データ管理

- データ管理
  - データを挿入する
  - データを選択
    - 選択の条件付け
    - LIKEとワイルドカード
    - その他のキーワード
    - データを整理する
    - LIMITとOFFSET
  - データを削除
  - データの更新
  - さまざまな種類のリクエスト
    - 明確な
    - 機能
    - エイリアス
    - グループ化
    - 持つ

これでテーブルは作成されましたが、そこにデータを挿入したいので、最初のユーザーを作成しましょう。

データを挿入する

```
ユーザーに挿入（ユーザー名、メールアドレス、パスワード）
    値('Maurice'、 'mo@gmail.com'、 'sock');
```

- 「INSERT INTO」は、次のテーブルにデータを挿入することを示します。括弧内の「(...)」は、追加する列をカンマで区切って指定します。ここでは「id」、「active」、「createdAt」の値が自動生成されているので、これらの値を入力する必要はなく、他の値のみを追加します。

- 「VALUES」は、後に続くものが行に割り当てる値であることを示します。「(...)」には、列に割り当てるさまざまな値
- が含まれており、前の括弧と同じ順序で配置する必要があります。

データを選択

データを表示したい場合は、以下を使用できます。

```
ユーザーから*を選択します。
```

- 「SELECT」は列を選択することを示し、「*」はすべての列を選択することを示し、
- 「FROM」はどのテーブルからこれらの列を選択するかを SQL に指示します。
-

「id」、「active」、「createdAt」フィールドが正しく入力されていることに注意してください。

しかし、ログイン時に必要なパスワードなど、常にすべての情報を取得する必要はありませんが、ユーザーのリストを表示する場合は必要ありません。

```
users からusernameを選択します。
```

「*」の代わりに、取得したい列の名前を指定できます。

```
users からユーザー名、メール、createdAt を選択します。
```

複数の列が必要な場合は、カンマで区切ることができます。

ユーザーを追加してみましょう。 「addUsers.sql」の内容をコピーします。addUsers.sqlファイルへのリンク

各括弧をコンマで区切ることで、複数の行を一度に追加できることに注意してください。

選択の条件付け

前のコマンドのいずれかを再利用すると、データベースに登録されているすべてのユーザーが取得されることがわかります。

10 個程度であれば問題ありませんが、数千個ある場合、ユーザーに関する情報が必要なときに、そのすべてを取得することはできなくなります。

```
SELECT username FROM users WHERE id = 8;
```

- 「WHERE」は特定の条件を指定することを示します。ここでの「id = 8」は、ID
- 列が 8 であるすべてのユーザーを取得することを意味します。

LIKEとワイルドカード

```
SELECT username FROM users WHERE email LIKE "%gmail.com";
```

- 「LIKE」は、正確な値を期待する「=」とは異なり、
  それから私たちは置きます。
- 「%」記号は「不明な文字数」を示します。 「%gmail.com」ここでは、
- 「gmail.com」で終わる要素を検索します。

```
SELECT username FROM users WHERE username LIKE "%il%";
```

したがって、ここではユーザー名に「he」が含まれるすべてのユーザーが対象となります。

しかし、場合によっては 1 つの仕様だけでは不十分で、複数の仕様を組み合わせたい場合があります。

```
SELECT username FROM users WHERE email LIKE "%gmail.com" AND username LIKE "%il%";
```

- 「AND」を使用すると、複数の条件を連続して作成できます。ここでは、 「gmail.com」にメールアドレスがあり、ユーザー名に「he/him」を含むすべてのユーザーが対象となります。

3 つの結果が得られたので、次のテストを実行してみましょう。

```
SELECT username FROM users WHERE email LIKE "%gmail.com" AND username LIKE "%_il%";
```

「%」と「_」の違いは何でしょうか？ 「%」は「0」から「無限大」までの任意の文字を意味し、 「_」は任意の文字の「1回」を意味します。つまり、ここでは「il」を含むが「il」で始まっていない名前を求めていることになります。

```
SELECT username FROM users WHERE email LIKE "%gmail.com" OR email LIKE "%laposte.net";
```

