

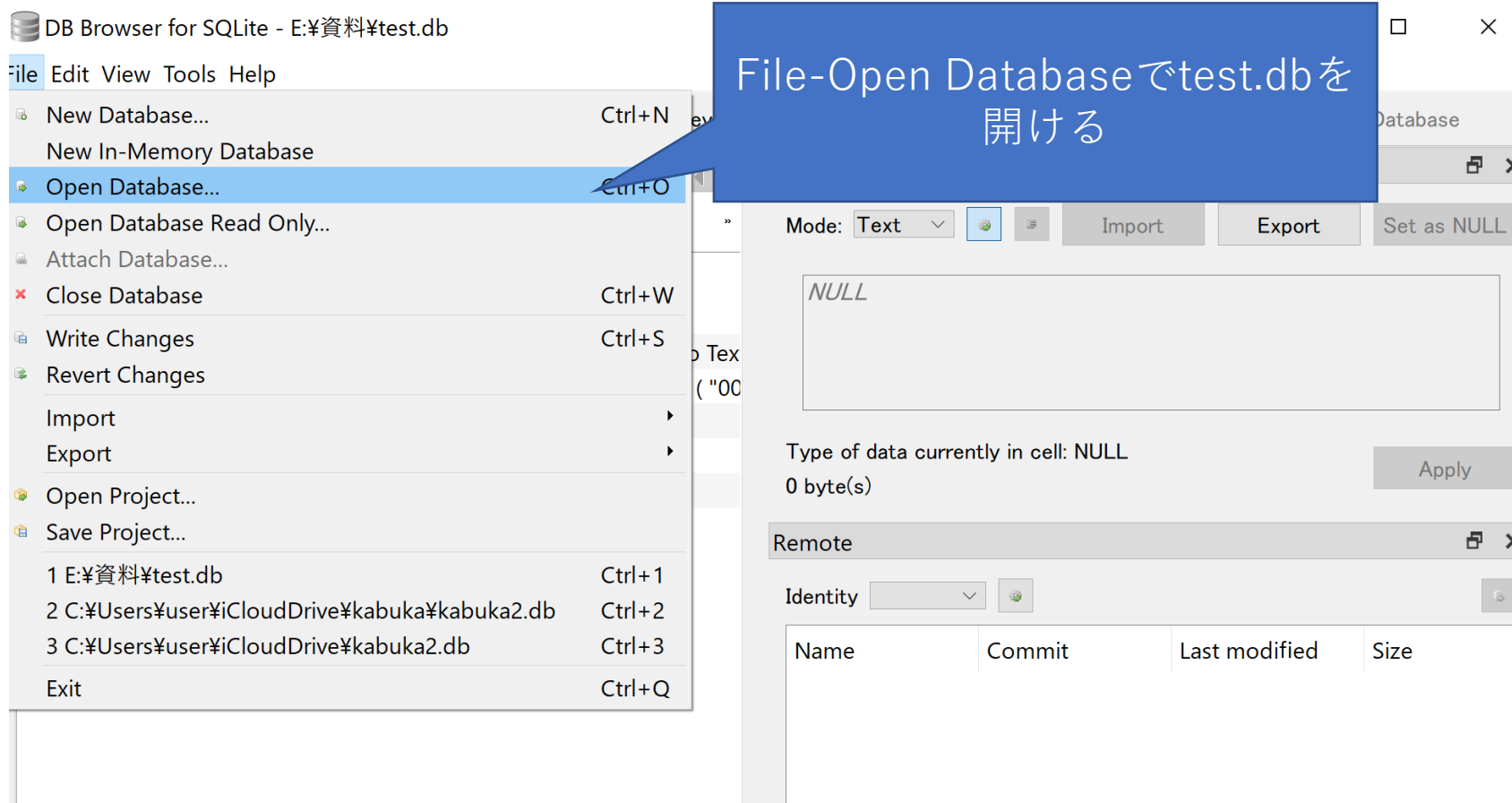
SQL

DB browser for SQLite

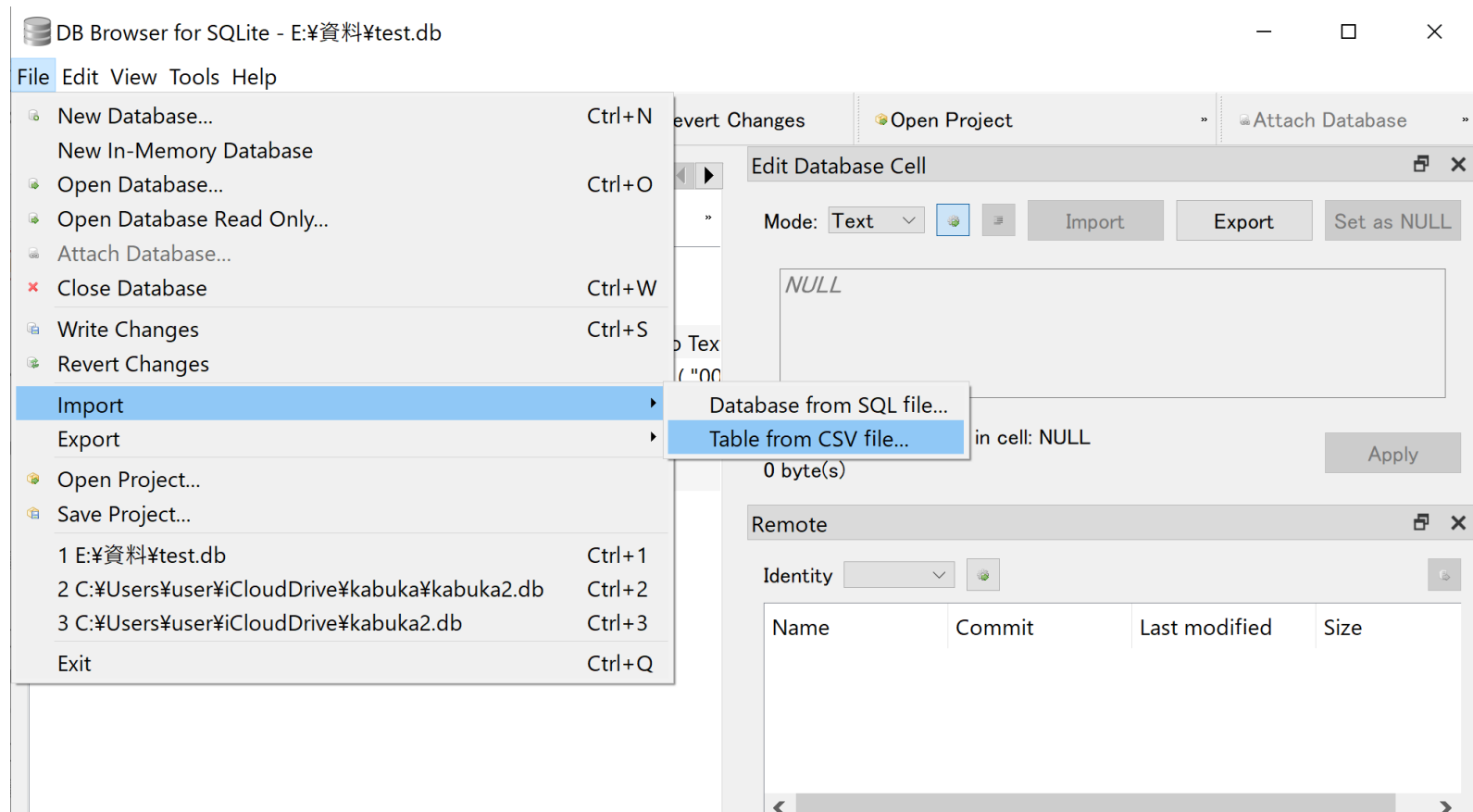
- DB browser for SQLiteをインストールする

import文を使う (DB Browser for SQLite)

