能力パラメータの種類が増えることは、同じことではありません。前者はデータの追加であるのに対し、後者はデータの仕様変更です。

[*12] 興味のある人は『理論から学ぶデータベース実践入門』（奥野幹也著、技術評論社、2015）を読んでみましょう。

[*13] こういう場合は別のプログラムで SQL を生成するのがいいでしょう。

いない場面で厳密な意味を持ち出してマウントするのは、悪い大人です。TPO をわきまえましょう。

♣ テーブルには行の順番がない

リレーショナルデータベースのテーブル（表）では、**行の順番は保持されません**。たとえばある行 A と B と C があったとして、さっき見たときは A → B → C の順番に並んでいたのに、今見たら B → A → C の順になっていた、ということはごく普通です。なぜかというと、リレーショナルデータベースでは**テーブルは「行の集合」であって「行のリスト」ではないから**[*14]です。

「リスト」と「集合」の違い

「リスト」と「集合」の違いは 2 点あります。

- リストは順番を保持するけど、集合は保持しません。たとえばリストでは「A → B → C」と「B → C → A」と「C → A → B」はどれも違うリストですが（なぜなら順番が違うから）、集合では「A, B, C」と「B, C, A」と「C, A, B」はどれも同じです。
- リストは要素の重複を許すけど、集合は許しません。たとえばリストでは「A → B → A」のように重複があっても OK ですが、集合では許されないので、重複のある「A, B, A」は重複のない「A, B」と同じと見なされます。

リレーショナルデータベースでは、テーブルは行の集合だと見なされます。なのでテーブルは行の順番を保持しません[*15]。そのかわりに、指定した順序で行を並び替える機能が発達しています。つまり行の順番は保持しないから、かわりに検索のたびに並び替えなさい、という方針です。

[*14] 実際には、主キーのないテーブルでは重複した行が存在できてしまいます。なのでテーブルには必ず主キーを設定しましょう。主キーについては第 10.3 節「主キーと主キー制約」(p.188) で説明します。

[*15] なお、列には順番（テーブル作成時に指定した列名の順番）があります。

第2章
PostgreSQLのインストール

この章では、SQL の勉強用に PostgreSQL をインストールする方法を説明します（本番環境で使うようなインストール方法ではないです）。

インストールと環境構築は、多くの初心者がつまずく箇所です。なるべく丁寧に説明するよう心がけましたが、分からない点があればサポートサイト（`https://kauplan.org/books/wakarimisql/`）までお問い合わせください。

なお対象環境は macOS と Docker です。Windows の場合は申し訳ありませんが Docker をお使いください。

【この章の内容】

2.1 macOS で使う

PostgreSQL を macOS で使う方法はいくつかあり、PostgreSQL のサイトで説明されています。

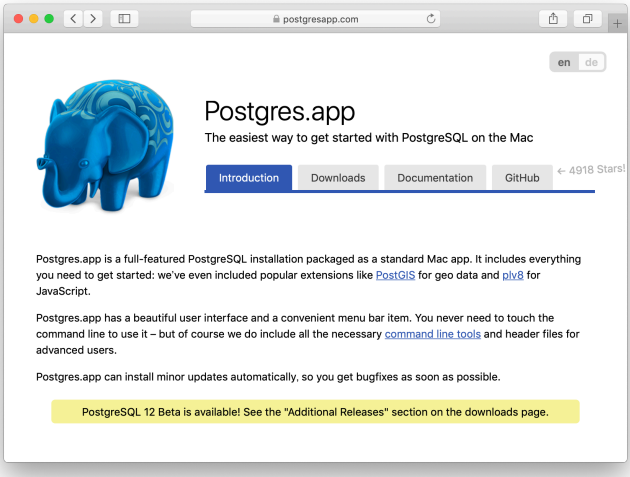
- `https://www.postgresql.org/download/macosx/`

その中から、この本では「Postgres.app」を使う方法を説明します。

「Postgres.app」は macOS 用のアプリケーションであり、通常のアプリケーションと同様の手順でインストールできるので、初心者でもつまずきません。本番環境で運用する用途には向きませんが、SQL を試してみる分には最適です。

