


## 4. SQL入門：SQLFiddleを活用したデータベース操作の基礎と応用

URL: <https://www.kkaneko.jp/de/ds/index.html>

金子邦彦



謝辞：この資料では「いらすとや」のイラストを使用しています

- 
- ① 実践的スキルと問題解決能力の向上
  - ② 自由で自主性を重視する SQL 学習環境
  - ③ 将来の成長の展望

実践的なSQLスキルを身につけることができ、自分自身のペースで効率的に学べる環境が提供され、多様な個人成長の機会が広がります。



# アウトライン

1. イントロダクション
2. オンラインツール SQLFiddle のメリットと基本操作
3. SQL によるデータの追加
4. SQL の全体像
5. テーブル定義 (SQL を使用)
6. データの追加 (SQL を使用)
7. 種々の問い合わせ (クエリ) (SQL を使用)

## 4-1. イントロダクション

# リレーショナルデータベースの仕組み

- データを**テーブル**と呼ばれる**表形式**で保存
- **テーブル間**は**関連**で結ばれる
- 複雑な構造を持ったデータを効率的に管理することを可能

商品

| ID | 商品名 | 単価  |
|----|-----|-----|
| 1  | みかん | 50  |
| 2  | りんご | 100 |
| 3  | メロン | 500 |

関連

購入

| 購入者 | 商品番号 |
|-----|------|
| X   | 1    |
| X   | 3    |
| Y   | 2    |

# リレーショナルデータベースの重要性

1. **データの整合性**：リレーショナルデータベースは、**データの整合性を保持するための機能**を有する。これにより、誤ったデータや矛盾したデータが保存されるのを防ぐことができる。
2. **柔軟な問い合わせ（クエリ）能力**：リレーショナルデータベースのSQL（Structured Query Language）（データベース操作言語）の使用により、**複雑な検索やデータの抽出**が可能になる。
3. **トランザクション機能**：一連の操作全体を一つの単位として取り扱うことができる機能。これにより、**データの一貫性と信頼性が向上**する。
4. **セキュリティ**：**アクセス権限の設定**などにより、セキュリティを確保する。

データの安全な保管，効率的なデータ検索・操作，ビジネスや研究の意思決定をサポート。

# SQL 理解のための前提知識

## ○ テーブル

データを**テーブル**と呼ばれる**表形式**で保存

| ID | 商品名 | 単価  |
|----|-----|-----|
| 1  | みかん | 50  |
| 2  | りんご | 100 |
| 3  | メロン | 500 |

| 購入者 | 商品番号 |
|-----|------|
| X   | 1    |
| X   | 3    |
| Y   | 2    |

## ○ 問い合わせ（クエリ）

- **問い合わせ（クエリ）**は、**データベース**から必要なデータを検索、加工するための指令
- SELECT, FROM, WHERE など、**多様**なコマンドが存在。
- **結合、集計、ソート、副問い合わせ**など、高度な操作も可能

# SQL のキーワード `select`, `from`, `where`

## **select**

問い合わせ（クエリ）のための基本的な命令。

取得したいデータの指定

## **from**

データ取得の対象となるテーブルを指定

例： `select * from` テーブル名;

## **where**

特定の条件を満たす行の選択

例： `select * from` テーブル名 `where` 列1 = 値1;



# 問い合わせ（クエリ）のバリエーション

① 全データの取得

**select \* from** 商品;

② 特定の属性（列）のみ取得

**select** 商品名, 単価 **from** 商品;

③ 条件付き検索

**select** 商品名, 単価 **from** 商品 **where** 単価 > 80;

## 4-2. オンラインツール SQLFiddle のメリットと基本操作

# オンラインで SQL を実行できるサイト

The screenshot shows the SQL Fiddle interface. On the left, there's a sidebar with 'Login', 'Daily Tokens Left: 0', and tabs for 'Chat', 'Editor', and 'History'. The main editor area is titled 'Untitled\*' and contains SQL code for creating a 'Product' table and inserting data. Below the editor, there are 'Execute' and 'Share' buttons. The 'Results' tab is active, displaying a table with 2 rows of product data.

```
1 -- INIT database
2 CREATE TABLE Product (
3   ProductID INT AUTO_INCREMENT KEY,
4   Name VARCHAR(100),
5   Description VARCHAR(255)
6 );
7
8 INSERT INTO Product(Name, Description) VALUES ('Entity Framework Extensions', 'Use <a href="https://entityframeworkextensions.com">Entity Framework Extensions</a> to extend your DbContext with high-performance bulk operations.
9 INSERT INTO Product(Name, Description) VALUES ('Dapper Plus', 'Use <a href="https://dapper-plus.net/">Dapper Plus</a> to extend your IDbConnection with high-performance bulk operations.
10 INSERT INTO Product(Name, Description) VALUES ('C# Eval Expression', 'Use <a href="https://eval-expression.com">C# Eval Expression</a> to evaluate C# expressions in your application.
11
12 -- QUERY database
13 SELECT * FROM Product;
14 SELECT * FROM Product WHERE ProductID = 1;
```

| ProductID | Name                        | Description   |
|-----------|-----------------------------|---|
| 1         | Entity Framework Extensions | Use <a href="https://entityframeworkextensions.com">Entity Framework Extensions</a> to extend your DbContext with high-performance bulk operations. |
| 2         | Dapper Plus                 | Use <a href="https://dapper-plus.net/">Dapper Plus</a> to extend your IDbConnection with high-performance bulk operations.                          |