- **giants.csv**をgiantsのテーブルに追加する



File-Importer-Table from CSV



gplayer.csvを選択する

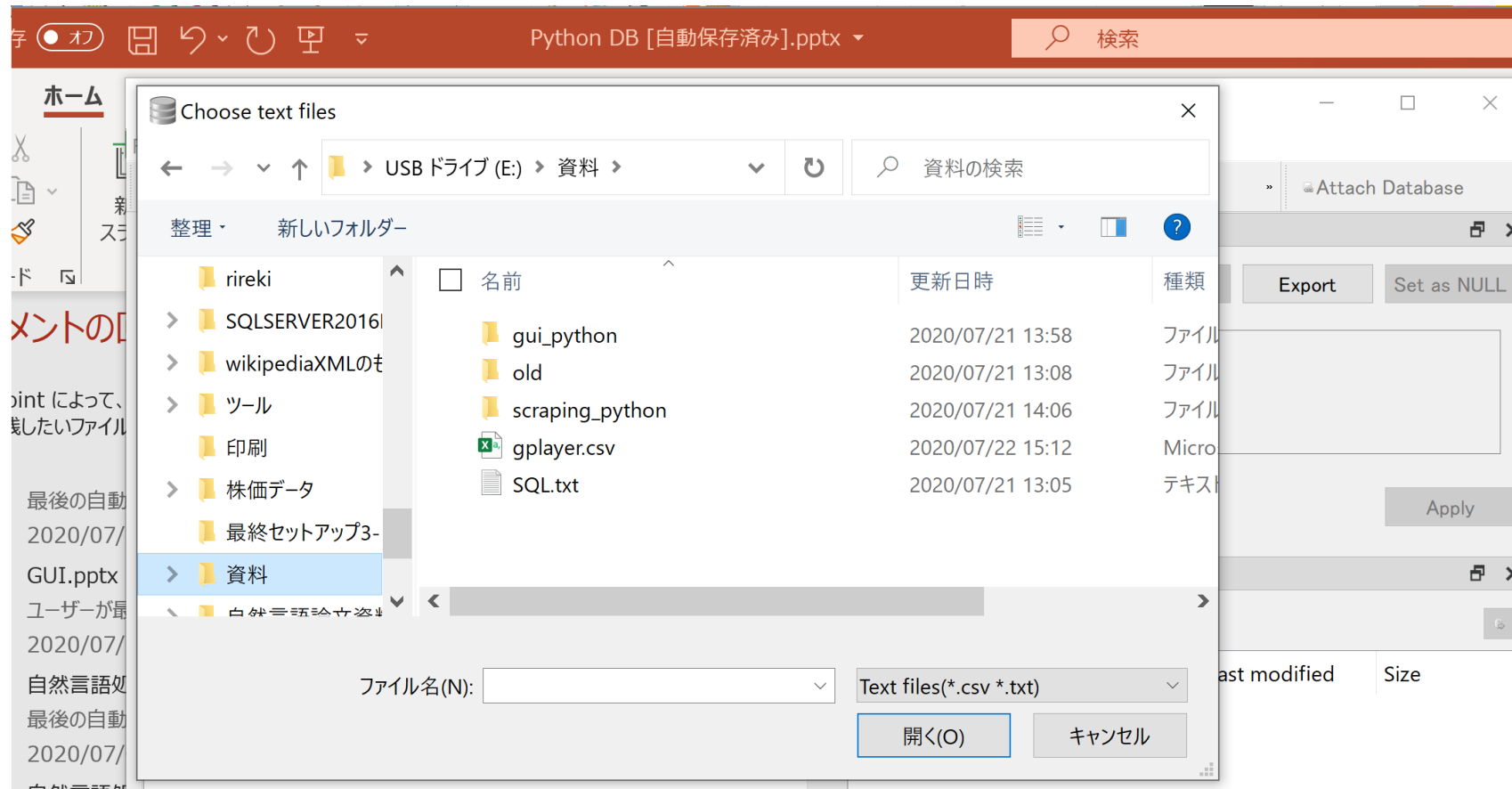


Table Name入力とColumn
names in fist lineのCheckを外
す


 Import CSV file

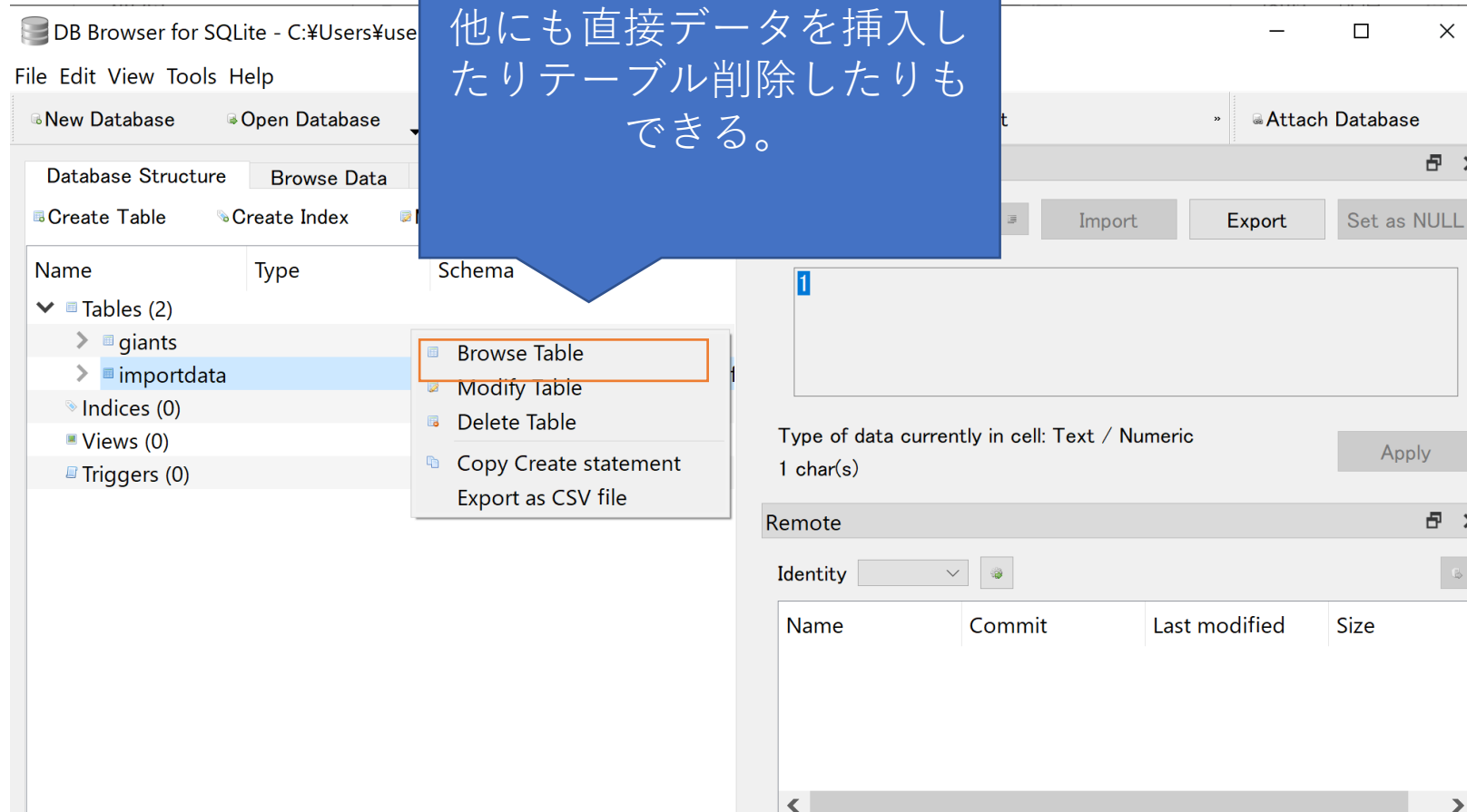
Table name	<input type="text" value="importdata"/>
Column names in first line	<input type="checkbox"/>
Field separator	<input type="text" value=","/>
Quote character	<input type="text" value="\"/>
Encoding	<input type="text" value="UTF-8"/>
Trim fields?	<input checked="" type="checkbox"/>
<input type="button" value="Advanced"/>	