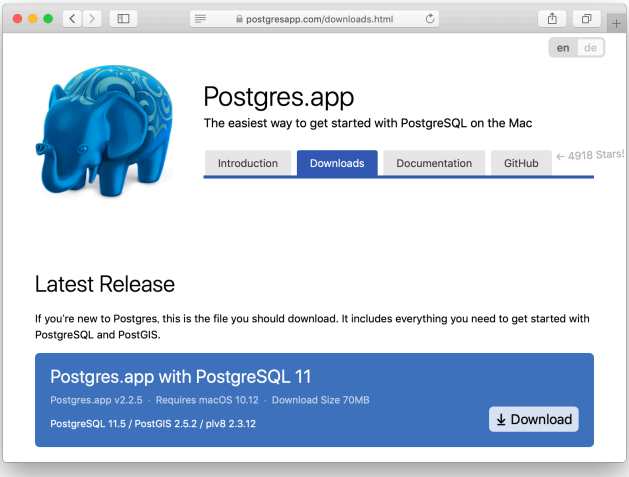
♣ Postgres.app をインストール

まず、Postgres.app のサイト（`https://postgresapp.com/`）にアクセスします（図 2.1）。



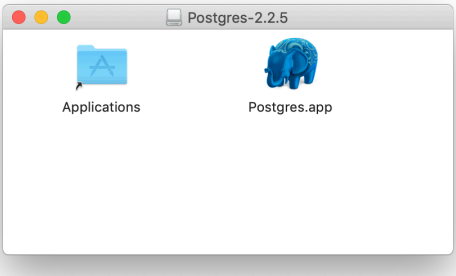
▲図 2.1　Postgres.app のサイト

ページ上部の「Downloads」タブをクリックし、「Latest Release」のところにある「Download」ボタンをクリックします（図 2.2）。



▲図 2.2　Postgres.app のダウンロードページ

Finder を開き、ダウンロードした「Postgres-2.X.X-XX.dmg」をダブルクリックします。しばらくするとフォルダが開くので、「Postgres.app」を「Applications」にドラッグアンドドロップします（図 2.3）。

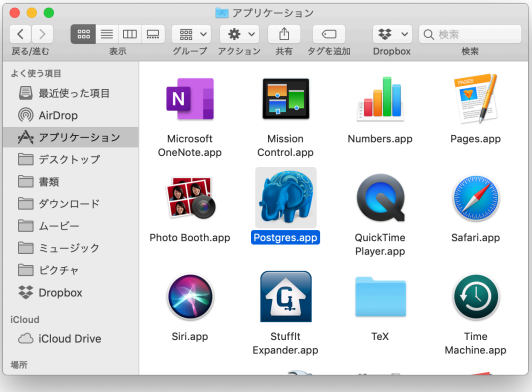


▲図 2.3　「Postgres.app」を「Applications」にドラッグアンドドロップ

これで Postgres.app のインストールが完了です。

♣ Postgres.app の起動と停止

アプリケーションフォルダから「Postgres.app」のアイコンをダブルクリックします（図 2.4）。

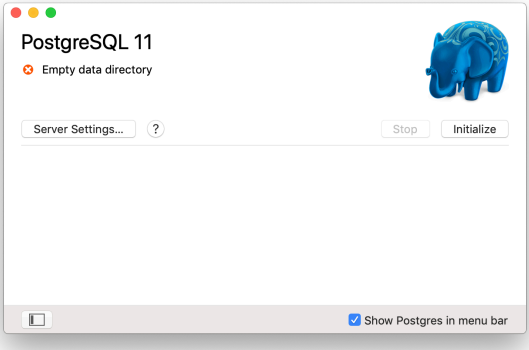


▲図 2.4　アプリケーションフォルダから「Postgres.app」をダブルクリックして起動

最初の起動時には「“Postgres.app”はインターネットからダウンロードされたアプリケーションです。開いてもよろしいですか?」というダイアログが出るので、「開く」をクリックします（図 2.5）。