## SQLFiddle

<http://sqlfiddle.com/>

シンプルなインタフェースがあり、SQLの理解に便利

The screenshot shows the DBFiddle interface. At the top, there are buttons for 'Run', 'Save', 'Load Example', and 'Collaborate', along with a 'Sign in' button and a 'Have any feedback?' link. The interface is split into two main sections: 'Schema SQL' and 'Query SQL'. The 'Schema SQL' section contains a CREATE TABLE statement for an 'employees' table. The 'Query SQL' section contains an INSERT statement and a SELECT statement. Below these, the 'Results' section shows the output of the queries, including a table with 3 rows of employee data.

```
Schema SQL
1 CREATE TABLE employees (
2   id INTEGER,
3   name TEXT,
4   age INTEGER,
5   salary INTEGER,
6   department_id INT);
~

Query SQL
1 INSERT INTO employees VALUES (1, 'Alice', 30, 50000, 1);
2 INSERT INTO employees VALUES (2, 'Bob', 40, 60000, 1);
3 INSERT INTO employees VALUES (3, 'Charlie', 35, 70000, 2);
4 SELECT * FROM employees;
```

Results

Query #1 Execution time: 0ms  
There are no results to be displayed.

Query #2 Execution time: 0ms  
There are no results to be displayed.

Query #3 Execution time: 0ms  
There are no results to be displayed.

Query #4 Execution time: 0ms

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

## DBFiddle

<https://www.db-fiddle.com/>

シンプルなインタフェースがあり、SQLの理解に便利

# オンラインで SQL を学ぶ

## メリット

- **アクセスの手軽さ**: Webブラウザがあれば、どこからでもアクセスでき、学ぶことができる。
- **インストール不要**: データベースソフトウェアのインストールが不要。
- **時間や場所に縛られない**: 自分の都合にあわせて練習できる。

## 注意点

- **制限事項**: オンラインツールは、**全てのSQL機能をカバーしていない**場合や、**大量データの取り扱いが難しい**場合があります。
- **セキュリティ**: 秘密情報はオンラインツールに**アップロードしない**。（オンラインツールでは情報漏洩に気を付ける）

# SQLFiddle のサイトにアクセス

Webブラウザを使用

1. ウェブブラウザを開く
2. アドレスバーにSQLFiddleのURLを入力

<http://sqlfiddle.com/>

3. **MySQL** を選ぶ

URLが分からないときは、Googleなどの**検索エンジン**を利用。  
「**SQLFiddle**」と**検索**し、表示された結果からSQLFiddleの  
ウェブサイトをクリック。

# SQLFiddle でのデータベース管理システムの選択

## SQL Fiddle

Welcome to SQL Fiddle, an online SQL compiler that lets you write, edit, and execute any SQL query.

Choose which SQL language you would like to practice today:

[SQL Server](#)

[SQLite](#)

[PostgreSQL](#)

[MySQL](#)

[MariaDB](#)

[Oracle](#)


[Oracle PLSQL](#)

データベース管理システムの選択  
(この授業では **MySQL** を使用)

# SQLFiddle の画面

上のパネル: SQLの入力 (複数可能)

- ・ テーブル定義 CREATE TABLE
- ・ データの追加 INSERT INTO
- ・ SQL問い合わせ。SELECT, FROM, WHERE など



```
1 -- INIT database
2 CREATE TABLE Product (
3   ProductID INT AUTO_INCREMENT KEY,
4   Name VARCHAR(100),
5   Description VARCHAR(255)
6 );
7
8 INSERT INTO Product(Name, Description) VALUES ('Entity Framework Extensions', 'Use
9 INSERT INTO Product(Name, Description) VALUES ('Dapper Plus', 'Use <a href="https
10 INSERT INTO Product(Name, Description) VALUES ('C# Eval Expression', 'Use <a href
11
12 -- QUERY database
13 SELECT * FROM Product;
14 SELECT * FROM Product WHERE ProductID = 1;
15
```

実行ボタン


Execute

< Share

MySQL

結果ウィンドウ

Results



| ProductID | Name                        | Description   |
|-----------|-----------------------------|---|
| 1         | Entity Framework Extensions | Use <a href="#">Entity Framework Extensions</a> to extend your DbContext with high-performance bulk operations. |
| 2         | Dapper Plus                 | Use <a href="#">Dapper Plus</a> to extend your IDbConnection with high-performance bulk operations.             |