	field1	field2	field3
1	5	XXXX	175

OK

Cancel

データを見る

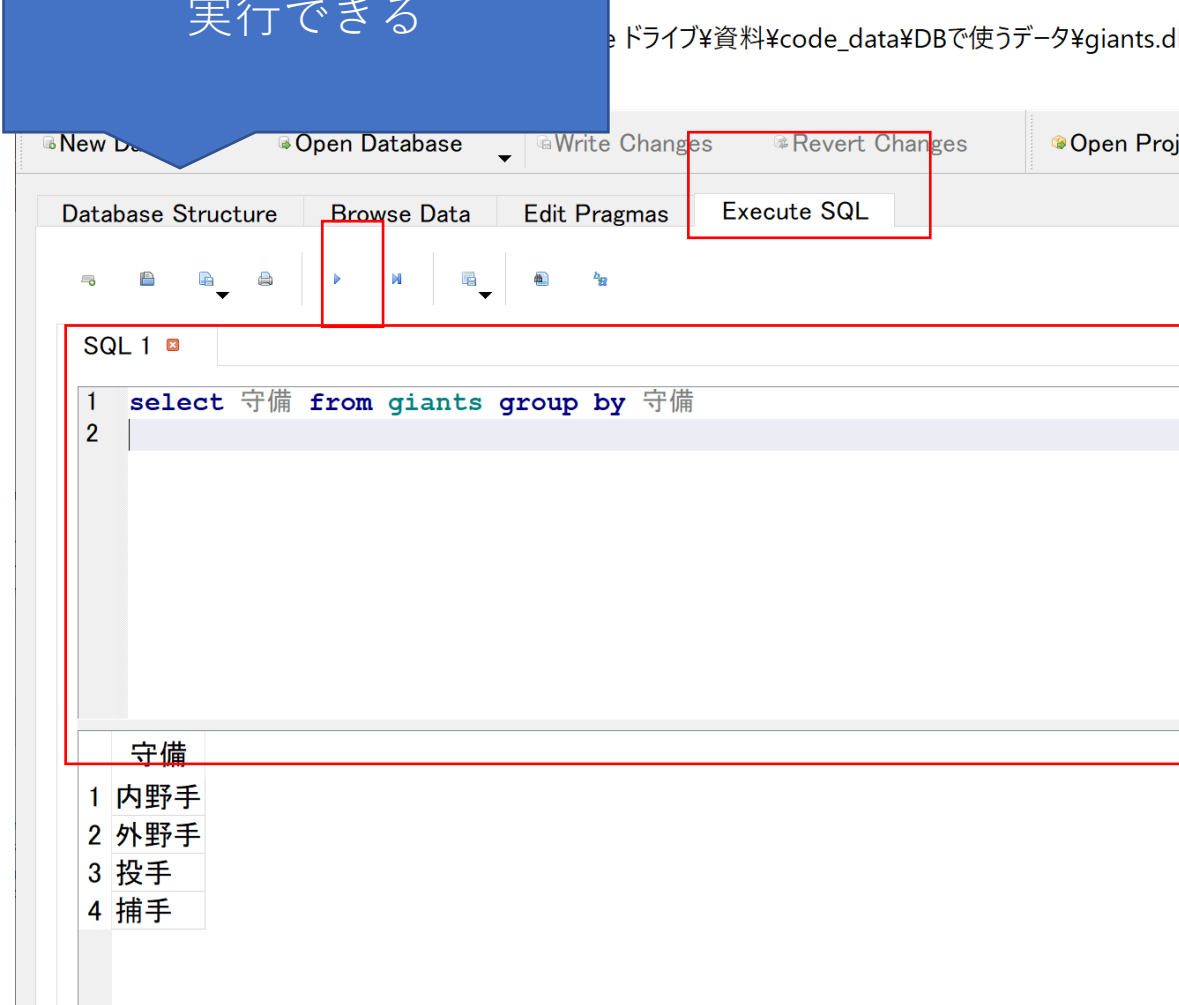


select

DB browser for SQLiteで
giants.dbでデータベースを開ける

SQL

SQLを書くことができ
実行できる



SQLを書くことができ
実行できる

テーブルスキーマ (テーブル名giants)

```
CREATE TABLE "giants" (  
    "選手名"    TEXT,  
    "守備"      TEXT,  
    "生年月日" TEXT,  
    "身長"      INTEGER,  
    "体重"      INTEGER  
);
```

データの検索(select)

select フィールド名,

セミコロンを入れる

...

フィールド名

from テーブル名

where フィールド名=検索値

※フィールド値のテキスト型はシングルクォートで囲む

select文の*

select * from テーブル名ですべてのフィールド
のデータを表示する

select フィールド,フィールド・・・ from テーブル名で表示
したいフィールド名を表示する

例

```
select * from giants
```

=

```
select 選手名,守備,生年月日,身長,体重 from giants
```

アスタリスクはすべてのフィールドを表す。

検索

- **where** フィールド=値

文字列の場合はシングルクォートで囲む
不等号も使える

例

```
select * from giants where 守備='内野手'
```

```
select * from giants where 身長>=170
```

など

論理演算子 and or

体重が75より大きいでかつ身長が170より大きい
例

```
select * from giants where 体重>75 and 身長  
>170
```


比較演算子

$a = b$

aとbは等しい

$a \neq b$

aとbは等しくない

$a > b$

aはbより大きい

$a \geq b$

aはb以上

$a < b$

aはbより小さい

$a \leq b$

aはb以下

aはカラムbは数値

重複のないデータを抜き出す

select フィールド from テーブル
group by フィールド

例

select 守備 from giants group by 守備

DB Browser for SQLite - C:\Users\user\Google ドライブ\資料\giants.db

File Edit View Tools Help

New Database Open Database Write Changes Revert Changes

Database Structure

Browse Data

Edit Pragmas

Execute SQL



SQL 1

```