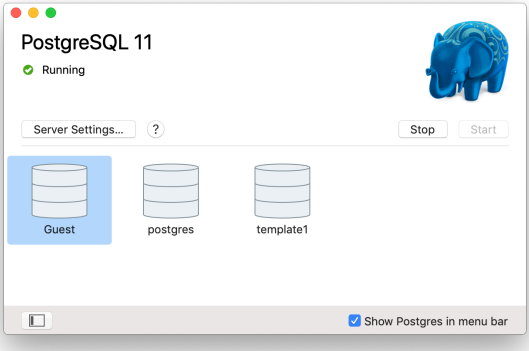
▲図 2.5　最初の起動時には確認用のダイアログが出る

Postgres.app の画面が表示されます（図 2.6）。



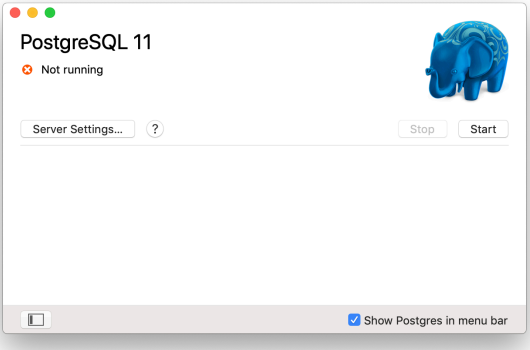
▲図 2.6　Postgres.app の画面

画面右の「initialize」ボタンを押します。データベースが作成されて、しばらくするとそれが画面に表示されます（図 2.7）。



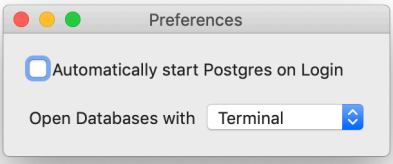
▲図 2.7　データベースが作成される

ここで、「stop」ボタンを押してみましょう。PostgreSQL が停止します（図 2.8）。「start」ボタンを押すと、もう一度 PostgreSQL が起動します。



▲図 2.8　「stop」ボタンで停止、「start」ボタンで起動

また Postgres.app のメニューから「Postgres」>「Preferences...」を選びます。「Automatically start Postgres on Login」にチェックがついているので、外しておきましょう（図 2.9）。通常はログイン時には起動する必要はなく、必要になったら起動すれば十分です。

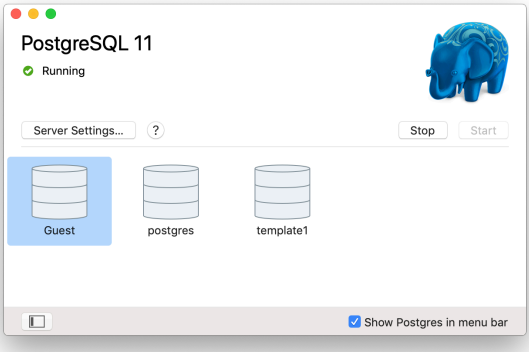


▲図 2.9　Postgres.app の環境設定

なお PostgreSQL サーバが使っているポート番号は通常は 5432 ですが、もし別のポート番号に変更したい場合は「Server settings」ボタンを押すと変更できます。

♣ Postgres.app で SQL を実行

Postgres.app の画面には、ログインしているユーザ名と同じ名前のデータベース[*1]が作成されているはずです（図 2.10）。表示されていないときは、「start」ボタンを押して起動してください。



▲図 2.10　ユーザ名と同じ名前のデータベースがあるはず

この、ユーザ名と同じ名前のデータベースをダブルクリックします。「Terminal.app を起動するけどいいか？」という趣旨の確認ダイアログが出るので、OK ボタンを押します（図 2.11）。

▲図 2.11　警告ダイアログ

[*1] 他に「postgres」と「template1」というデータベースが表示されていますが、これは PostgreSQL のシステム用なので、変更や削除をしないでください。

Terminal.app が起動され[*2]、その中で psql コマンド[*3]が実行されるので、SQL が入力可能な状態になります（図 2.12）。

▲図 2.12　Terminal.app 経由で psql コマンドが起動され、SQL が入力可能に

ためしに「`select current_date;`」と入力して Enter キーを押してください（最後の「`;`」を忘れないように）。今日の日付が表示されるはずです（図 2.13）。

▲図 2.13　「select current_date;」を実行して今日の日付を表示

これで SQL を実行できるようになりました。

[*2] Terminal.app はデフォルトのフォントサイズが小さいので、Command キーを押しながら「＋」キーを 4〜5 回押して、フォントサイズを大きくするといいでしょう。

[*3] コマンドラインから SQL をインタラクティブに実行するためのコマンド。Ruby の irb コマンドや Node.js の node コマンドや Python の ipython コマンドに相当。

2.2 Docker で使う

「Docker」とは、Linux を動かす仮想環境のひとつです[*4]。Docker を使うと、Linux で動くシステムやアプリケーションを Windows や Mac でも動かせます。このおかげで、Docker さえあれば Windows でも Mac でも PostgreSQL を動かせます。