# まとめ

## SQLFiddleの要点

- **アクセス**: Webブラウザから簡単にアクセス

<http://sqlfiddle.com/>

- **インストール不要**: ソフトウェアは不要

## 注意点

- **セキュリティ**: 秘密情報は避ける

## 画面構成

- **上パネル**: テーブル定義, データ追加, SQL問い合わせ
- **下パネル**: 実行結果



## 4-3. SQL によるデータの追加

# SQL によるデータの追加

テーブルに行を追加する

- SQL でのコマンド: **INSERT INTO**

次は、テーブル「朝食と値段」に、3行のデータを追加。

```
INSERT INTO 朝食と値段 VALUES ('A', 'カレーライス', 400);
```

```
INSERT INTO 朝食と値段 VALUES ('B', 'うどん', 250);
```

```
INSERT INTO 朝食と値段 VALUES ('C', 'カレーライス', 400);
```

| 名前 | 朝食     | 値段  |
|----|--------|-----|
| A  | カレーライス | 400 |
| B  | うどん    | 250 |
| C  | カレーライス | 400 |



## 演習 1 . SQLFiddle を用いたSQL の実行

### 【トピックス】

1. SQLFiddle
2. SQL によるテーブル定義
3. SQL によるデータの追加
4. SQL による問い合わせ（クエリ）

Webブラウザを使用

① アドレスバーにSQLFiddleのURLを入力

**http://sqlfiddle.com/**

② 「**MySQL**」を選択



③ 上のパネルに元からある SQL は不要なので消す

④ **上のパネルに、テーブル定義とデータの追加と問い合わせを行う SQL を入れる。**

```
create table 朝食と値段 (  
  名前 text,  
  朝食 text,  
  値段 integer  
);  
insert into 朝食と値段 values ('A', 'カレーライス', 400);  
insert into 朝食と値段 values ('B', 'うどん', 250);  
insert into 朝食と値段 values ('C', 'カレーライス', 400);  
select * from 朝食と値段;
```

⑤ 「Execute」をクリック。下側のウィンドウで、結果を確認。

```
1 create table 朝食と値段 (  
2   名前 text,  
3   朝食 text,  
4   値段 integer  
5 );  
6 insert into 朝食と値段 values ('A', 'カレーライス', 400);  
7 insert into 朝食と値段 values ('B', 'うどん', 250);  
8 insert into 朝食と値段 values ('C', 'カレーライス', 400);  
9 select * from 朝食と値段;  
10
```

Execute

Share

Messages

```
+-----+-----+-----+  
| 名前 | 朝食 | 値段 |  
+-----+-----+-----+  
| A | カレーライス | 400 |  
| B | うどん | 250 |  
| C | カレーライス | 400 |  
+-----+-----+-----+
```

# まとめ

## SQLの**INSERT INTO**によるデータ追加

- テーブルに行（データ）を追加

例

**INSERT INTO 朝食と値段 VALUES ('A', 'カレーライス', 400);**

- オンライン実行（SQLFiddleを使用）

SQLFiddle では、**上のパネル**に、**テーブル定義**（**create table**）と**データ追加**（**insert into**）を書く

## 4-4. SQL の全体像



# 授業での SQL の学習の進め方

授業での SQL の学習は以下の順番で進めます。

【今回の授業】

- **概観：SQLの主要な機能**を簡単に紹介します。

【次回から】

- **基礎を固める**：SELECT, FROM, WHEREなど、基本的な機能をしっかりと理解します。
- **応用に進む**：基本がマスターできたら、JOINやGROUP BYなどの応用的な命令に進みます。

このように**段階を踏んで学ぶ**ことで、**効率的にSQLをマスター**できます。

# 最初に全体像を把握する学習のメリット

- **広範な理解**：SQLの**多様な機能と用途**を**早期に理解**できます。
- **学習意欲の向上**：多くの機能を初めて体験することで、何ができるのかを知り、**興味が湧きます**。
- **効率的な学習**：全体像を先に知ること、**後の学習がスムーズに進みます**。

## 注意点

- **自分のペースで**：最初の授業ですべてをマスターする必要はありません。
- **基礎の確立**：**次回以降**で基本的な部分をしっかり学び、その後で応用に進みます。

# SQL 問い合わせの全体像①

- **select**: データの検索・加工や射影

例 : **SELECT \* FROM employees**

- **from**: 問い合わせ対象テーブルの指定

例 : **FROM employees**

- **where**: 条件に一致する行を選択

例 : **WHERE age > 30**

- **join, on**: 結合、結合条件

例 : **JOIN B ON A.b\_id = B.id**

- **insert into**: 新しい行の追加（挿入）

- **update, set**: 条件に一致する行を更新

- **delete from**: 条件に一致する行を削除

## SQL 問い合わせの全体像②

- **distinct**: 重複行の除去

例 : **SELECT DISTINCT age FROM employees**

- **count**: 行数のカウント

例 : **SELECT COUNT(\*) FROM employees**

- **avg, max, min, sum**: 平均、最大、最小、合計の計算

例 : **AVG(salary), MAX(salary)**

- **group by**: 属性でグループ化

例 : **GROUP BY department\_id**

- **order by**: 並べ替え (ソート)

例 : **ORDER BY age**

- 副問い合わせ: SQL文の中に別のSQL文を埋め込む。

例 : **WHERE salary > (SELECT AVG(salary) FROM employees)**

# SQL のその他のトピックス

- 原則、大文字と小文字を**区別しない**

例 **SELECT** と **select** は同じ意味

- 途中で**改行**しても意味が**変わらない**
- 複数のテーブルから「あるテーブルの特定の属性」を特定するのに「**.**」を使う

例 **T.id** テーブル **T** の **id** 属性

- 種々の**データ型**のサポート

例 **integer** は**整数**, **text** は**テキスト**,

**datetime** は**日付**, **real** は**浮動小数点数**

- 比較演算 **<**, **<=**, **>**, **>=**, **=**, **<>** や範囲指定 **between**
- テキストのパターンマッチ **like**
- 問い合わせ以外のさまざまな機能