1 select 守備 from giants group by 守備
```

	守備
1	内野手
2	外野手
3	投手
4	捕手

並び替え order by

- 昇順の場合は `asc` 、降順の場合は `desc` （指定しない場合に）
- 例
- `select * from giants order by 身長 asc;`
- `select * from giants order by 身長 desc;`

あいまい検索 like

- `select * from giants where 選手名 like '坂%';`
%は任意の文字

平均,合計,最大

- select avg(体重) from giants;
- select sum(体重) from giants;
- select max(体重) from giants;

count関数

データの個数を求めるときに使う
テーブルgiantsの数を求めるとき
select count(*) from giantsで求まる。

さらに

SELECT COUNT(DISTINCT 守備) from giants

で重複のないデータの数が出てくる

distinctは重複がないデータをとる

演習

SELECT 守備, COUNT(守備) FROM giants group by 守備
はどのような結果になるか考察してください。

演習(select)

01. 以下のデータベース/テーブルを作成せよ。

データベース名：test_database

テーブル名：test_usertable

UserId	UserName	UserAge
1	鈴木	21
2	佐藤	25
3	田中	18
4	山田	24
6	戦場ヶ原	17

演習

02 .田中さんの歳を19歳に変更せよ。

03 .戦場ヶ原さんを削除せよ。

04. 生年月日のカラム（カラム名:Birth）を追加せよ。

05. 以下のようにbirthdayカラムを作成し生年月日を追加せよ。

鈴木：1992-12-2

佐藤：1988-3-9

田中：1995-4-27

山田：1989-9-29

06. 歳が若い順に、名前を表示せよ。

07. レコードの数を数えよ。

08. 登録してあるユーザの歳の平均を表示せよ。

09. ユーザ名が”山”で始まるレコードを表示せよ。

内部結合

表の重複をなくす

- 表 employee

id	Name	department
1	松下	総務
2	田中	総務
3	高橋	営業
4	吉原	総務

繰り返しがあり
データの重複があ
り保守も大変

正規化（このようなデータベースをRDB）

id	Name	department
1	松下	総務
2	田中	総務
3	高橋	営業
4	吉原	総務

営業を1総務を3とする

テーブルを分割する→正規化

department_idでつな
げる

employee

id	Name	department_id
1	松下	3
2	田中	3
3	高橋	1
4	吉原	3

department

department_i d	department
1	営業
2	開発
3	総務
4	秘書

内部結合SQLの書き方

SELECT カラム名1, カラム名2, ...