PostgreSQL では、公式の Docker イメージが提供されています。

- `https://hub.docker.com/_/postgres`

これを使って、Docker 上で PostgreSQL を使ってみましょう。

なお Docker はすでにインストール済みとします。まだの場合、公式サイトのドキュメント[*5]を参考にしてインストールしてください。

♣ コンテナを起動する

PostgreSQL 公式の Docker イメージをダウンロードし、コンテナを起動します。

▼ Docker イメージをダウンロードしてコンテナを起動するまで

```
### PostgreSQL ver11の公式Dockerイメージを取得
$ docker pull postgres:11

### コンテナをバックグラウンドで起動
$ docker run --name posgre11 -d -e POSTGRES_PASSWORD=pass1 postgres:11
3c9a1f6dac7854d48dddd253a7fcc98ce04372ea60ea28017ab9c69ebcaddb7c

### コンテナが起動したことを確認
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
             STATUS              PORTS               NAMES
3c9a1f6dac78        postgres:11         "docker-entrypoint.s…"   5 seco
nds ago       Up 5 seconds        5432/tcp            posgre11
```

[*4] Docker を初心者に説明するのはとても難しいので、ここではこの程度の理解で結構です

[*5] Windows なら `https://docs.docker.com/docker-for-windows/`、macOS なら `https://docs.docker.com/docker-for-mac/` です。

♣ データベースを作成

PostgreSQL のデータベースを作成します。そのためにはデータベースユーザが必要なので、先にデータベースユーザ「user1」を作成してから、そのあとにデータベース「testdb1」を作成します。

▼データベースユーザとデータベースの作成

```
### コンテナにアクセス
$ docker exec -it posgre11 bash

### 各種コマンドが存在することを確認
root@3c9a1f6dac78:/# which psql createuser createdb
/usr/bin/psql
/usr/bin/createuser
/usr/bin/createdb

### データベースユーザ「user1」を作成
root@3c9a1f6dac78:/# createuser -U postgres user1  ← 作成

### 作成されたことを確認
root@3c9a1f6dac78:/# psql -U postgres -c '\du'
                                  List of roles
 Role name |                         Attributes                         | Member
 of
-----------+------------------------------------------------------------+-------
----
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
 user1     |                                                            | {}
   ↑ 「user1」があることを確認

### データベース「testdb1」を作成
bash$ createdb -U postgres -O user1 -E UTF8 --locale=C -T template0 testdb1

### 作成されたことを確認
root@3c9a1f6dac78:/# psql -U postgres -l
                                List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
```

```
-----------+----------+----------+------------+------------+-------------------
---
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres
  +
           |          |          |            |            | postgres=CTc/postgr
es
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres
  +
           |          |          |            |            | postgres=CTc/postgr
es
 testdb1   | user1    | UTF8     | C          | C          |
(4 rows)
   ↑ 「testdb1」があることを確認
```

これでデーターベースユーザ「user1」とデータベース「testdb1」が作成されました。

アクセスできるか、確かめてみましょう。

▼データベースユーザ「user1」で、データベース「testdb1」にアクセス

```
### 作成したデータベースユーザで、作成したデータベースにアクセス
root@3c9a1f6dac78:/# psql -U user1 testdb1
psql (11.5 (Debian 11.5-1.pgdg90+1))
Type "help" for help.

testdb1=> select current_date;      ← 簡単なSQLが実行できることを確認
 current_date
--------------
 2019-09-22
(1 row)

testdb1=> \q                        ← 「\q」でpsqlコマンドを抜ける
root@3c9a1f6dac78:/#
```

これで、SQL が実行できるようになりました。

「PostgreSQL」と「データベース」の違い

初心者の人が理解するのはちょっと難しいかもしれませんが、PostgreSQL や MySQL はデータベースを動かすソフトウェア[*6]であって、データベースそのものとは違います。なので PostgreSQL や MySQL をインストールしたあとに、データベースを別途作成する必要があります。

またマシンに PostgreSQL や MySQL をインストールすれば、その 1 台のマシンで複数のデータベースを作成できます。たとえば「開発用のデータベース」と「テスト用のデータベース」、あるいは「旧システムのデータベース」と「新システムのデータベース」などのように、1 台に複数のデータベースを作成することはよくあります。

長い SQL を実行する方法

先ほどもやったように Terminal.app の画面に SQL を直接入力する方法だと、長い SQL を入力するには不向きです。

長い SQL を実行したいときは、まずエディタで SQL を書いて、それをコピー＆ペーストで Terminal.app の画面に貼りつけるほうが簡単です[*7]。これだと SQL が長くても入力がしやすいです。