**トランザクション** (**begin**, **commit**, **rollback**) 、

性能向上のための**インデックス** (**create index**)

# まとめ

## 態度

- **好奇心**: SQLの多様な機能と用途に**興味を持つ**
- **自主性**: 自分のペースで学び、ときには、疑問点や興味のある点を**自ら調べる**
- **継続性**: 基礎から応用まで**段階的に学ぶ意欲**を持つ

## 授業での SQL の学習の進め方

- **最初は概観**: SQLの主要な機能を**広く**浅く理解する
- **基礎を固める**: 次に、基本的な命令（SELECT, FROM, WHEREなど）をマスターする

## メリットと注意点

- **学習意欲の向上**: 多くの機能を体験することで**興味が湧く**
- **効率的な学習**: 全体像を先に知ること、**後の学習がスムーズ**に進む
- **自分のペースで**: 最初にすべてをマスターする必要は**ない**
- **基礎の確立**: 次回以降で基本をマスター、その後で応用に進む

## 4-5. テーブル定義（SQL を使用）

## 関連する 2つのテーブル

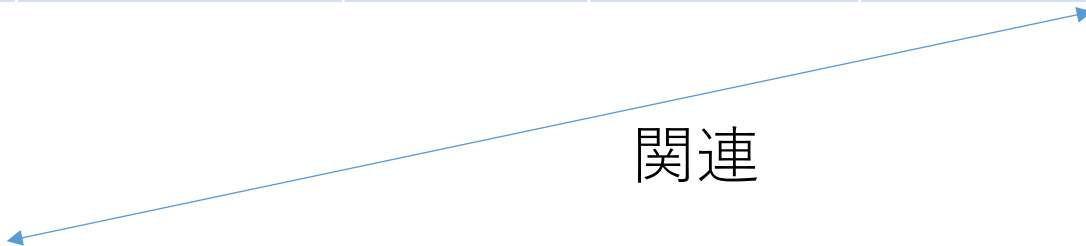
従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

関連





# SQL によるテーブル定義

- テーブル名 : 従業員
- 属性名 : id、name、age、salary、department\_id
- 属性のデータ型 : 数値、テキスト、数値、数値、数値
- データの整合性を保つための制約 : なし

```
CREATE TABLE 従業員 (  
    id INTEGER,  
    name TEXT,  
    age INTEGER,  
    salary INTEGER,  
    department_id INTEGER);
```

# SQL によるテーブル定義

- テーブル名 : **部署**
- 属性名 : **id、name**
- 属性のデータ型 : **数値、テキスト**
- データの整合性を保つための制約 : **なし**

```
CREATE TABLE 部署 (  
    id INTEGER,  
    name TEXT);
```

## 4-6. データの追加 (SQL を使用)

# データ追加のSQL

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

```
INSERT INTO 従業員 VALUES (1, 'Alice', 30, 50000, 1);
```

```
INSERT INTO 従業員 VALUES (2, 'Bob', 40, 60000, 1);
```

```
INSERT INTO 従業員 VALUES (3, 'Charlie', 35, 70000, 2);
```

```
INSERT INTO 部署 VALUES (1, 'HR');
```

```
INSERT INTO 部署 VALUES (2, 'Engineering');
```