FROM テーブル名 1 **INNER JOIN** テーブル
名2 **ON** 結合の条件

(重複するカラム名があるのでテーブル名.カ
ラム名で表現する)

内部結合の例

```
select * from employee
```

```
inner join
```

```
department on employee.department_id  
=department.department_id
```

(すべてでなくテーブル名.カラムで個別で表すことができる)

※すべてのフィールドではなく `employee.Name` で名前だけ表示)

課題

- 以下のテーブルを正規化して内部結合してテーブルを表示せよ
- テーブル名 成績表

学生番号	名前	科目名	成績
0001	田中	国語	80
0002	佐藤	数学	90
0003	馬場	国語	65

SQL(データ部)

```
create table
```

```
科目表(科目番号,  
科目);
```

```
create table
```

```
成績表(学生番号,  
名前,  
科目番号,  
成績);
```

```
insert into 科目表(科目番号,科目) values('001','国語');
```

```
insert into 科目表(科目番号,科目) values('002','数学');
```

```
insert into 科目表(科目番号,科目) values('003','理科');
```

```
insert into 成績表(学生番号,名前,科目番号,成績) values('001','田中','001',80);
```

```
insert into 成績表(学生番号,名前,科目番号,成績) values('002','佐藤','002',90);
```

```
insert into 成績表(学生番号,名前,科目番号,成績) values('003','馬場','001',65);
```

学生番号	名前	科目名	成績
0001	田中	国語	80
0002	佐藤	数学	90
0003	馬場	国語	65



学生番号	名前	科目番号	成績
0001	田中	001	80
0002	佐藤	002	90
0003	馬場	001	65



科目番号	科目
001	国語
002	数学
003	理科

外部結合

学生番号	名前	科目番号	成績	
001	田中	001	80	
002	佐藤	002	90	
003	馬場	001	65	
004	松下	004	70	

科目番号	科目
001	国語
002	数学
003	理科

対応する科目番号004がない
内部結合だと学生番号004は表示されない
inner join を **left outer join** にすると学生番号004は
表示される。これを外部結合という

外部結合（左を残すleft outer join）

```
select * from 成績表 inner OUTER join 科目表 on 成績表.科目番号=科目表.科目番号;
```



```
select * from 成績表 left OUTER join 科目表 on 成績表.科目番号=科目表.科目番号;
```

ビュー

- 仮想の表（テーブル）
- 例えば 1 8 0 c m以上の身長 of 選手名をテーブルとして作成したい場合

giants



ビュー



select文で作
これが1つのテーブルとして扱える

ビューの作り方

構文

`create view ビュー名 as select`文

例

問題 身長 1 8 0 c m以上の選手のビューを作ってください

```
create view player_hight180 as select * from giants where 身長 >= 180
```

select * from player_hight180で見れる

ここで重要なのはplayer_hight180が実際にテーブルがないということを確認してください

確認

New Database

Open Database

Write Changes

Revert Changes

Open Project

Save Project

Attach

Database Structure

Browse Data

Edit Pragmas

Execute SQL

Create Table

Create Index

Print

Name	Type	Schema
▼ Tables (4)		
> department		CREATE TABLE department (department_id, department)
> employee		CREATE TABLE employee (id, name, department_id)
> giants		CREATE TABLE "giants" ("選手名" TEXT, "守備" TEXT, "生年月日" TEXT,
> sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
Indices (0)		
▼ Views (1)		
> player_hight180		CREATE VIEW player_hight180 as select * from giants where 身長>=1
Triggers (0)		

インデックス

- 高速に検索するためにテーブルのカラムにインデックスを付与する→高速に検索できる
(データ数百万件などでは発揮する)
→重複のないカラムに威力を発揮する
(性別など種類が少ない場合は負荷になる)
- ★ビックデータになればなるほどインデックスは重要になります。データベースのチューニングにはまず基本です

インデックス

CREATE INDEX インデックス名 **ON** テーブル名
(カラム名1, カラム名2, ...);

create index index_giants on giants(選手名);

.indicesで確かめる

```
(base) C:\Users\user\Google ドライブ\資料\code_data>sqlite3 giants.db
SQLite version 3.31.1 2020-01-27 19:55:54
Enter ".help" for usage hints.
sqlite> .indices
index_giants
sqlite>
```

index実習

Wikipedia.dbを開けます

select * from Wikipedia where titile='サンセール'を検索します
(時間が少しかかります)

Wikipedia.dbからテーブルwikipediaのフィールドtitileにインデックスを振ります

```
create index title_idx on wikipedia(title);
```

再びselect * from Wikipedia where titile='サンセール'を検索します。速く感じられるはずです

副問い合わせ

insert into テーブル名(フィールド・・・)
values (フィールド・・・)



insert into テーブル名(フィールド・・・)
select フィールド・・・ from テーブル名

selectで表示されるデータ

がテーブルに追加される (ただし、フィールドは完全に合わせる)

副問い合わせ例

```
create test1(f1 text,f2 text);  
insert into test1(f1,f2)values('testdata1','testdata1');  
insert into test1(f1,f2)values('testdata,2'testdata2');  
create test2(f1,f2);  
  
insert into test2(f1,f2)  
select f1,f2 from test1;
```

insert into test2(f1,f2)
select f1,f2 from test1

insert into test2(f1,f2)

f1	f2



select f1,f2 from test1

f1	f2
test1	test1
test2	test2



コピー

演習

- テーブルgiantsから身長180cm以上
体重80kg以上の選手を新たにテーブルを作成して
副問い合わせを利用してデータを入れて
ください。スキーマは各自任せます。

解答例

データ削除(delete文)

- delete from テーブル名 where フィールド = 値;

例

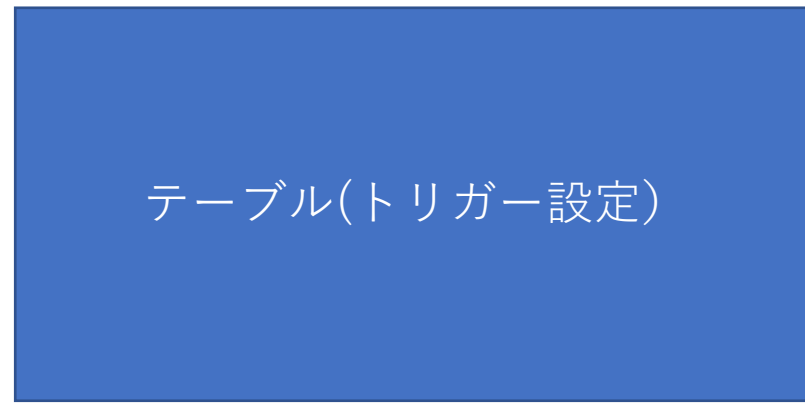
insert into giants(選手名) values('Test')

でデータを入れて確かめてください

select * from giants where 選手名='Test'で確かめてdelete 文を実行する

もし **where文を書かず delete from giantsを実行すればすべてのデータが消える。**

トリガー



(1) テーブルにinsert delet upate後または前にSQLが実行できる

トリガー構文

```
CREATE TRIGGER トリガー名  
INSERT ON テーブル名  
BEGIN  
    SQL文1;  
    SQL文2;  
END;
```

