WSL & PostgreSQL 開発環境構築ガイド

学校の授業環境のローカル再現と SQL 操作まとめ

hrm

目 次

1 WSL 環境構築 (gcc & PostgreSQL)

VScode の WSL ターミナルで、C 言語コンパイラ (gcc) とデータベース (PostgreSQL) をインストールする。

```
# 1. パッケージリストを更新
sudo apt update

# 2. 言語開発ツールC (gcc, などmake) をインストール
sudo apt install build-essential

# 3. PostgreSQL サーバーとクライアント() をインストール
sudo apt install postgresql postgresql-client
```

2 PostgreSQL 初期設定

インストール後、PostgreSQL サービスを開始し、SQL 操作用の自分のユーザーを作成する。

2.1 サービス開始と管理者ログイン

```
# 1. サービスを開始PostgreSQL
sudo service postgresql start

# 2. 'postgres' 管理者ユーザーになる
sudo -i -u postgres

# 3. を起動psql ここから(SQL)
psql
```

2.2 ロール (ユーザー) と DB の作成

psql 内で、自分の WSL ユーザー名 (例: hrm) をロールとして作成する。

```
-- ユーザー 'hrm' をパスワード '0000' で作成スーパーユーザー権限付与 ()
CREATE ROLE hrm WITH LOGIN SUPERUSER PASSWORD '0000';

-- 'hrm' が所有するデータベース 'my_practice_db' を作成
CREATE DATABASE my_practice_db OWNER hrm;

-- を終了psql
\q
```

```
# 'postgres' ユーザーから抜ける
exit
```

3 基本的な SQL 操作

3.1 データベースへの接続

作成したデータベース (my_practice_db) に、作成したユーザー (hrm) で接続する。

```
# -d [名DB] -U ユーザー名[] -W パスワード入力()
psql -d my_practice_db -U hrm -W
```

3.2 DDL (データ定義言語)

テーブルの構造を定義するコマンド。

```
-- 'list' テーブルの作成 (を主キーにno)
CREATE TABLE list (
    no INT PRIMARY KEY,
    name TEXT,
    sex TEXT
);

-- カラムの追加
ALTER TABLE list ADD COLUMN age INT;

-- テーブル自体の削除
DROP TABLE list;
```

3.3 DML (データ操作言語)

テーブル内のデータを操作するコマンド。

```
-- データの挿入複数行 ()
INSERT INTO list (no, name, sex) VALUES
  (1, 'Tanaka', 'Male'),
  (2, 'Suzuki', 'Female');

-- データの更新 (の指定が非常に重要WHERE)
UPDATE list SET age = 25 WHERE no = 2;

-- データの全削除テーブルの枠は残る ()
DELETE FROM list;
```

3.4 DQL (データ問い合わせ言語) と ORDER BY

データを検索するコマンド。ORDER BY を使わないと、結果の順序は保証されない。

```
-- 全件表示順序は保証されない ()
SELECT * FROM list;

-- 順でソートして表示no 順序を保証()
SELECT * FROM list ORDER BY no;
```

3.5 psql 専用コマンド

SQL 文ではなく、psql クライアントを操作するコマンド。

```
\dt       -- テーブル一覧を表示
\d list   -- 'list' テーブルの詳細カラム一覧 () を表示
\q        -- を終了psql
```

4 応用 SQL (正規化と JOIN)

'Male' のような文字列を直接保存するのではなく、別の「マスタテーブル」を参照するように変更し、データの整合性を高める（正規化）。

4.1 マスタテーブルの作成

性別の対応表 (sex_master) を作成する。

```
-- 'sex_master' テーブルを作成
CREATE TABLE sex_master (
  sex_id INT PRIMARY KEY,
  sex_name TEXT
);

-- マスタデータ対応表 () を挿入
INSERT INTO sex_master (sex_id, sex_name) VALUES
  (1, 'Male'),
  (2, 'Female');
```