## 演習 2. テーブル定義とデータの追加

### 【トピックス】

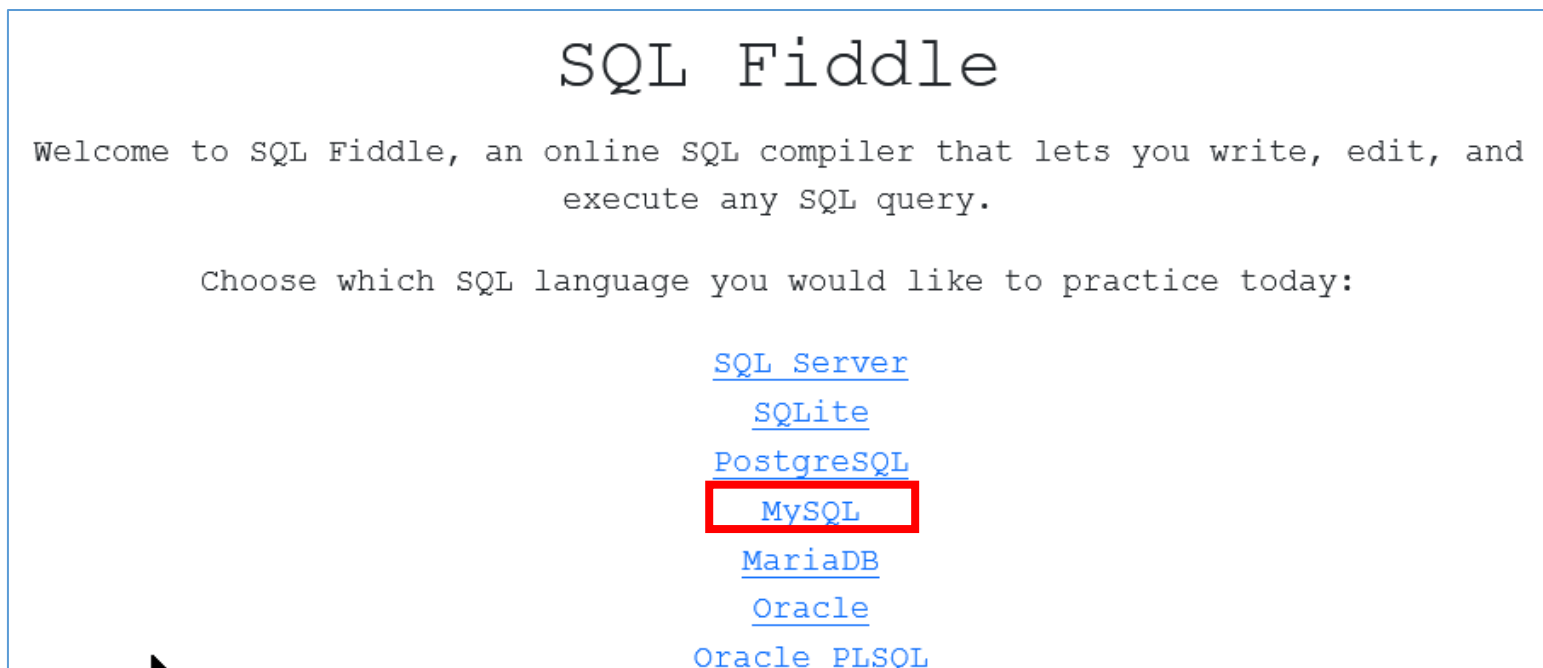
1. SQL によるテーブル定義
2. SQL によるデータの追加
3. 問い合わせ（クエリ）による確認

Webブラウザを使用

① アドレスバーにSQLFiddleのURLを入力

**http://sqlfiddle.com/**

② 「**MySQL**」を選択



③ 上のパネルに元からある SQL は不要なので消す

④ **上のパネルに、テーブル定義とデータの追加と問い合わせを行う SQL を入れる。**

```
CREATE TABLE 従業員 (  
  id INTEGER,  
  name TEXT,  
  age INTEGER,  
  salary INTEGER,  
  department_id INTEGER);  
CREATE TABLE 部署 (  
  id INTEGER,  
  name TEXT);  
INSERT INTO 従業員 VALUES (1, 'Alice', 30, 50000, 1);  
INSERT INTO 従業員 VALUES (2, 'Bob', 40, 60000, 1);  
INSERT INTO 従業員 VALUES (3, 'Charlie', 35, 70000, 2);  
INSERT INTO 部署 VALUES (1, 'HR');  
INSERT INTO 部署 VALUES (2, 'Engineering');  
select * from 従業員;  
select * from 部署;
```

⑤ 「Execute」をクリック。下側のウィンドウで、結果を確認。

```
1 CREATE TABLE 従業員 (  
2     id INTEGER,  
3     name TEXT,  
4     age INTEGER,  
5     salary INTEGER,  
6     department_id INTEGER);  
7 CREATE TABLE 部署 (  
8     id INTEGER,  
9     name TEXT);  
10 INSERT INTO 従業員 VALUES (1, 'Alice', 30, 50000, 1);  
11 INSERT INTO 従業員 VALUES (2, 'Bob', 40, 60000, 1);  
12 INSERT INTO 従業員 VALUES (3, 'Charlie', 35, 70000, 2);  
13 INSERT INTO 部署 VALUES (1, 'HR');  
14 INSERT INTO 部署 VALUES (2, 'Engineering');  
15 select * from 従業員;  
16 select * from 部署;  
17
```

Execute Share

Results

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

| id | name |
|----|------|
|----|------|

あとで使用するのでブラウザを閉じないこと



# 余裕がある人向け

次の SQL を試してみる。

**SELECT age FROM 従業員;**

**SELECT \* FROM 従業員 WHERE age = 30;**

結果（表示が見えないときは、スクロールバーでスクロール）

| age |
|-----|
| 30  |
| 40  |
| 35  |

| id | name  | age | salary | department_id |
|----|-------|-----|--------|---------------|
| 1  | Alice | 30  | 50000  | 1             |

あとで使用するのでブラウザを閉じないこと

## 4-7. 種々の問い合わせ（クエリ）（SQL を使用）

# 選択

選択は、特定の条件に一致する行を選択

元のテーブル  
テーブル名: P

| CUST | PRODUCT | PRICE |
|------|---------|-------|
| 100  | P100    | 20    |
| 101  | P100    | 30    |
| 101  | X200    | 1000  |
| 102  | P300    | 100   |