トリガー確認および削除

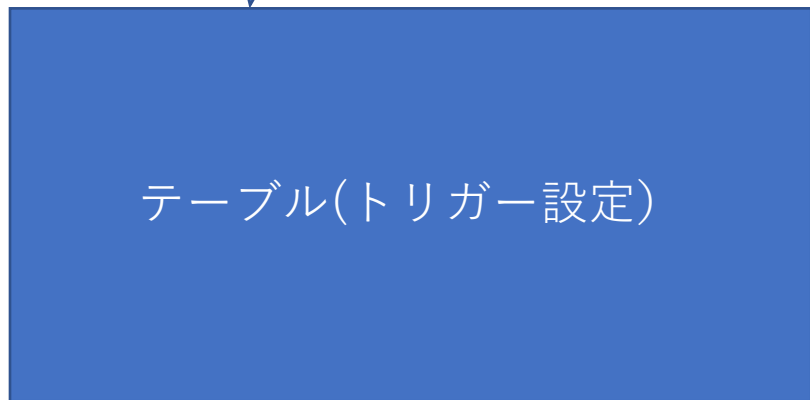
作成したトリガーの確認

```
SELECT * FROM sqlite_master WHERE type = 'trigger';
```

トリガーの削除

```
DROP trigger my_trigger;
```

具体例



この処理はトリガーで書かれているのでコーディングしなくてもいい

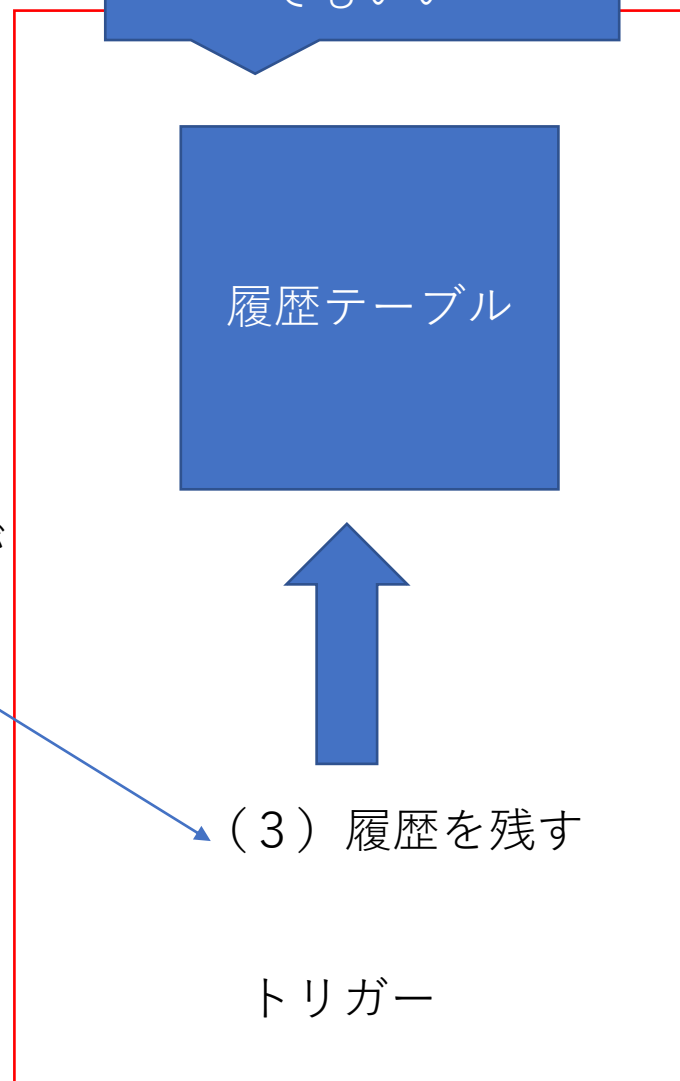


(2)トリガーが引かれる

(1)テーブルを追加した後

(3)履歴を残す

トリガー



例 以下 試してみる

```
create table product(id integer, name text, price integer);  
create table log(id integer primary key, act text);
```

```
create trigger itrigger insert on product  
begin  
insert into log(act) values('INSERT Action');  
end;
```

```
create trigger dtrigger delete on product  
begin  
insert into log(act) values('DELETE Action');  
end;
```

```
create trigger uttrigger update on product  
begin  
insert into log(act) values('UPDATE Action');  
end;
```

```
insert into product(id , name, price) values(12,'test',12);
```

データ更新

- Update テーブル名 set フィールド=値,
フィールド=値,
...
フィールド=値
where フィールド=値

例

- 笠井 駿|180|80の選手名を空白をなくした笠井駿にしてみる。

SQL

Update giants set 選手名='笠井駿' where 選手名='笠井 駿';

(※もし、where文がない場合どうなるか？

確かめてみてください)

