データベース

1. データベースとSQLite

データベースとは

- システムは、通常、何らかのデータを管理している。システムが管理しているデータとしては顧客情報や商品の売上などがあるが、システムにはこうしたデータが大量に集められている。これらのデータを正しく効率的に利用するためにはデータを集中的に管理し、きちんと保守をする必要がある。こうしたニーズを満たすために考えだされた仕組みがデータベースである。
- 本来データベースはデータの集合を指す言葉であって、形のない概念的なものである。その概念的なデータベースをソフトウェアで操作・管理できるようにしたものをデータベース管理システム(DBMS:Data Base Management System)という。

データベースの役割

- データベース管理システム(DBMS)には次のような機能が備わっている。
  - データの管理
    データベースで取り扱うデータの形式を定義する。また、データの追加・削除・更新・検索機能も提供する。
  - トランザクション管理
    口座振替の入金処理と出金処理を必ずセット実行するなど、関連する複数の処理で矛盾が起こらないよう、データベース操作の一貫性を保証する。
  - 同時実行制御
    複数のユーザが同時にデータを操作しても、データに不整合が生じないように制御する。

データベースの役割

- セキュリティ機構
  データベースが権限のないユーザによってアクセスされ、データベース内に格納されていた機密情報などが漏洩すると、重大な問題となる。データベースには、このような不正アクセスを防止するセキュリティ機構が備えられている。
- 障害回復管理
  コンピュータが故障するなど、データベースに何らかの障害が発生した場合でも、データベース内のデータを障害が発生する前の状態に復元できる機能が備えられている。

リレーショナルデータベース

- データベースはさまざまな構造のものが研究されてきたが、現在、広く使われているのは「リレーショナルデータベース（RDB）」と呼ばれるものとなっている。
- リレーショナルデータベースでは、データを表計算ソフトの表のような形式で管理している。このデータを入れた表を単に表（テーブル）という。表は、たとえば、一人分の会員データを、会員番号、氏名、住所、という項目に分けて、これを人数分集める。一件分のデータが横方向の行に並び、そして、会員番号といった種類のデータが縦方向の列に並んでいる。*→ 次ページの表の例参照*
- 一件一件のデータが入っている行を単に行（レコード）といい、項目毎の値が入っている縦の列を単に列（カラム）という。

リレーショナルデータベース

- 表の例

会員番号	氏　　名	住　　所	・・・
Z10234	佐藤翼	東京都港区	・・・
Z10235	田中翔太	千葉県船橋市	・・・
Z10237	山田陸	神奈川県横浜市	・・・
Z10459	山田太一	・・・	・・・
・・・	・・・	・・・	・・・

代表的なデータベース製品

- Oracle Database
- IBM Db2 Database
- Microsoft SQL Server
- Microsoft Access
- MySQL
- PostgreSQL
- SQLite

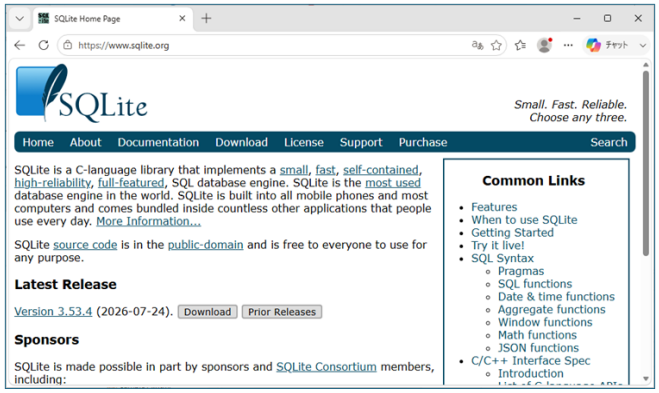
SQLite

- SQLiteはリレーショナルデータベースのソフトウェア。小規模なシステムなどで広く活用されている。
- 特徴
  - データベースの準備が不要でアプリに組み込める
    サーバ構築が不要で、アプリケーション内に組み込めるため、セットアップやメンテナンスが簡単
  - マルチプラットフォームに対応
    Windows、macOS、Linux、Android、iOSなど、さまざまなプラットフォームで利用できる
  - 軽くて速い
    ファイル形式であり、メモリ消費量が少ないため、リソースの少ない環境でも高速に動作する。
  - パブリックドメイン
    著作権が放棄されており、自由に利用できる。

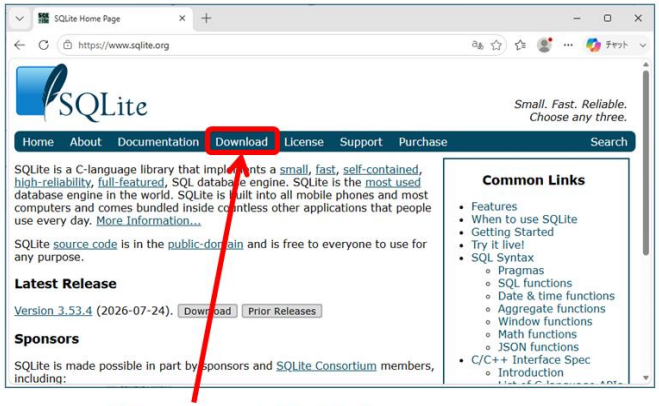
SQLite

●SQLite（ライブラリおよびコマンド）は公式ホームページからダウンロードして使うことができる。

https://www.sqlite.org

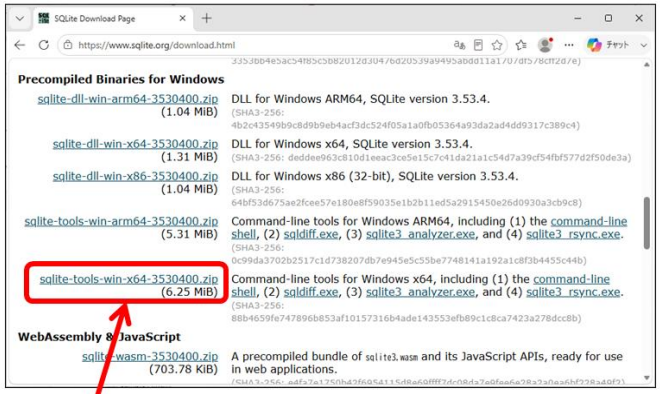


今回使用するコマンドのダウンロード(参考)