誰が、何を、いくらで買ったか

## 選択

**SELECT \***  
**FROM P**

**WHERE PRICE > 50;**

|  | CUST | PRODUCT | PRICE |
|--|------|---------|-------|
|  | 101  | X200    | 1000  |
|  | 102  | P300    | 100   |

# 選択と射影の組み合わせ

## 射影は、必要な属性のみを抽出

元のテーブル  
テーブル名: P

| CUST | PRODUCT | PRICE |
|------|---------|-------|
| 100  | P100    | 20    |
| 101  | P100    | 30    |
| 101  | X200    | 1000  |
| 102  | P300    | 100   |

誰が、何を、いくらで買ったか

## 選択と射影の組み合わせ

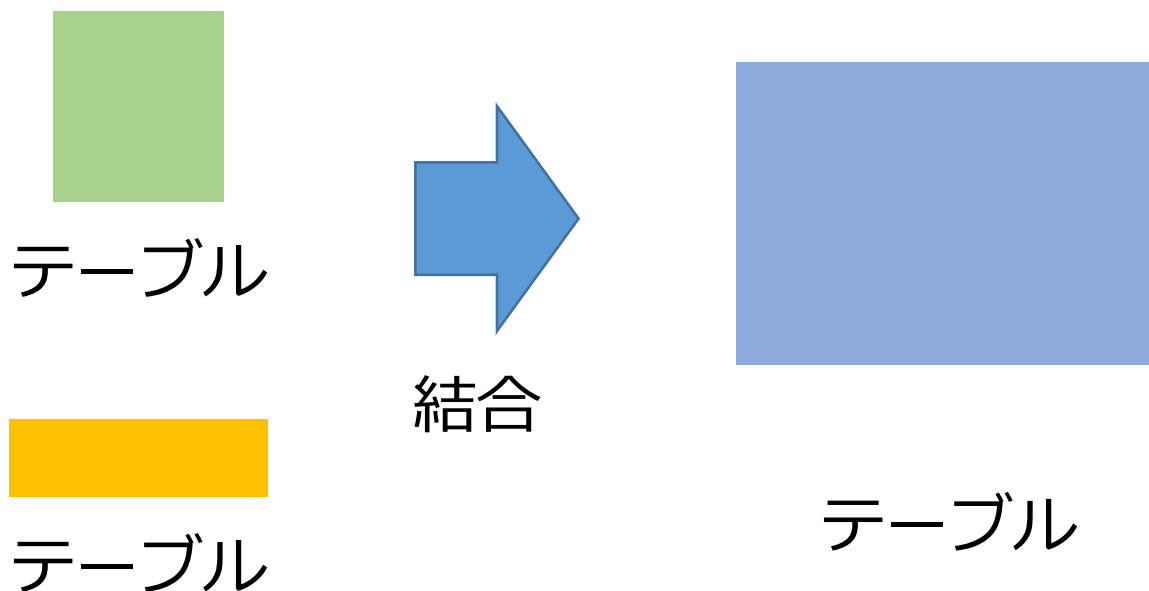
```
SELECT PRODUCT  
FROM P  
WHERE PRICE > 50;
```

| PRODUCT |
|---------|
| X200    |
| P300    |

# テーブル結合

**結合**は、テーブル間の関連に基づいて**複数のテーブルを1つにまとめる操作**

例：従業員テーブルと部署テーブルを結合，従業員の名前と所属部署の名前を**1つのテーブル**に集める．



# 重複行除去

元のテーブル  
テーブル名: P

| CUST | PRODUCT | PRICE |
|------|---------|-------|
| 100  | P100    | 20    |
| 101  | P100    | 30    |
| 101  | X200    | 1000  |
| 102  | P300    | 100   |

誰が、何を、いくらで買ったか

重複行除去しない

SELECT CUST  
FROM P;

| CUST |
|------|
| 100  |
| 101  |
| 101  |
| 102  |

重複行除去する

SELECT DISTINCT CUST  
FROM P;

| CUST |
|------|
| 100  |
| 101  |
| 102  |

# グループ化

GROUP BY句を使用.

指定した属性でデータをグループ化する機能.

各グループに対して集計関数（例：count）を適用

元のテーブル  
テーブル名: P

| CUST | PRODUCT | PRICE |
|------|---------|-------|
| 100  | P100    | 20    |
| 101  | P100    | 30    |
| 101  | X200    | 1000  |
| 102  | P300    | 100   |

## 集計の例

**SELECT CUST, COUNT(\*)**  
**FROM P**  
**GROUP BY CUST;**

誰が、何を、いくらで買ったか

|     |   |
|-----|---|
| 100 | 1 |
| 101 | 2 |
| 102 | 1 |

CUST 属性について 100, 101, 102  
のグループ化が行われる

# 並べ替え（ソート）

## 指定した属性についてソートする

元のテーブル  
テーブル名: P

| CUST | PRODUCT | PRICE |
|------|---------|-------|
| 100  | P100    | 20    |
| 101  | P100    | 30    |
| 101  | X200    | 1000  |
| 102  | P300    | 100   |

誰が、何を、いくらで買ったか

# 並べ替え（ソート）

**SELECT \***

**FROM P**

**ORDER BY PRICE;**

|  | CUST | PRODUCT | PRICE |
|--|------|---------|-------|
|  | 100  | P100    | 20    |
|  | 101  | P100    | 30    |
|  | 102  | P300    | 100   |
|  | 101  | X200    | 1000  |