また SQL を別ファイルに書いて、それを読み込んで実行するのもいいでしょう。たとえばホームディレクトリに「`test1.sql`」というファイルを作成し、その中に「`select current_date;`」と書いて保存します[*8]。そして Terminal.app の画面[*9]で「`\i test1.sql`」（または「`\i ~/test1.sql`」）と入力すれば、ファイルが読み込まれて中の SQL が実行されます。

[*6] 「RDBMS」(Relational Database Management System) や「データベースエンジン」と言います。

[*7] 最後の「`;`」を忘れないように。

[*8] この場合は最後の「`;`」がなくてもいいですが、あっても問題はないのでつけておきましょう。

[*9] 正確には psql コマンドのプロンプト。

第3章
基本操作

データベースが作成できたので、いよいよテーブルを作成し、基本的なデータ操作をしてみましょう。

本格的なデータ操作は次の章から説明するとして、ここでは「テーブルを作成」「行を挿入」「行を検索」「行を変更」「行を削除」「テーブルを削除」の6つの操作を説明します。

【この章の内容】

3.1 テーブルを作成

テーブルを作成するには、create table 文を使います。

create table 文は長くなることが多いので、ここでは次のような SQL ファイルを用意しましょう。

▼ファイル「create-testtable.sql」（「←」とそれ以降は含めなくてよい）

```
create table testtable1 (       ← テーブルを作成
  id     integer   primary key
, name   text      not null unique
, age    integer
);
```

そしてこれをエディタでコピーし、ターミナル画面の psql プロンプトにペーストしてください。

または、ターミナル画面の psql プロンプトで「`\i SQL ファイル名`」[*1]としてください。下の 2 つはプロンプト[*2]が違いますが、どちらもターミナル画面です。

▼Postgres.app のターミナル画面を使っている場合

```
Username=# \i create-testtable.sql
```

▼Docker のターミナル画面を使っている場合

```
testdb1=> \i create-testtable.sql
```

または、psql コマンドの引数に SQL ファイル名を指定しても実行できます。このとき、データベースユーザ名「user1」とデータベース名「testdb1」を指定してください。

[*1] Postgres.app の場合はターミナル画面がホームフォルダで起動するので、ホームフォルダに作った SQL ファイルはパス名を指定しなくても認識されます。それ以外の場所に作った場合はファイルのパス名も指定してください。

[*2] プロンプト (Prompt) とは、入力を促す文字列のこと。ここでは「`Username=#`」や「`testdb1=>`」のこと。

▼SQL ファイルを実行

```
bash$ psql -U user1 testdb1 < create-testtable.sql
```

テーブルが作成されたか、確認してみましょう。psql コマンドのプロンプトで以下を実行し、出力結果にテーブル名「testtable1」が含まれていたら成功です。

▼テーブルが作成されたことを確認

```
testdb=> \dt        ← テーブルの一覧を表示する
             List of relations
 Schema |        Name        | Type  | Owner
--------+--------------------+-------+-------
 public | testtable1         | table | user1
(1 row)
```

なお、このあとの説明では SQL とその実行結果だけを紹介します。たとえば上の create table 文は次のように紹介するので、コピペでターミナル画面に貼りつけてもいいし、「\i ファイル名」でもいいし、好きな方法で実行してください。

▼テーブルを作成する（「←」とそれ以降は含めなくてよい）

```
create table testtable1 (       ← テーブルを作成
  id     integer   primary key
, name   text      not null unique
, age    integer
);
```

▼実行結果

```
CREATE TABLE
```

3.2 テーブルに新しい行を挿入

すでに説明したように、テーブルは行の集合です。なので、作成したテーブルに行を挿入してみましょう。そのためには insert 文を使います。

▼ insert 文を使って、テーブルに新しい行を挿入

```
insert into testtable1(id, name, age)
values (101, 'Alice', 20);

insert into testtable1(id, name, age)
values (102, 'Bob', 25);

insert into testtable1(id, name, age)
values (103, 'Cathy', 22);
```

▼ 実行結果

```
INSERT 0 1    ← 「1行挿入された」ことを表す

INSERT 0 1    ← 「1行挿入された」ことを表す

INSERT 0 1    ← 「1行挿入された」ことを表す
```

ただ、この書き方は「`insert into testtable1(id, name, age) values`」という箇所を何度も書く必要があります。これに対し、次のように 1 つの insert 文で複数の行を挿入する書き方もできます。これをバルクインサート (Bulk insert) といいます。