- 「OR」では2つの条件に応じて選択することができます。

```
SELECT username FROM users WHERE NOT email LIKE "%gmail.com";
```

- 「NOT」を使用すると条件を反転できます。ここでは、Gmail アドレスを持っていない人を検索します。

その他のキーワード

```
SELECT username FROM users WHERE username <> "Mauritius";
```

同様に、 「SQL」のバージョンに応じて「<>」または「!=」を使用すると、指定された値とは異なるものを選択できます。

ユーザースライスを選択するには、次のようにします。

```
SELECT username FROM users WHERE id >= 5 AND id <= 8;
```

または以下を使用します:

```
SELECT username FROM users WHERE id BETWEEN 5 AND 8;
```

- 「BETWEEN」は、 2 つの値の間にあるエントリを検索することを示します。

```
SELECT username FROM users WHERE username IN ("Maurice", "Hypolite", "Florestan");
```

- 「IN」を使用すると、指定された値のリスト内で列に一致するものを検索できます。このIN内に別のSQLクエリを作成することもできます。例えば、次のようになります。

```
SELECT username FROM users WHERE username IN (
    SELECT username FROM users WHERE email LIKE "%gmail%" );
```

この例は興味深いものではありませんが、より大規模なデータベースでは役立つ可能性があります。

```
SELECT username FROM users WHERE password IS NULL;
```

- 「IS NULL」または「IS NOT NULL」を使用すると、フィールドが NULL かどうかを確認できます。

データを整理する

場合によっては、結果を特定の方法で順序付けて受け取りたいことがあります。

```
SELECT id, username, createdAt FROM users ORDER BY createdAt;
```

「ORDER BY」ボタンを使用すると、特定の条件に従って結果を並べ替えることができます。これは「WHERE」ボタンの後に配置されます。

ユーザーは作成日順に昇順で並び替えられていますが、より正確に並び替えることもできます。

```
SELECT * FROM users ORDER BY active DESC, username ASC;
```

ここでは、アクティブな降順「DESC」でユーザーを取得し、値が等しい場合はユーザー名「ASC」で並べ替えます。

(デフォルトでは値は「ASC」でソートされますが、それを指定するように記述することもできます。)

LIMITとOFFSET

「LIMIT」を使用すると、取得される結果の数を制限できます。

```
SELECT * FROM users LIMIT 5;
```

「OFFSET」を使用すると、最初の結果を無視できます (ここでは最初の 3 つを無視します)。

```
SELECT * FROM users LIMIT 5 OFFSET 3;
```

これら 2 つのプロパティはページ区切りによく使用されます。たとえば、ページあたり 10 件の結果が必要な場合は、 LIMIT 10 と指定します。

- ページ 1:オフセット 0
- ページ2:オフセット10
- ページ3:オフセット20

データを削除

この場合、ユーザー「Maurice」を削除したいと思います。

```
ユーザーから削除;
```

やめてください !これを行うと、テーブル全体が削除されます。

```
DELETE FROM users WHERE username = "Mauritius";
```

ここではユーザー「Maurice」のみを削除します。 「WHERE」を忘れないように注意してください。

データの更新

ユーザーを更新する必要があります。

```
ユーザーを更新し、 active = 1、createdAt = CURRENT_TIMESTAMP に設定します。
```

停止!!!!もう一度、ここですべてのユーザーを更新します。

```
ユーザーを更新します。SET active = 1、createdAt = CURRENT_TIMESTAMP WHERE email LIKE "%gmail.com";
```

更新では、 「WHERE」ステップも忘れます。

さまざまな種類のリクエスト

明確な

ユーザーが使用するさまざまなパスワードを知りたいです (実際には実行されませんが、いずれにしてもハッシュされます)。

```
ユーザーからパスワードを選択します。
```