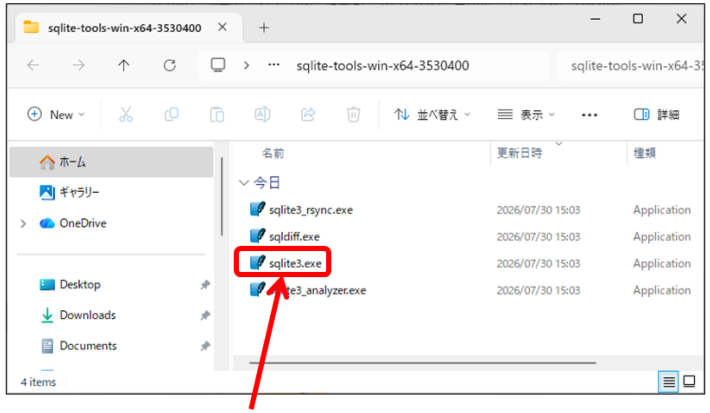
「Download」をクリック

今回使用するコマンドのダウンロード(参考)



sqlite-tools-win-x64-3530400.zip をクリックしてダウンロード

今回使用するコマンドのダウンロード(参考)



ダウンロードしたzipファイルを展開して、sqlite3.exeを
C:¥windows(などパスの通ったところ) にコピー

データベース

2. データベースの接続

SQLiteへの接続

- SQLiteは1つのデータベースが1ファイルで構成される（ファイルを使わずにメモリー上にも作成も可能）。
- SQLiteに接続するにはsqlite3コマンドを使いファイル名を指定する。ファイルが存在しない場合には新規作成する。（インターン生用PCにはsqlite3コマンドはダウンロードしてインストール済み）

```
sqlite3 ファイル名
```

- SQLiteにはユーザーの概念がない。したがって、ユーザーごとの権限設定を行うことができない。
- 表名や列名は大文字と小文字を区別しない。

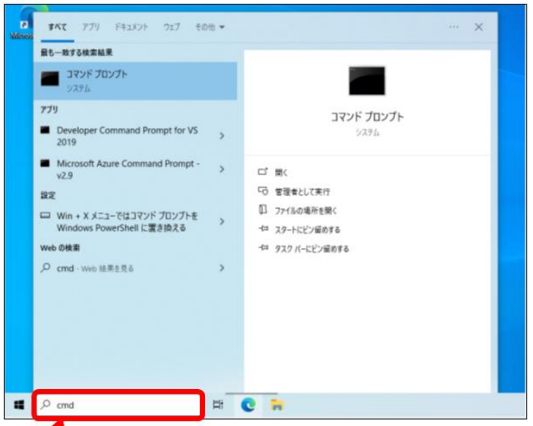
SQLiteへの接続

- sqlite3の基本的なコマンドには次がある。

書式	説明
.databases	データベースの一覧を表示
.tables [パターン]	テーブル名を表示
.exit	プログラムを終了する。
.quit	プログラムを終了する。
.help	ヘルプを表示する。

SQLiteへの接続

- sqlite3コマンドを実行するためにコマンドプロンプトを起動する。



検索ボックスに「cmd」と入れて[Enter]キーを押す

データベースの接続

```
C:¥Users¥jcreation> sqlite3 test.db     ← DB接続
SQLite version 3.53.4 2026-07-24 19:02:57
Enter ".help" for usage hints.

sqlite> .databases     ← 接続DBの確認
main: C:¥Users¥jcreation¥test.db r/w

sqlite> .quit     ← 接続解除
```

注. フォントや設定により「￥」は「＼」と表示される。

データベース

3. 表の作成・挿入・抽出

SQLとは

- データベースを操作するとき、ユーザはデータベースに対して命令を出す。この命令を文字として表したのがクエリ(Query)である。クエリは日本語で「問い合わせ」と訳される。たとえば、表を作るときは「CREATE TABLE…」、データを挿入するときは「INSERT INTO…」というクエリを使う。そして、クエリを書く規則がSQL(Structured Query Language)という言語となる。SQLの規則で作成した命令の記述をSQL文と呼ぶ。
- SQL文は最後にセミコロン「;」を付けて実行する。

```
> SQL文 ;
```

- SQLでは大文字・小文字を区別しない。一般的にはキーワードには大文字が使われる習慣がある。

表の作成

- 表(テーブル)を作成するときには次の構文に従って記述する。

```
CREATE TABLE 表名 (列名1 データ型1,
                   列名2 データ型2, …);
```

CREATE TABLEの後に表名(表の名前)、そして括弧()内に列名(列の名前)とデータ型を記述する。一列ごとに「列名　データ型」のようにスペースで間をあけて記述する。列ごとの記述はカンマ「,」で区切る。

- 各列にはデータ型を指定する。列に入力できるのは指定されているデータ型の値のみとなる。ただし、SQLiteではデータ型の指定は省略可能で、この場合どのような値でも入力できる。

基本的なデータ型

●SQLiteで列に指定できるデータ型には次がある。

型名	説明	一般的なSQLの型
TEXT	文字列	「CHAR」、「CLOB」、「TEXT」という文字列を含むデータ型
INTEGER	整数	「INT」という文字列を含むデータ型
REAL	浮動小数点数	「REAL」、「FLOA」、「DOUB」という文字列を含む
NUMERIC	数値	NUMERIC、DECIMAL、BOOLEAN、DATE、DATETIME
NONE	入力データをそのまま格納	「BLOB」という文字列を含む、または型の指定がない