Pythonによるデータベース操作

データ参照(selectdb.py)

```
import sqlite3
dbname='test.db'
conn=sqlite3.connect(dbname)
c = conn.cursor()
select_sql = "select No,name,height,weight from giants
"

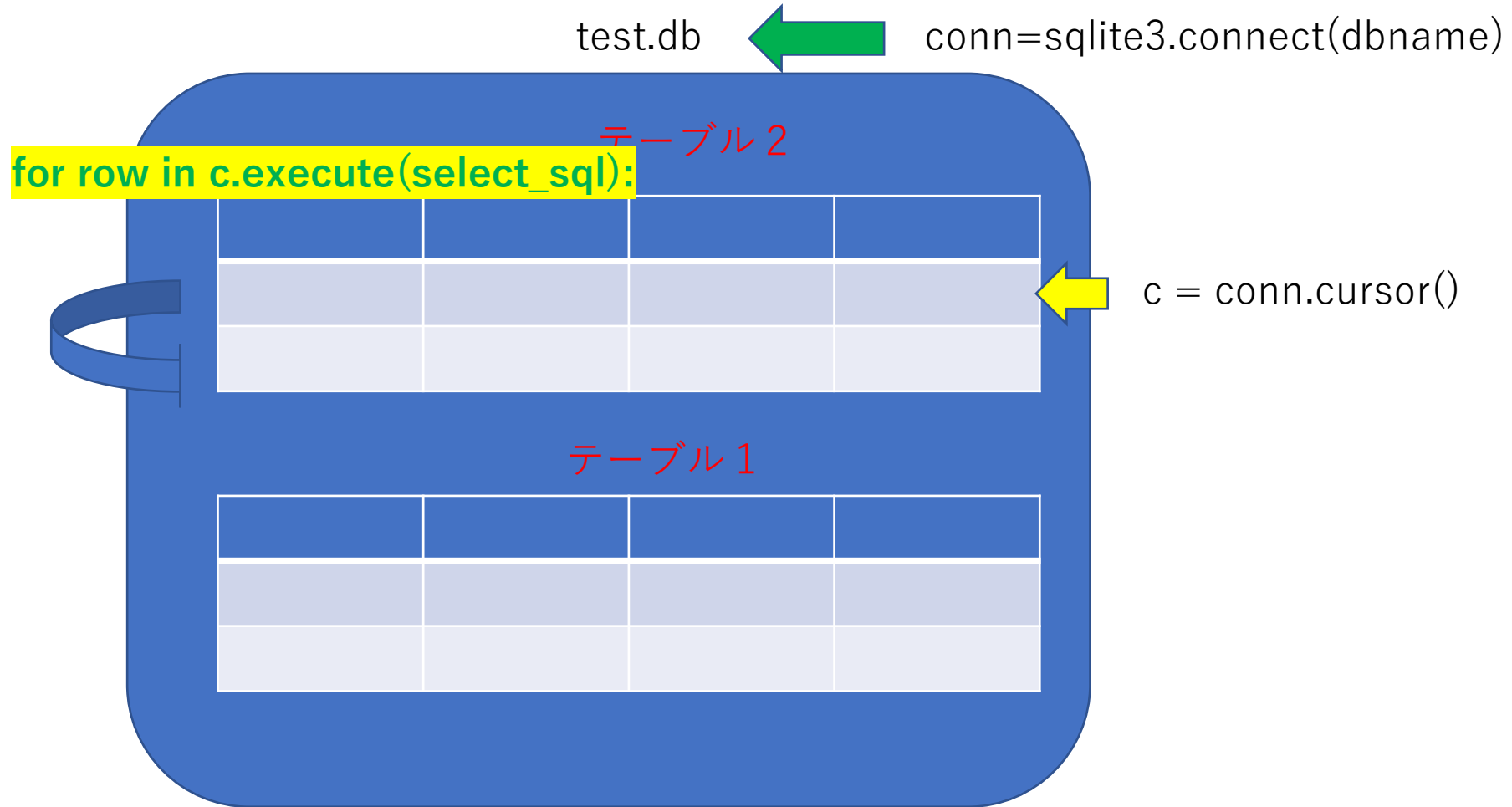
for row in c.execute(select_sql):
    print(row)
conn.close()
```

解説

```
import sqlite3←sqliteにアクセスできるライブラリをインポート  
dbname='test.db'←データベースの名前（この場合はプログラムと同じ位  
置にあるとしている）  
conn=sqlite3.connect(dbname)←データベースに接続
```

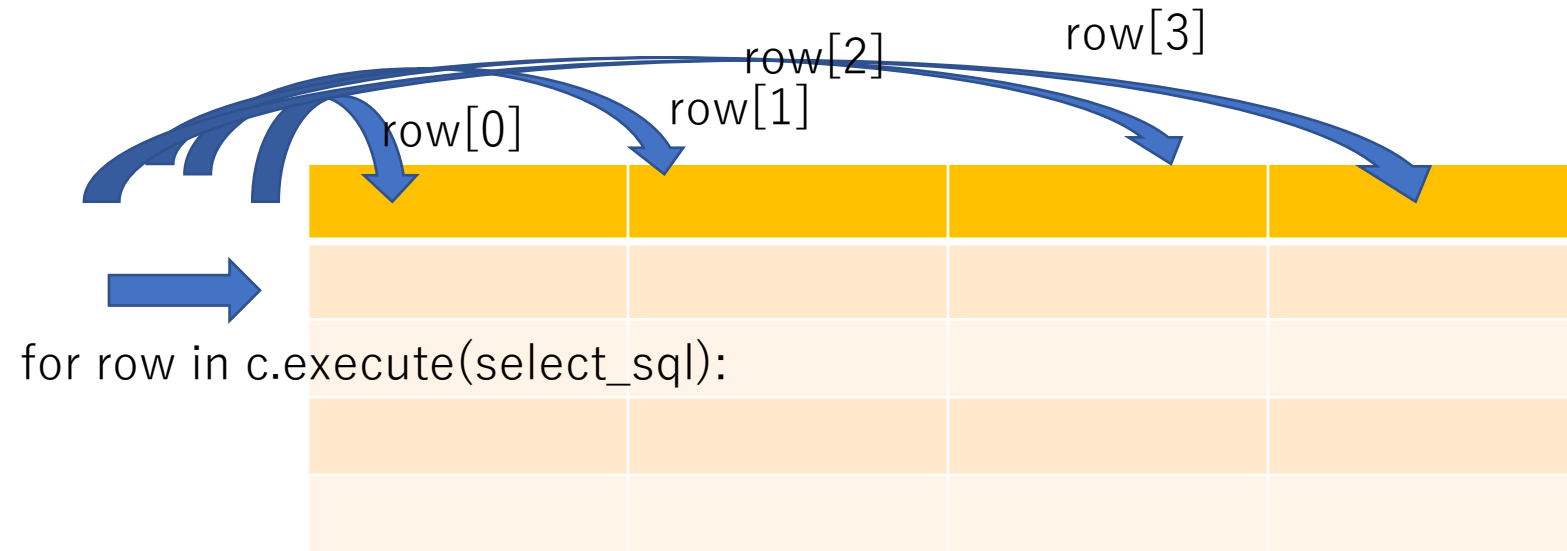
```
c = conn.cursor()←カーソルの設定
select_sql = "select No,name,height,weight from giants "←
実行する      SQLを書く
for row in c.execute(select_sql):←カーソルを実行（ループ）
    print(row)←カーソルの位置のデータを出力
conn.close()←接続を切断
```

図でまとめる



データ追加(insertdb.py)

```
import sqlite3
dbname='test.db'
conn=sqlite3.connect(dbname)
c = conn.cursor()
No='5'
name='Tset Name'
hight=170
weight=80
sql = 'INSERT INTO giants(No,name,hight,weight) VALUES(?,?,?,?)'
data = (No,name,hight,weight)
c.execute(sql, data)
conn.commit()
conn.close()
```