PRICE で並べ替え



## 副問い合わせ

元のテーブル  
テーブル名: P

| CUST | PRODUCT | PRICE |
|------|---------|-------|
| 100  | P100    | 20    |
| 101  | P100    | 30    |
| 101  | X200    | 1000  |
| 102  | P300    | 100   |

誰が、何を、いくらで買ったか

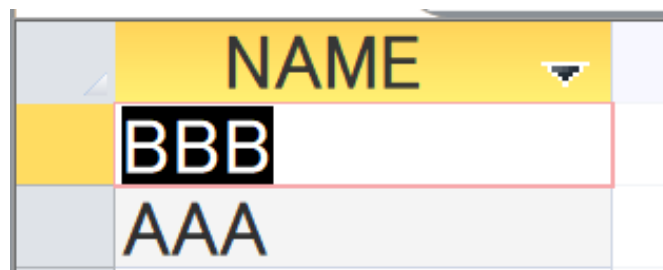
テーブル名: C

| ID  | NAME | MEMBER     |
|-----|------|------------|
| 100 | BBB  | 2023/10/02 |
| 101 | AAA  | 2023/10/10 |
| 102 | CCC  | 2023/10/15 |

名簿（番号と氏名と入会日）

## 副問い合わせ

```
SELECT NAME  
FROM C  
WHERE ID IN  
  (SELECT CUST  
   FROM P  
   WHERE PRODUCT='P100');
```



| NAME |
|------|
| BBB  |
| AAA  |

P100 を買ったのは誰？



## 演習 3. SQL 問い合わせ (クエリ) の概観

### 【トピックス】

1. データの検索や射影
2. 問い合わせ対象テーブルの指定
3. 選択
4. 結合、結合条件
5. 重複行の除去
6. 行数のカウント
7. 平均、最大、最小、合計の計算
8. 属性でグループ化
9. 並べ替え (ソート)
10. 副問い合わせ

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

## ① SELECT

SELECTは、データベースからデータを取得する

```
SELECT * FROM 従業員;
```

```
SELECT name, age FROM 従業員;
```

2つめの SQL では、name, age 属性を指定

結果（表示が見えないときは、スクロールバーでスクロール）

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

| name    | age |
|---------|-----|
| Alice   | 30  |
| Bob     | 40  |
| Charlie | 35  |

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

## ② FROM（問い合わせ対象テーブルの指定）

FROMは、問い合わせ（クエリ）が対象とするテーブルを指定

```
SELECT name FROM 従業員;  
SELECT name FROM 部署;
```

結果（表示が見えないときは，スクロールバーでスクロール）

```
name  
Alice  
Bob  
Charlie
```

```
name  
HR  
Engineering
```

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

### ③ WHERE (選択)

WHEREは、特定の条件に一致する行を選択するために使う。

```
SELECT * FROM 従業員 WHERE age > 30;
```

age の値が 30 より大きいという条件

結果（表示が見えないときは，スクロールバーでスクロール）

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

#### ④ JOIN、ON（結合、結合条件）

関係のあるテーブルを、結合条件を指定して1つに結合する。

```
SELECT 従業員.name, 部署.name  
FROM 従業員  
JOIN 部署 ON 従業員.department_id = 部署.id;
```

結果（表示が見えないときは、スクロールバーでスクロール）

| name    | name_1      |
|---------|-------------|
| Alice   | HR          |
| Bob     | HR          |
| Charlie | Engineering |

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

## ⑤ DISTINCT（重複行の除去）

```
SELECT DISTINCT department_id FROM 従業員;
```

結果（表示が見えないときは，スクロールバーでスクロール）

| department_id |
|---------------|
| 1             |
| 2             |



従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

## ⑥ COUNT（行数のカウント）

```
SELECT COUNT(*) FROM 従業員;
```

テーブル全体の行数をカウント

結果（表示が見えないときは、スクロールバーでスクロール）

```
COUNT (*)
```

```
3
```

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

## ⑦ AVG、MAX、MIN、SUM（平均、最大、最小、合計の計算）

これらは、数値データに対する平均、最大、最小、合計を計算する。

```
SELECT AVG(salary), MAX(salary), MIN(salary), SUM(salary)
FROM 従業員;
```

salary の値の平均、最大、最小、合計

結果（表示が見えないときは、スクロールバーでスクロール）

```
COUNT (*)
```

```
3
```

| AVG(salary) | MAX(salary) | MIN(salary) | SUM(salary) |
|-------------|-------------|-------------|-------------|
| 60000.0000  | 70000       | 50000       | 180000      |

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

## ⑧ GROUP BY（属性でグループ化）

GROUP BY は、指定した属性についてグループ化する。

```
SELECT department_id, COUNT(*)  
FROM 従業員  
GROUP BY department_id;
```

department\_id の属性について 1 と 2 でグループ化が行われ、  
それぞれの行数をカウント

結果（表示が見えないときは、スクロールバーでスクロール）

| department_id | COUNT (*) |
|---------------|-----------|
| 1             | 2         |
| 2             | 1         |