表の作成

●ここでは、次のような3つの列を持つ表tb1を作成する。列codeとnameには文字列が入り、列ageには整数が入る。

列名	code	name	age
データ型	TEXT	TEXT	INTEGER
具体的な内容	社員コード	氏名	年齢

●表の一覧を表示するには次のようにする。

```
.tables;
```

●表の構造は次のようにして確認できる。

```
.schema [表名前]
```


表の作成

●表を作成して、表の一覧と、テーブルの構造を表示する例は次のようになる。

```
C:¥Users¥jcreation> sqlite3 test.db    ← DB接続

> CREATE TABLE tb1(code TEXT, name TEXT, age INTEGER);

> .tables
tb1

> .schema tb1
CREATE TABLE tb1(code TEXT, name TEXT, age INTEGER);
```


表にデータを挿入する

●表tb1は作成したが、データはまだ何も入っていない。データを挿入するときは次の形式のINSERTコマンドを使う。

```
INSERT INTO 表名 VALUES(列1のデータ,列2のデータ,…);
```

●データの挿入では、列の順番に関係なく、列名を指定して挿入することもできる。実際次のように指定する。

```
INSERT INTO 表名(列名A,列名B,…)
       VALUES(列Aのデータ,列Bのデータ,…);
```

●入力するSQL文が長くなると、1行で見やすく表示するのは難しくなってくる。SQL文はセミコロン「;」で終わりとなるので、見やすく改行し、適宜スペースなどを入れて記述しても構わない。

表にデータを挿入する

●先ほど作成した表tb1に次のデータを挿入していく。

code	name	age
E01	中村	40
E02	山田	23
E03	中川	20
E04	田中	28
E05	西沢	35

●列に入力する数値データはそのまま記述するが、文字列データはシングルクォート「' 」で囲って記述する。

表にデータを挿入する

●表tb1にデータを挿入する例は次のようになる。

```
> INSERT INTO tb1 VALUES('E01', '中村', 40);
> INSERT INTO tb1 VALUES('E02', '山田', 23);
> INSERT INTO tb1 VALUES('E03', '中川', 20);
> INSERT INTO tb1(age,name,code) VALUES(28,'田中','E04');
> INSERT INTO tb1
...>         VALUES(
...>                 'E05',
...>                 '西沢',
...>                 35
...>         );
```

コマンドプロンプトでは［↑］キーや［↓］キーで実行コマンドの履歴を辿れる

表データの表示

●表のデータを表示するには「SELECT」コマンドを使う。複数の列を表示させる場合、列の間をカンマ「,」で区切って指定する。

```
SELECT 列名1, 列名2, … FROM 表名;
```

●すべての列を表示するには列名にアスタリスク「*」を指定することもできる。

```
SELECT * FROM 表名;
```


表データの表示

●表tb1から列名を表示するには次のようにする。

```
> SELECT name FROM tb1;
中村
山田
中川
田中
西沢

> SELECT code,name FROM tb1;
E01|中村
E02|山田
E03|中川
E04|田中
E05|西沢
```

表データの表示

●表tb1からすべての列の値を表示するには次のようにする。

```
> SELECT * FROM tb1;
E01|中村|40
E02|山田|23
E03|中川|20
E04|田中|28
E05|西沢|35
```


問題1

① 次のような3つの列を持つ表table1を作成せよ。

列名	code	name	addr
データ型	INTEGER	TEXT	TEXT
具体的な内容	コード	名前	住所

② 表table1に次のデータを挿入せよ。

code	name	addr
101	中村	東京
102	山田	埼玉
103	中川	千葉

③ 表table1の内容を一覧表示して確認せよ。

問題1(解答)

① 次のような3つの列を持つ表table1を作成せよ。

列名	code	name	addr
データ型	INTEGER	TEXT	TEXT
具体的な内容	コード	名前	住所

```
CREATE TABLE table1 (
  code INTEGER,
  name TEXT,
  addr TEXT
);
```


問題1(解答)

② 表table1に次のデータを挿入せよ。

code	name	addr
101	中村	東京
102	山田	埼玉
103	中川	千葉

```
INSERT INTO table1 VALUES(101,'中村','東京');
INSERT INTO table1 VALUES(102,'山田','埼玉');
INSERT INTO table1 VALUES(103,'中川','千葉');
```

③ 表table1の内容を一覧表示して確認せよ。

```
SELECT * FROM table1;
```

WHEREを使った抽出

●条件に一致する行だけをSELECTで表示する場合にはWHEREを用いる。

```
SELECT 列名 FROM 表名 WHERE 条件;
```

条件は次の比較演算子を使って指定する。

演算子	意味
=	等しい
>	より大きい
>=	以上
<	より小さい
<=	以下
!= または <>	等しくない

WHEREを使った抽出

●表tb1において列ageの値が30以上の行のみを表示するには次のようにする。

```
> SELECT * FROM tb1 WHERE age >= 30;
code  name  age
----  ----  ---
E01   中村    40
E05   西沢    35
```


WHEREを使った抽出

●比較演算子には次のようなものもある。ここで「リスト」とは値の一覧を意味し、直接記述する場合には「(値1,値2,値3,…)」などと表記する。

演算子	意味
a IN b	bのリスト中にaがあるかどうか
a NOT IN b	bのリスト中にaがないかどうか
a BETWEEN b AND c	aがb以上c以下の間にあるかどうか
a NOT BETWEEN b AND c	aがb以上c以下の間にないかどうか

WHEREを使った抽出

●たとえば、表tb1の列nameの値が「山田」か「田中」のものだけを表示するには次のようにする。

```
> SELECT * FROM tb1 WHERE name in ('山田', '田中');
code  name  age
----  ----  ---
E02   山田    23
E04   田中    28

> SELECT * FROM tb1 WHERE age BETWEEN 25 AND 35;
code  name  age
----  ----  ---
E04   田中    28
E05   西沢    35
```