▼ 1 つの insert 文で複数の行を挿入（バルクインサート）

```
insert into testtable1(id, name, age)
values (101, 'Alice', 20)
     , (102, 'Bob'  , 25)
     , (103, 'Cathy', 22);
```

▼ 実行結果

```
INSERT 0 3    ← 「3行挿入された」ことを表す
```

3.3 テーブルの行を検索

テーブルに挿入した行を検索してみます。そのためには select 文を使います。

▼ psql コマンドを使って、テーブルの行をすべて検索

```
select *          ← 「*」はすべての列を表示
from testtable1;  ← テーブルの行をすべて表示
```

▼実行結果

```
 id  | name  | age
-----+-------+-----
 101 | Alice |  20
 102 | Bob   |  25
 103 | Cathy |  22
(3 rows)
```

これでテーブルに含まれる行をすべて表示できました。

ある特定の行だけを検索するには、select 文に検索条件をつけます。また列名を指定すれば、その列だけが表示されます。

▼名前が「Bob」の行だけを検索、また名前と年齢の列だけを表示

```
select name, age      ← name列とage列だけを表示
from testtable1
where name = 'Bob';   ← nameが「Bob」の行だけを表示
```

▼実行結果

```
 id  | name | age
-----+------+-----
 102 | Bob  |  25
(1 row)
```

これで、条件に合った行だけが表示されました。

select 文については、第 4 章 (p.47) で詳しく説明します。

3.4 テーブルの行を更新

テーブルの行を更新するには、update 文を使います。

update 文を使うまえに、テーブル「testtable1」の内容を確認してみましょう。

▼テーブル「testtable1」の内容（つまりすべての行）

```
select * from testtable1;
```

▼実行結果（年齢がそれぞれ 20 歳、25 歳、22 歳であることを確認）

```
 id  | name  | age
-----+-------+-----
 101 | Alice |  20
 102 | Bob   |  25
 103 | Cathy |  22
(3 rows)
```

内容を確認したら、行を更新してみましょう。次の update 文は、年齢を 1 つ増やします。

▼update 文を使って、年齢を 1 つ増やす

```
update testtable1
set age = age + 1;
```

▼実行結果

```
UPDATE 3    ← 「3行更新された」という意味
```

更新後の内容を表示すると、すべての行で年齢が増えています！

▼テーブル「testtable1」の内容を表示

```
select * from testtable1;
```

▼実行結果（年齢がそれぞれ 20 → 21、25 → 26、22 → 23 に増えている！）

```
 id  | name  | age
-----+-------+-----
 101 | Alice |  21
 102 | Bob   |  26
 103 | Cathy |  23
(3 rows)
```

ある特定の行だけを更新するには、update 文に条件式を追加します。

▼名前が「Bob」の行だけ、年齢を「27」に更新

```
update testtable1
set age = 27
where name = 'Bob';
```

▼実行結果

```
UPDATE 1      ← 「1行だけ更新された」という意味
```

テーブルの中身を確認してみましょう。

▼テーブル「testtable1」の内容を表示

```
select * from testtable1;
```

▼実行結果（Bob だけ年齢が 26 → 27 に更新された！）

```
 id  | name  | age
-----+-------+-----
 101 | Alice |  21
 103 | Cathy |  23
 102 | Bob   |  27    ← 26だったのが27に変更された
(3 rows)
```

たしかに Bob の年齢だけが更新され、ほかの 2 人は更新されていません。

これで、行を更新できるようになりました。

行の並び順は保持されない

上の実行結果で、Bob が 3 番目に表示されていることに気づいたと思います。前は 2 番目に表示されていたのに、どうしてでしょうか？

その理由については、すでに「テーブルには行の順番がない」(p.16) で説明しました。実は PostgreSQL や MySQL では、行の並び順は保持されません。最初は「Alice → Bob → Cathy」の順番で並んでいたとしても、どれかの行を変更したらその順番は変更されます。これは Excel とは大きく違う点なので注意してください。

PostgreSQL や MySQL では行の並び順は保持されないかわりに、select 文で検索するときに行の並び方を指定できます。たとえば次のようにすると、行が名前順で表示されます。

▼名前順で表示するよう指定

```
select * from testtable1
order by name;      ← name列の値で並び替える
```

▼実行結果

```
 id  | name  | age
-----+-------+-----
 101 | Alice |  21
 102 | Bob   |  27
 103 | Cathy |  23
(3 rows)
```