4.2 list テーブルの改造

既存の list テーブルの sex (TEXT 型) を、sex_id (INT 型) を使うように改造する。

```
-- 1. 新しい数値用カラムを追加
ALTER TABLE list ADD COLUMN sex_id INT;

-- 2. 既存の文字列データから数値に変換
UPDATE list SET sex_id = 1 WHERE sex = 'Male';
UPDATE list SET sex_id = 2 WHERE sex = 'Female';

-- 3. 古い文字列カラムを削除
ALTER TABLE list DROP COLUMN sex;
```

4.3 外部キー (FOREIGN KEY) の設定

データの不整合 (sex_id に 3 など「ありえない値」が入る) を防ぐための制約。

```
ALTER TABLE list
ADD CONSTRAINT fk_list_sex
FOREIGN KEY (sex_id) -- 'list' の 'sex_id' が
REFERENCES sex_master(sex_id); -- 'sex_master' の 'sex_id' を参照する
```

4.4 INNER JOIN

2 つのテーブルを sex_id で結合し、no, name, sex_name, age を一覧表示する。

```
SELECT
    list.no,
    list.name,
    sex_master.sex_name, -- 結合したテーブルのカラム
    list.age
FROM
    list
INNER JOIN
    sex_master ON list.sex_id = sex_master.sex_id
ORDER BY
    list.no;
```

5　C 言語との連携

C 言語の printf で SQL 文を出力し、それをパイプ (|) で psql に渡して実行する。

5.1　C ソースコード (list_insert.c)

正規化後の list テーブル (no, name, age, sex_id) に 50 件のランダムデータを挿入するプログラム。(実行前に TRUNCATE でテーブルを空にし、PRIMARY KEY 重複エラーを防ぐ)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    // 乱数の「種」を初期化
    srand((unsigned int)time(NULL));

    char* names[] = {"Sato", "Suzuki", "Takahashi", "Tanaka", "Watanabe"};
    int i;

    // 最初にテーブルを空にする文を出力SQL
    // (PRIMARY KEY 重複エラーを防ぐため)
    printf("TRUNCATE TABLE list;\n");

    // 件分のデータを作成するループ50
    for (i = 1; i <= 50; i++) {

        int no = i; // no を 1 から 50 まで指定
        char* name = names[rand() % 5];
        int age = rand() % 101;
        int sex_id = (rand() % 2) + 1; // 1 (Male) or 2 (Female)

        // 'list' テーブルの最終形に合わせた文INSERT
        printf("INSERT INTO list (no, name, age, sex_id) VALUES (%d, '%s',
           %d, %d);\n",
               no, name, age, sex_id);
    }

    return 0;
}
```

5.2　コンパイルと実行

my_sql_project ディレクトリ内で実行する。

```
# 1. ファイルをコンパイルC 実行ファイル( 'list_insert' を作成)
gcc -o list_insert list_insert.c

# 2. プログラムの出力C (文SQL) を psql に流し込む
./list_insert | psql -d my_practice_db -U hrm -W
```

6　主なトラブルシューティング

- **Bash:** cc1: fatal error: list_insert.c: No such file or directory

原因: C ファイルがないディレクトリで gcc を実行している。
対策: `cd ~/my_sql_project` で正しいディレクトリに移動する。

- **C (Compile Error):** `error: 'null' undeclared`

  原因: C 言語のヌルポインタ定数は小文字の `null` ではなく大文字の `NULL`。
  対策: `time(NULL)` に修正する。

- **C (Compile Error):** `error: expected ';' before ...`

  原因: C 言語の文の末尾のセミコロン (;) が抜けている。
  対策: `srand((unsigned int)time(NULL));` のように ; を追加する。

- **PostgreSQL (Connection Error):** `FATAL: database "hrm" does not exist`

  原因: `psql` とだけ実行すると、ユーザー名と同じ DB に接続しようとする。
  対策: `psql -d my_practice_db -U hrm -W` のように `-d` で DB 名を明示的に指定する。

- **PostgreSQL (INSERT Error):** `violates not-null constraint` **(on column "no")**

  TODO: C コードが `no` を指定しない `INSERT` 文 (SERIAL 用) を出力していたが、テーブル側が `no INT PRIMARY KEY` (NULL 不可) だった。
  対策: C コードを修正し、`no` を含む `INSERT` 文を生成するようにした。