文字列を使った条件

●文字列を対象にした条件設定で、文字列が等しいかどうかには数値と同様に「=」を使う。また、文字列が等しくないかどうかには「!=」または「<>」を使う。

```
> SELECT * FROM tb1 WHERE code = 'E01';
code  name  age
----  ----  ---
E01   中村    40
> SELECT * FROM tb1 WHERE code != 'E01';
code  name  age
----  ----  ---
E02   山田    23
E03   中川    20
E04   田中    28
E05   西沢    35
```


LIKEによる曖昧検索

●「LIKE演算子」を使うと、ぴったりと条件に一致したものではなく、部分一致による曖昧検索が可能となる。このときワイルドカードを使って曖昧な部分を指定する。

●ワイルドカードには次の文字を使う。

文字	意味
%	任意の長さ（ゼロを含む）の文字列
_	任意の1文字

LIKEによる曖昧検索

●ワイルドカードの使用例

指定例	意　味	該当する例
%県	'県'で終わる文字列	神奈川県、県
福%	'福'で始まる文字列	福井県、福
%県%	'県'を含む文字列	都県名、埼玉県
長_県	長と県の間に一文字入る文字列	長野県、長崎県

●LIKEによる曖昧検索の構文は次のようになる。

```
SELECT 列名 FROM 表名 WHERE 条件列名 LIKE パターン;
```

LIKEによる曖昧検索

●次の例では表tb1中の列nameに文字「中」を含む行だけを表示している。

```
> SELECT * FROM tb1 WHERE name LIKE '%中%';
code  name  age
----  ----  ---
E01   中村    40
E03   中川    20
E04   田中    28
```


NOT LIKEによる曖昧検索

●LIKEの否定は「NOT LIKE」となる。

●次の例では表tb1中の列nameに文字「中」を含まない行だけを表示している。

```
> SELECT * FROM tb1 WHERE name NOT LIKE '%中%';
code  name  age
----  ----  ---
E02   山田    23
E05   西沢    35
```


NULLを使った条件

●NULLは「何もない値」あるいは「不明な値」を意味する。列にデータを入れて行を挿入し、初期値も設定してなければNULLが入ることになる。

●テスト用に表tb1に、列nameだけ'山本'に設定した文字列を入力しておく。

```
> INSERT INTO tb1(name) VALUES('山本');
> SELECT * FROM tb1;
code  name  age
----  ----  ---
E01   中村    40
E02   山田    23
E03   中川    20
E04   田中    28
E05   西沢    35
NULL  山本  NULL
```


NULLを使った条件

●列の値がNULLである行を抽出するときは「IS NULL」を使う。また、列の値がNULLでない行を抽出するときは「IS NOT NULL」を使う。「=」での比較はうまくいかない。

```
> SELECT * FROM tb1 WHERE age IS NULL;
code  name  age
----  ----  ---
NULL  山本  NULL
> SELECT * FROM tb1 WHERE age IS NOT NULL;
code  name  age
----  ----  ---
E01   中村    40
E02   山田    23
E03   中川    20
E04   田中    28
E05   西沢    35
```

複数条件の指定

- SELECT 文に指定するWHERE 句などの条件で、複数の条件を設定したい場合にはAND 演算子、OR 演算子が使用できる。
- AND 演算子は指定した条件が両方満たされる場合、OR 演算子は指定した条件のいずれかが満たされる場合にSQL 文の結果が表示される。
- NOTを使うと条件の否定ができる。

複数条件の指定

```
> SELECT * FROM tb1 WHERE age > 25 AND age <= 35;
code  name  age
----  ----  ---
E04   田中    28
E05   西沢    35
> SELECT * FROM tb1 WHERE age <= 25 OR age > 35;
code  name  age
----  ----  ---
E01   中村    40
E02   山田    23
E03   中川    20
> SELECT * FROM tb1 WHERE NOT(age > 25 AND age <= 35);
code  name  age
----  ----  ---
E01   中村    40
E02   山田    23
E03   中川    20
```


並べ替えて表示

- 行を指定した列の値の順に表示させるにはORDER BYを使う。
- 行を昇順に整列して表示するには次のようにする。最後のASC(Ascending)はつけなくてもよい。

```
SELECT 列名 FROM 表名 ORDER BY キーとなる列 [ASC];
```

- 行を降順に整列して表示するには次のようにする。最後にはDESC(Descending)を付ける。

```
SELECT 列名 FROM 表名 ORDER BY キーとなる列 DESC;
```

- 並べ替えは文字コードを基準に行われるので、日本語の並べ替えを行うにはふりがなの列などを用意する必要がある。

並べ替えて表示

```
> SELECT * FROM tb1 ORDER BY age;
code  name  age
----  ----  ---
      山本
E03   中川    20
E02   山田    23
E04   田中    28
E05   西沢    35
E01   中村    40
> SELECT * FROM tb1 ORDER BY age DESC;
code  name  age
----  ----  ---
E01   中村    40
E05   西沢    35
E04   田中    28
E02   山田    23
E03   中川    20
      山本
```

問題2

① 表table1の列codeの値が102以上の行のみを表示せよ。
② 表table1の列codeの値が101以上、102以下の行のみを表示せよ。
③ 表table1の列codeの値が101または103の行のみを表示せよ。
④ 表table1の列nameの値が「中」で始まる行のみを表示せよ。
⑤ 表table1に、列codeの値が「104」、列nameの値が「中村」の行を追加せよ(addrは指定しない)。
⑥ 表table1の列addrの値がNULLである行を表示せよ。

問題2(解答)

① 表table1の列codeの値が102以上の行のみを表示せよ。
```
SELECT * FROM table1 WHERE code >= 102;
```
② 表table1の列codeの値が101以上、102以下の行のみを表示せよ。
```
SELECT * FROM table1
        WHERE code BETWEEN 101 AND 102;
SELECT * FROM table1
        WHERE code >= 101 AND code <= 102;
```
③ 表table1の列codeの値が101または103の行のみを表示せよ。
```
SELECT * FROM table1 WHERE code IN (101,103);
SELECT * FROM table1 WHERE code = 101 OR code = 103;
```
④ 表table1の列nameの値が「中」で始まる行のみを表示せよ。
```
SELECT * FROM table1 WHERE name LIKE '中%';
```