行の並び順を指定する方法は、第 4.5 節「order by 句：整列する」(p.55)で詳しく説明します。

なおここで説明したように、update 文では条件を指定すれば特定の行だけを更新できますが、条件を指定しなければすべての行が更新されます。大事なことなので 2 回言いますが、**update 文では条件を指定しなければすべての行が更新されます**。気をつけてください。

また 1 つの update 文で（つまり一度に）複数の**列**を更新するには、次のようにします。

▼一度に複数の列を更新する

```
update testtable1
set age = 27         ← age列を変更
  , name = 'Bill'    ← name列を変更
where name = 'Bob';
```

ただし、一度に複数の**行**を更新するのは、update 文はとても苦手です。たとえば「Alice と Cathy の年齢を 1 つ増やす」というのは得意ですが、「Alice の年齢を 21 歳に、Cathy の年齢を 23 歳に更新する」というのは一度にはできません[*3]。このような場合は一度に更新しようとせず、素直に 2 回の update 文で更新してください。

[*3] より正確に言うと、update 文では 1 つの更新対象列に対して複数の値や式を指定できません。また case 式や JSON 型を使えばできなくはないのですが、そのような無理やりな方法はこの本が対象としている初心者の人は気にしなくていいです。

3.5 テーブルから行を削除

delete 文を使うと、テーブルから行を削除できます。
たとえば次のようにすると、名前が「Bob」である行だけを削除します。

▼テーブル「testtable1」から、名前が「Bob」である行だけを削除

```
delete from testtable1
where name = 'Bob';      ← 削除対象を表す条件
```

▼実行結果

```
DELETE 1        ← 「1行削除した」という意味
```

テーブルの内容を表示してみると、「Bob」の行がなくなって、3 行だったのが 2 行になっていることが分かります。

▼テーブル「testtable1」の内容を表示

```
select * from testtable1;
```

▼実行結果（「Bob」の行がなくなっている！）

```
 id  | name  | age
-----+-------+-----
 101 | Alice |  21
 103 | Cathy |  23
(2 rows)
```

また delete 文に条件式をつけずに実行すると、すべての行が削除されます。大事なことなので繰り返しますが、**delete 文に条件式をつけないとすべての行が削除されます**。

▼すべての行を削除する

```
delete from testtable1;
```

▼実行結果

```
DELETE 2        ← 「2行削除された」という意味
```

これで、テーブルから行を削除できるようになりました。

3.6 テーブルを削除

最後に、テーブル自体を削除してみましょう。

「テーブルの行をすべて削除」と「テーブル自体を削除」は違う

テーブルの行をすべて削除することと、テーブル自体を削除することは、似てはいるけど違います。

- テーブルの行をすべて削除すると、行はなくなりますが、テーブルは残ったままです。例えるなら、本棚の中を空にしても本棚はなくならないのと同じです。
- テーブル自体を削除すると、テーブル自体がなくなるのはもちろんのこと、テーブルの行もすべて削除されます。例えるなら、本が入っている本棚ごと捨てるようなものです（もちろん本も捨てられます）。

テーブルを削除するには、drop table 文を使います。

▼テーブル「testtable1」を削除する

```
$ drop table testtable1;
```

▼実行結果

```
DROP TABLE        ← 「テーブルが削除された」ことを表す
```

ここでテーブル「testtable1」を検索しようとしても、テーブルがないのでエラーになります。

▼テーブル「testtable1」の内容を表示

```
select * from testtable1;
```

▼実行結果

```
ERROR:  relation "testtable1" does not exist
LINE 1: select * from testtable1;
                      ^
```

これで、テーブルを削除できるようになりました。

3.7 練習問題

解答は 2 ページ後に載っています。

♣ 問題 1：テーブルを作成（難易度：★☆☆）

ID と名前と年齢の列を持つ、次のようなテーブルを作成してください[*4]。

```
create table testtable1 (
  id        integer       primary key
, name      text          not null
, age       integer
);
```

♣ 問題 2：行を挿入（難易度：★☆☆）

テーブル「testtable1」に、次のような行を挿入してください。

- ID=101、名前=Alpha、年齢=20
- ID=102、名前=Blavo、年齢=25
- ID=103、名前=Charlie、年齢=23

♣ 問題 3：行を検索（難易度：★☆☆）

テーブル「testtable1」の行をすべて検索してください。
それができたら、名前が「Blavo」の行だけを検索してください。
それもできたら、年齢が 22 歳以上の行だけを検索してください。