データの削除(deleted.py)

```
import sqlite3
dbname='test.db'
conn=sqlite3.connect(dbname)
c = conn.cursor()
No="5"
sql ="delete from giants where No=?"
data=(No)

c.execute(sql,data)
conn.commit()
conn.close()
```

データの削除(deleted2.py)

```
import sqlite3
dbname='test.db'
conn=sqlite3.connect(dbname)
c = conn.cursor()
sql ="delete from giants where No='4'"
c.execute(sql)
conn.commit()
conn.close()
```

No="5"



sql ="delete from giants where No=?"



c.execute(sql,data)

data=(No)



パラメータ
(?の部分)
はフィールド
のNoであるこ
と教える

pythonからCSVへのアクセス

```
import csv
with ('sample.csv') as f:
    reader = csv.reader(f)
    header = next(reader)    # ヘッダーを読み飛ばしたい
                              時
    for row in reader:
        print(row)
```

演習

- wikipediaのデータをテーブルに入れるようにしてください。

トランザクション処理

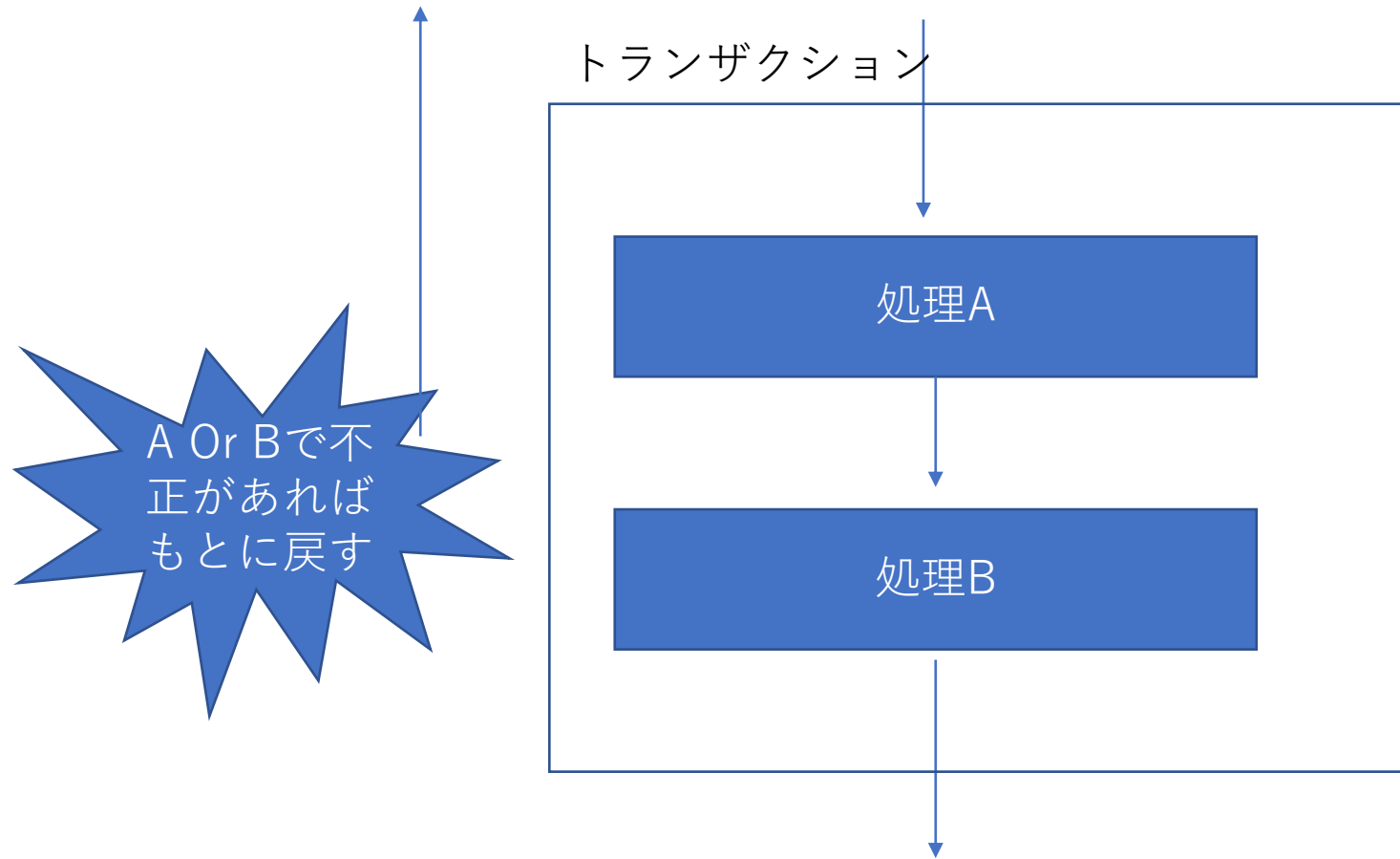
(1) Aさんの口座から5,000円分差し引き、残高 $10,000 - 5,000 = 5,000$ 円にする。

(2) Bさんの口座へ5,000円分プラスし、残高 $20,000 + 5,000 = 25,000$ 円にする。

もし(1)の処理ができなくなれば

Bさんの口座にお金だけ増える。Aさんは何もせずBさんにお金を振り込める。

トランザクション処理 (ALL or Nothing)



コード例A

```
import sqlite3
dbname='giants.db'
try:
    conn = sqlite3.connect(dbname)
    for i in range(5):
        print(i)
        conn.execute("insert into giants(選手名) values ('テスト')")
except sqlite3.Error as e:
    print(e)
    if conn: conn.rollback()
    print("rollback run")
finally:
    if conn: conn.commit()
```

コード例B

```
import sqlite3
dbname='giants.db'
try:
    conn = sqlite3.connect(dbname)
    for i in range(5):
        print(i)
        if i==2:
            conn.execute("xxxinsert into giants(選手名) values ('テスト')")

    else:
        conn.execute("insert into giants(選手名) values ('テスト')")
except sqlite3.Error as e:
    print(e)
    if conn: conn.rollback()
    print("rollback run")
finally:
    if conn: conn.commit()
```

わざとエラーのSQLを作る