問題2(解答)

⑤ 表table1に、列codeの値が「104」、列nameの値が「中村」の行を追加せよ(addrは指定しない)。
```
INSERT INTO table1(code,name) VALUES(104,'中村');
```
⑥ 表table1の列addrの値がNULLである行を表示せよ。
```
SELECT * FROM table1 WHERE addr IS NULL;
```


データベース

5. 更新と削除

列データの更新

●データを更新するときにはUPDATEコマンドを使う。条件を指定すれば該当行が、条件を指定しなければすべての行が更新される。

```
UPDATE 表名 SET 列名 = 値 WHERE 条件;
```

```
> UPDATE tb1 SET age = 24 WHERE name = '山田';
> SELECT * FROM tb1;
code  name  age
----  ----  ---
E01   中村    40
E02   山田    24   ← 23から24に更新
E03   中川    20
E04   田中    28
E05   西沢    35
      山本
```


行の削除

●表自体は削除せずに、表に保管されている行を削除するには次のようにする。条件を指定すれば該当行が、条件を指定しなければすべての行が削除される。

```
DELETE FROM 表名 WHERE 条件;
```

●表tb1から列nameの値が'山本'の行を削除する。

```
> DELETE FROM tb1 WHERE name = '山本';
> SELECT * FROM tb1;
code  name  age
----  ----  ---
E01   中村    40
E02   山田    24
E03   中川    20
E04   田中    28
E05   西沢    35
```


表の削除

●表を削除するにはDROP TABLEを使う。このコマンドは通常はROLLBACKで元に戻すことができない。

```
DROP TABLE 表名;
```

●表tb1を削除して、DESCコマンドにより存在を確認してみる。

```
> DROP TABLE tb1;   ← 表tb1の削除
> .tables           ← 表の一覧表示
table1              ← tb1は表示されない
> SELECT * FROM tb1;
Parse error: no such table: tb1.  ← 存在しないためエラー
```


問題3

① 表table1において、列codeの値が103の行のaddrを「神奈川」に変更せよ。

② 表table1において、列addrの値がNULLのものを「不明」に変更せよ。

③ 表table1の内容を一覧表示せよ。

④ 表table1において、列codeの値が103以上の行を削除せよ。

⑤ 表table1を削除せよ。

問題3(解答)

① 表table1において、列codeの値が103の行のaddrを「神奈川」に変更せよ。
```
UPDATE table1 SET addr = '神奈川' WHERE code = 103;
```
② 表table1において、列addrの値がNULLのものを「不明」に変更せよ。
```
UPDATE table1 SET addr = '不明' WHERE addr IS NULL;
```
③ 表table1の内容を一覧表示せよ。
```
SELECT * FROM table1;
```
④ 表table1において、列codeの値が103以上の行を削除せよ。
```
DELETE FROM table1 WHERE code >= 103;
```
⑤ 表table1を削除せよ。
```
DROP TABLE table1;
```


データベース

6. 主キー

主キー

●「主キー」あるいは「プライマリーキー」とは、表において行を一意に特定するために使われる情報で、つぎの要件を満たさなければならない:
- ➢未入力は不可(NOT NULL)
- ➢値の重複がない(UNIQUE)
- ➢1組しか設定できない

●表の作成時に主キーを設定する場合、列定義の直後に指定する書式と、列定義の最後に指定する書式がある。

●SQLiteでは、主キーの型がINTEGERのとき、主キーに値を指定しない場合には、自動的に値が設定される。また、AUTOINCREMENTキーワードを付け足すと、その値は常に最大値が使用される(指定しないと空いている番号が使い回される)。

主キー

●列定義の直後に指定

```
CREATE TABLE 表名(
   列名1 INTEGER PRIMARY KEY [AUTOINCREMENT],
   列の定義2,
   ・・・・・
);
```

主キー

```
> CREATE TABLE tb2(
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT);
> INSERT INTO tb2(name) VALUES('渡辺');
> INSERT INTO tb2(name) VALUES('高橋');
> SELECT * FROM tb2;
id  name
--  ----
1   渡辺
2   高橋
> INSERT INTO tb2(id, name) VALUES(1, '伊藤');
Error near line 56: UNIQUE constraint failed: tb2.id
```

id列に値を指定せずに行を挿入

id列に値が設定されている

主キーには重複した値が挿入できない

問題4

① 次の表table2を作成せよ。ここで、列idは主キーとせよ。

列名	id	name	age
データ型	INTEGER	TEXT	INTEGER
具体的な内容	主キー	社員名	年齢

問題4

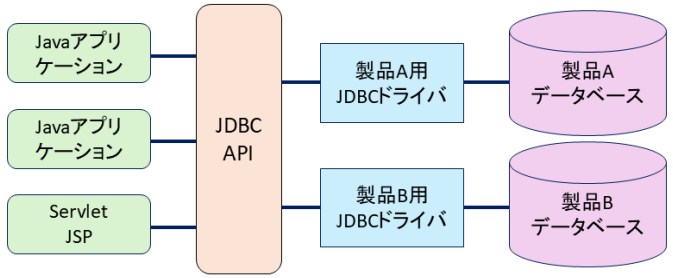
① 次の表table2を作成せよ。ここで、列idは主キーとせよ。

列名	id	name	age
データ型	INTEGER	TEXT	INTEGER
具体的な内容	主キー	社員名	年齢

```
CREATE TABLE table2(id INTEGER PRIMARY KEY,
                    name TEXT,
                    age INTEGER);
```


JDBC

●JavaプログラムからさまざまなデータベースにアクセスするためのインタフェースやクラスとしてJDBC APIが用意されている。JDBC APIを使用することで、データベース製品に依存しないJavaプログラムを作成できる。



JDBCを使用する手順

●JDBCを使用するときの一般的な処理の流れは次のようになる。

① java.sqlパッケージのインポート

② データベースの指定(SQLiteの場合)

```
String url = "jdbc:sqlite:DBファイルのパス";
```

③ データベースとの接続

```
Connection con = DriverManager.getConnection(url)
```

④ ステートメントの取得(引数にSQL文を指定)

```
PreparedStatement ps = con.prepareStatement(SQL文)
```