♣ 問題 4：行を更新（難易度：★☆☆）

Blavo の年齢だけを 26 歳に変更してください。
それができたら、全員の年齢を 1 だけ増やしてください。

[*4] なおこの本では、create table 文を覚えなくてもコピー＆ペーストで実行できれば、ひとまずはよしとします。

♣ 問題 5：行を削除（難易度：★☆☆）

Blavo の行だけを削除してください。
それができたら、テーブル「testtable1」からすべての行を削除してください。

♣ 問題 6：テーブルを削除（難易度：★☆☆）

テーブル「testtable1」を削除してください。

♣ 解答 1：テーブルを作成（難易度：★☆☆）

次のような SQL ファイルを作成し、「psql」コマンドで実行します。

▼ファイル「create-testtable.sql」

```
create table testtable1 (
  id        integer       primary key
, name      text          not null
, age       integer
);
```

▼実行例

```
bash$ psql -U user1 testdb1 < create-testtable.sql
```

♣ 解答 2：行を挿入（難易度：★☆☆）

新しい行を挿入するには、insert 文を使います。

▼新しい行を挿入する

```
insert into testtable1(id, name, age)
values (101, 'Alpha', 20)
     , (102, 'Blavo', 25)
     , (103, 'Charlie', 23);
```

♣ 解答 3：行を検索（難易度：★☆☆）

テーブルの行を検索するには、select 文を使います。

▼テーブル「testtable1」の行をすべて検索

```
select *
from testtable1;
```

▼実行結果

```
 id  |  name   | age
-----+---------+-----
 101 | Alpha   |  20
```

```
 102 | Blavo   |  25
 103 | Charlie |  23
(3 rows)
```

特定の行を検索するには、select 文に条件式をつけます。

▼名前が「Blavo」の行だけを検索

```
select *
from testtable1
where name = 'Blavo';
```

▼実行結果

```
 id  | name  | age
-----+-------+-----
 102 | Blavo |  25
(1 row)
```

▼年齢が 22 歳以上の行だけを検索

```
select *
from testtable1
where age >= 22;
```

▼実行結果

```
 id  |  name   | age
-----+---------+-----
 102 | Blavo   |  25
 103 | Charlie |  23
(2 rows)
```

♣ 解答 4：行を更新（難易度：★☆☆）

特定の行だけを更新するには、update 文に条件式をつけます。

▼Blavo の年齢だけを 26 歳に変更

```
update testtable1
set age = 26
where name = 'Blavo';
```

▼実行結果

```
UPDATE 1
```

update 文に条件式をつけない場合は、すべての行が更新されます。

▼全員の年齢を 1 だけ増やす

```
update testtable1
set age = age + 1;
```

▼実行結果

```
UPDATE 3
```

▼更新後のテーブルの中身（すべての行）を検索

```
select * from testtable1;
```

▼実行結果（年齢が更新されていることを確認）

```
 id  |  name   | age
-----+---------+-----
 101 | Alpha   |  21
 103 | Charlie |  24
 102 | Blavo   |  27
(3 rows)
```

♣ 解答 5：行を削除（難易度：★☆☆）

行を削除するには、delete 文を使います。またある特定の行だけを削除するには、delete 文に条件式をつけます。

▼Blavo の行だけを削除

```
delete from testtable1
where name = 'Blavo';
```

▼実行結果

```
DELETE 1
```

▼削除後のテーブルの中身（すべての行）を検索

```
select * from testtable1;
```

▼実行結果（Blavo の行がないことを確認）

```
 id  |  name   | age
-----+---------+-----
 101 | Alpha   |  21
 103 | Charlie |  24
(2 rows)
```

delete 文に条件式をつけずに実行すると、すべての行が削除されます。

▼すべての行を削除

```
delete from testtable1;
```

▼実行結果

```
DELETE 2
```

▼削除後のテーブルの中身（すべての行）を検索

```
select * from testtable1;
```

▼実行結果

```
 id | name | age
----+------+-----
(0 rows)
```

♣ 解答 6：テーブルを削除（難易度：★☆☆）

テーブルを削除するには、drop table 文を使います。

▼テーブル「testtable1」を削除

```
drop table testtable1;
```

▼実行結果

```
DROP TABLE
```

第 II 部

初級編

第 III 部

中級編

第 IV 部

修行編