もちろん、すべてのパスワードは把握していますが、重複したパスワードも持っています。これを避けるには：

```
SELECT DISTINCT password FROM users;
```

「DISTINCT」を使用すると、列から異なる値のみを取得できます。

たとえば、ユーザーがどの国から来ているかを知りたい場合に役立ちます。

機能

次のような実用的な機能がいくつか利用可能です。

```
users からCOUNT(id) を選択します。
```

「COUNT()」を使用すると、結果の数をカウントできます。

```
usersからAVG(id)を選択します。
```

「AVG()」を使用すると、列の平均を取得できます。

```
users からSUM(id)を選択します。
```

「SUM()」を使用すると、列の合計を計算できます。

```
users からMIN(id)を選択します。
```

「MIN()」は列内の最小値を返します。

```
users からMAX(id)を選択します。
```

「MAX()」は列内の最大値を返します。

```
`users`からMONTH(createdAt)を選択します。
```

「MONTH()」は日付の月のみを返します。（年、分などについても同様です。）

```
SELECT CONCAT(ユーザー名, "                : "、メール） FROM ユーザー;
```

「CONCAT()」を使用すると、クエリの結果を 1 つの列に連結できます。

```
選択キャスト(
        (SELECT COUNT(id) FROM users WHERE email LIKE "%gmail%") /

        (ユーザーからCOUNT(id)を選択)
        *100 AS INT );
```

「CAST()」はフィールドの値を変更できます。変更する値を指定し、次に「AS」 、最後に型を指定します（ここではCAST(AS INT)という計算を使用しています）。

「CAST」を使用しない場合、計算結果は「FLOAT」（浮動小数点数)になります。「CAST」を使用すると、 「INT」（小数点なしの数値)になります。

エイリアス

注意深く見てみると、以前の結果の名前が「COUNT(id), AVG(id),...」となっていることがわかりますが、これはあまり読みやすくありません。(CAST() の場合はさらに読みやすくなります)さらに、バック言語を使用する場合は、これらの名前を使用してデータを選択する必要があります。

しかし、列やテーブルの名前を変更して、異なる表示にすることも可能です。
結果。

```
SELECT COUNT(id) AS TotalUser FROM users;
```

「AS」のおかげで、結果は「TotalUser」と呼ばれます。

先ほどの「CAST()」で試してみると、結果がはるかに読みやすくなります。

「AS」という単語を使わずにテーブルに「エイリアス」を設定することもできます。

```
users u からユーザー名を選択します。
```

役に立たないと思われるかもしれませんが、「ジョイント」の有用性がわかります。

グループ化

どのユーザーが「gmail」を使用し、何人が使用していないかを数えたいのですが、これを行うには、次のようにします。

```
SELECT id, (email LIKE '%gmail%') AS gmail FROM `users`;
```

各ユーザーのIDとブール値を示すGmail列を用意します。しかし、IDを数えると次のようになります。

```
SELECT COUNT(id), (email LIKE '%gmail%') AS gmail FROM `users`;
```

合計がカウントされるため、結果は希望どおりにはなりません。また、 「WHERE」句を追加すると、 Gmail を持つユーザーのみが表示されます。

「GROUP BY」を使用して特定の結果をグループ化することができます。

```
SELECT COUNT(id), (email LIKE '%gmail%') AS gmail
    `users`から
    Gmailでグループ化;
```

すると、 「gmail」列の結果をグループ化し、それぞれのIDの合計を表示します。カンマで区切ることで複数のグループを作成できることに注意してください。

持つ

キーワード「WHERE」のように、グループに特化して特定のグループ結果のみを表示することも可能です。また、 「HAVING」の場合は次のようになります。

```
SELECT COUNT(id), (email LIKE '%gmail%') AS gmail
    `users`から
    Gmailでグループ化
    COUNT(id) >= 5 である;
```

ここでは、 「COUNT(id)」が 5 以上の行のみを取得します。