⑤ SQL文の実行

```
int rows = ps.executeUpdate()    //SELECT以外
ResultSet rs = ps.executeQuery() //SELECT
```

⑥ SELECTの場合は結果の取得

```
while(rs.next()){  /* getXXX()で列の値を取得 */  }
```

⑦ 接続のクローズ

```
con.close();
ps.close();
```


JDBCを使用する手順

① java.sqlパッケージの Connection, DriverManager, ResultSet, SQLException, Statement をインポートする。

④ PreparedStatementを使う場合はprepareStatement()メソッドの引数にSQL文を指定する。ここで、SQL文の中にクエスチョンマーク(?)の形で値を仮置きできる。

```
String sql = "SELECT * FROM tb2 WHERE id = ?";
PreparedStatement ps = con.prepareStatement(sql);
```

仮置きの"?"は実行前に、setXXX()メソッドを使って実際の値を割り当てる。第1引数には何個目の?かの位置を1,2,…という数え方で指定し、第2引数には置き換える値を指定する。
値が文字列の場合はsetString()メソッドを使い、整数の場合はsetInt()を使う

例.1個目の？を30で置き換える。

```
setInt(1, 30)
```


JDBCを使用する手順

⑥ SELECT文の結果は ResultSet オブジェクトに格納される。ここから値を取り出すには、next() メソッドを呼び出してカーソル(操作の対象行)を次の行(最初は開始行)に移動する。next() メソッドは結果があれば true を返しそうでなければ false を返す。カーソル行から列の値を取り出すには次のようなgetXXX() メソッドを使用する。getXXX() は引数に列番号または列名を指定して、列の値を取得する(列番号は1から数える)。

メソッド名	概要
int getInt(int index) int getInt(String label)	フィールドの値をJavaデータ型の int で取得する
String getString(int index) String getString(String label)	フィールドの値をJavaデータ型の String で取得する

JDBCを使用する手順

⑦ 全ての処理が終了したら、リソースを開放する。解放はResultSet、PreparedStatement、Connectionのそれぞれに用意されているclose()メソッドを利用する。Statementオブジェクトのclose()メソッドを呼ぶと、使用しているResultSetも一緒にクローズされるので、ResultSetのclose()は明示的にコールされないこともある。
Javaでは try-with-resources 文を使うと close() メソッドを自動的にコールするので明示的に記述する必要がない。
try(リソースを使うオブジェクト){ 処理 }

●データベースの接続ではさまざまな原因によりエラーが発生する可能性がある。これらのエラーは例外 SQLException とそのサブクラスとしてまとめられており、詳細は例外オブジェクトのメッセージで確認できる。

DBSample1.java

```
package db;
import java.sql.*;
public class DBSample1 {
   public static void main(String[] args) {
      String url = "jdbc:sqlite:c:/Users/jcreation/test.db";
      String sql = "SELECT * FROM tb2 WHERE id = ?";
      try( Connection con = DriverManager.getConnection(url);
           PreparedStatement ps = con.prepareStatement(sql) ){
              ps.setInt(1, 2); //追加(1個目の?を2に置き換え)
              ResultSet rs = ps.executeQuery(); //引数なし
              while( rs.next() ){
                 System.out.println(
                        rs.getInt(1) + "¥t" + rs.getString(2));
              }
      }catch(SQLException e){
         e.printStackTrace();
      }
   }
}
```

DBファイルのパス

実行結果

```
2	高橋
```


クラスパスにJDBCを追加

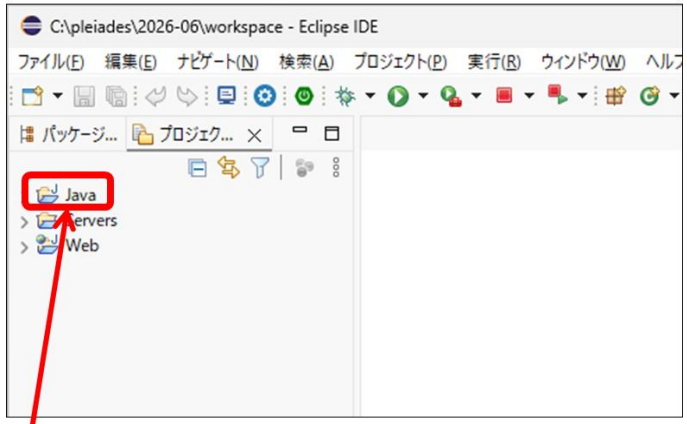
●アプリケーションを実行するにはクラスパスにJDBCのJARファイルを追加する必要がある。ここで、XXXはバージョン番号を表すものとする。
- sqlite-jdbc-XXX.jar

●今回のPCでは「C:¥pleiades」下にこれらのファイルを配置してある。

●Eclipseで実行する場合のクラスパスは、プロジェクトのプロパティから追加できる。

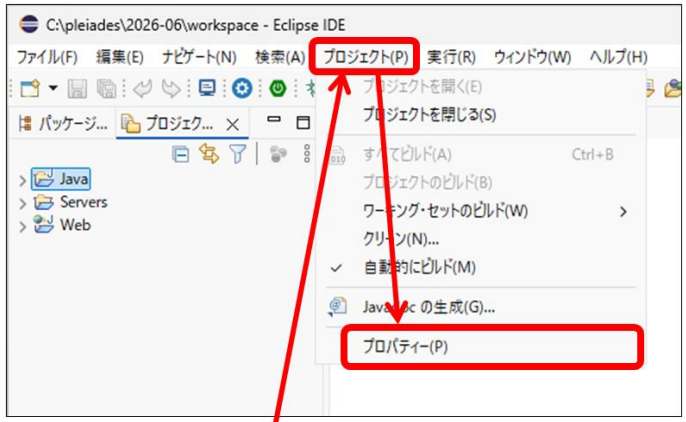
注. sqlite-jdbcのJARファイルは次のサイトからダウンロードできる。
https://repo1.maven.org/maven2/org/xerial/sqlite-jdbc/3.53.2.1/