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

## ⑨ ORDER BY (並べ替え (ソート))

ORDER BY は、指定した属性についてソートする。

```
SELECT name, age  
FROM 従業員  
ORDER BY age DESC;
```

age 属性の値が高い行を先に表示

結果 (表示が見えないときは, スクロールバーでスクロール)

| name    | age |
|---------|-----|
| Bob     | 40  |
| Charlie | 35  |
| Alice   | 30  |

従業員

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 1  | Alice   | 30  | 50000  | 1             |
| 2  | Bob     | 40  | 60000  | 1             |
| 3  | Charlie | 35  | 70000  | 2             |

部署

| id | name        |
|----|-------------|
| 1  | HR          |
| 2  | Engineering |

## ⑩ 副問い合わせ

INや括弧()は、SQLでさまざまな用語があるが、問い合わせ内に別の問い合わせを含めるときにも使える。

これは、ある問い合わせの結果を別の問い合わせのために使うもの。

```
SELECT * FROM 従業員  
WHERE salary > (SELECT AVG(salary) FROM 従業員);
```

salary 属性の値が、salary 属性の平均よりも高いという条件で選択結果（表示が見えないときは、スクロールバーでスクロール）

| id | name    | age | salary | department_id |
|----|---------|-----|--------|---------------|
| 3  | Charlie | 35  | 70000  | 2             |

# 全体まとめ①

## オンラインツール SQLFiddle のメリット

- **アクセスの手軽さ**: Webブラウザがあればどこからでもアクセス可能。
- **インストール不要**: データベースソフトウェアのインストールが不要。
- **時間や場所に縛られない**: 自分の都合に合わせて練習可能。

## SQLによるデータの追加

- INSERT INTO コマンドでテーブルに行を追加。

例: **INSERT INTO 朝食と値段 VALUES ('A', 'カレーライス', 400);**

## 授業でのSQLの学習の進め方

- 概観から始め、次回から、基本的な機能をマスターしながら、応用に進む。
- 学習意欲の向上と効率的な学習が可能。

## SQL問い合わせの全体像

- SELECT, FROM, WHERE など、**多様**なコマンドが存在。
- **結合、集計、ソート、副問い合わせ**など、高度な操作も可能。

# 全体まとめ②

## テーブル定義

- CREATE TABLE でテーブルを定義。

例: **CREATE TABLE 従業員 (id INTEGER, name TEXT, age INTEGER, salary INTEGER, department\_id INTEGER);**

## 主要なSQLの機能

- 1.データの検索や射影 (SELECT)
- 2.問い合わせ対象テーブルの指定 (FROM)
- 3.条件による選択 (WHERE)
- 4.テーブルの結合 (JOIN, ON)
- 5.重複行の除去 (DISTINCT)
- 6.行数のカウント (COUNT)
- 7.平均、最大、最小、合計の計算 (AVG, MAX, MIN, SUM)
- 8.属性でグループ化 (GROUP BY)
- 9.並べ替え（ソート） (ORDER BY)
- 10.副問い合わせ



## ① 実践的スキルと問題解決能力の向上

SQLの基本から応用までを学び、オンラインツール（SQLFiddle）での演習を、アクティブラーニングで実施することで、即戦力となるスキルと論理的思考が身につきます。

## ② 自由で自主性を重視する SQL 学習環境

場所や時間に縛られず、自分自身のペースで学べるオンライン環境が、学習意欲を高め、自信をつける基盤を提供します。

## ③ 将来の成長の展望

SQLは多くの職種で求められるスキルです。基礎からしっかりと学ぶことで、将来的に多様なキャリアパスが開かれ、自信と成長の機会が広がります。

実践的なSQLスキルを身につけることができ、自分自身のペースで効率的に学べる環境が提供され、多様な個人成長の機会が広がります。



## 自習 1 : テーブルの作成とデータの追加

目的: SQLの基本的なテーブル作成とデータ挿入について、理解を深める

**SQLFiddleでCREATE TABLEとINSERT INTOを使って、従業員と部署のテーブルを定義し、データを追加してみましよう。資料をなるべく見ずに、どこまで自力で行えるか確認してみましよう。**

自習は提出の必要はありません。

## 自習 2 : 単純なデータの検索

目的: SELECTとFROMを使った基本的なデータの検索

従業員テーブルから、**age の属性だけ**を射影する SQLを考え、実行して確認してみましょう

自習は提出の必要はありません。

## 自習 3 : 条件に基づいたデータの検索

目的: WHEREを使って条件に合ったデータを選択

従業員テーブルから、**年齢が 35 より大きい**従業員のデータを選択する SQLを考え、実行して確認してみましょう

自習は提出の必要はありません。

## 自習 1 . 解答例

```
CREATE TABLE 従業員 (  
    id INTEGER,  
    name TEXT,  
    age INTEGER,  
    salary INTEGER,  
    department_id INTEGER);  
CREATE TABLE 部署 (  
    id INTEGER,  
    name TEXT);  
INSERT INTO 従業員 VALUES (1, 'Alice', 30, 50000, 1);  
INSERT INTO 従業員 VALUES (2, 'Bob', 40, 60000, 1);  
INSERT INTO 従業員 VALUES (3, 'Charlie', 35, 70000, 2);  
INSERT INTO 部署 VALUES (1, 'HR');  
INSERT INTO 部署 VALUES (2, 'Engineering');
```