クラスパスにJDBCを追加



プロジェクト名「Java」をクリック

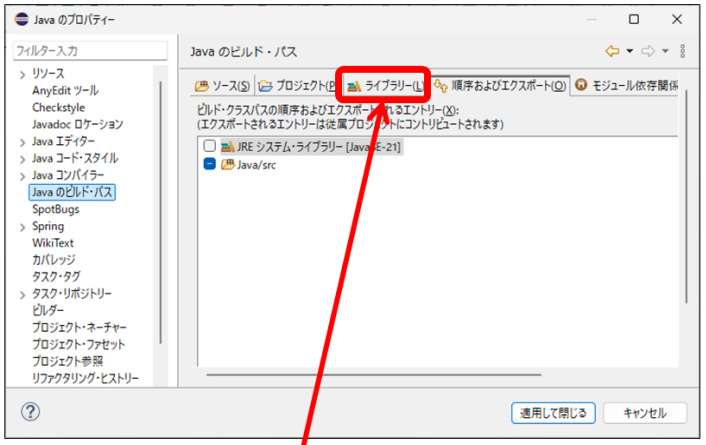
クラスパスにJDBCを追加



「プロジェクト → プロパティ」を選択

73

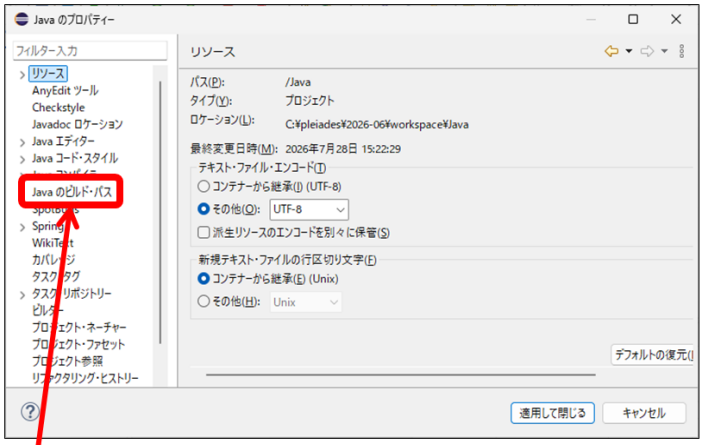
クラスパスにJDBCを追加



「ライブラリー」をクリック

75

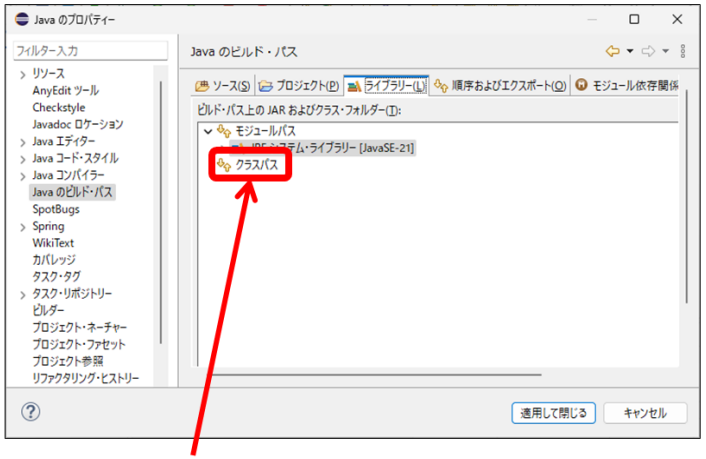
クラスパスにJDBCを追加



「Javaのビルドパス」をクリック

74

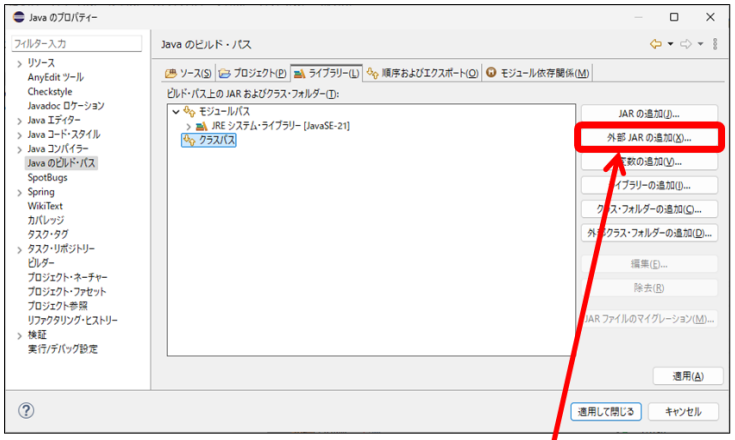
クラスパスにJDBCを追加



「クラスパス」をクリック

76

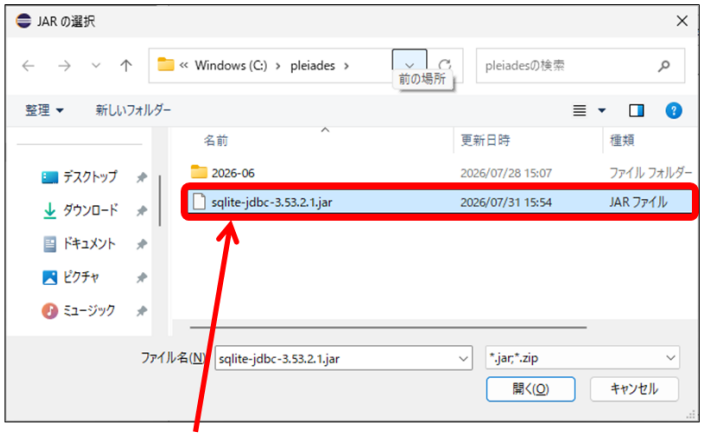
クラスパスにJDBCを追加



「外部JARの追加」をクリック

77

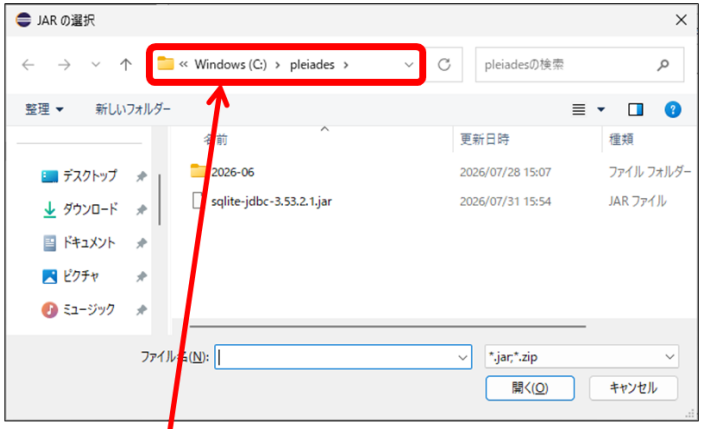
クラスパスにJDBCを追加



sqlite-jdbc-XXX.jar をクリックして選択
（XXXはバージョン番号）

79

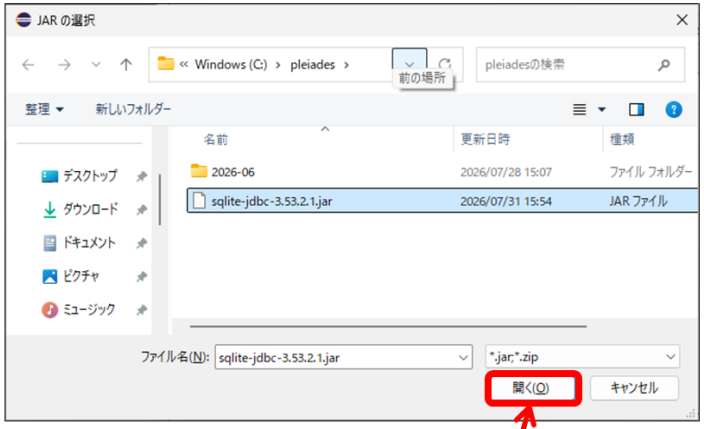
クラスパスにJDBCを追加



「C:¥pleiades」に移動

78

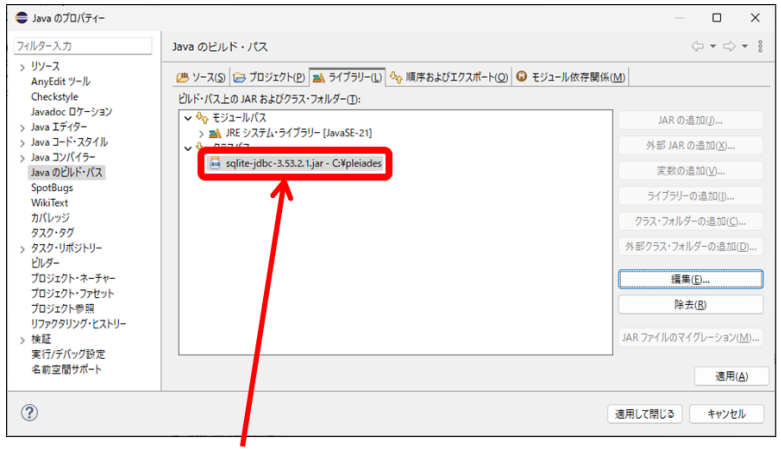
クラスパスにJDBCを追加



「開く」をクリック

80

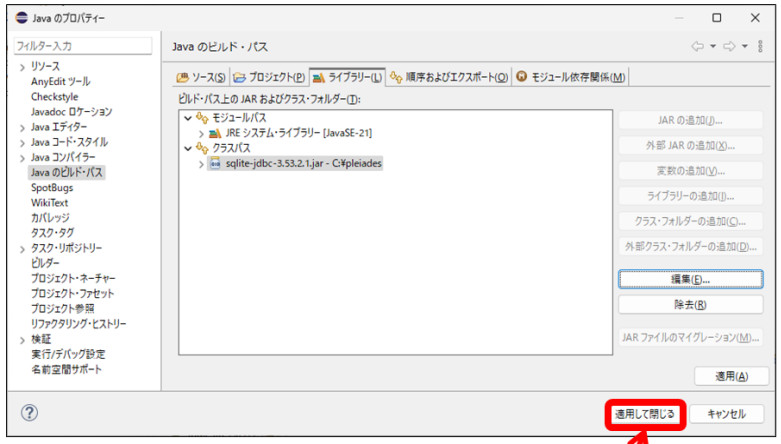
クラスパスにJDBCを追加



JARファイルが追加される

81

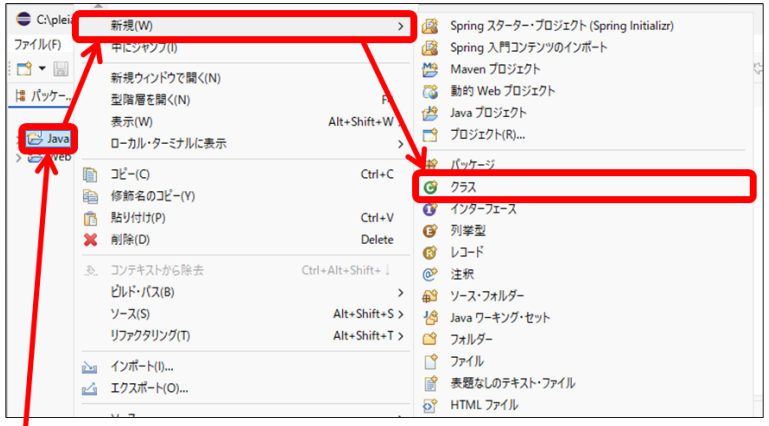
クラスパスにJDBCを追加



「適用して閉じる」をクリック

82

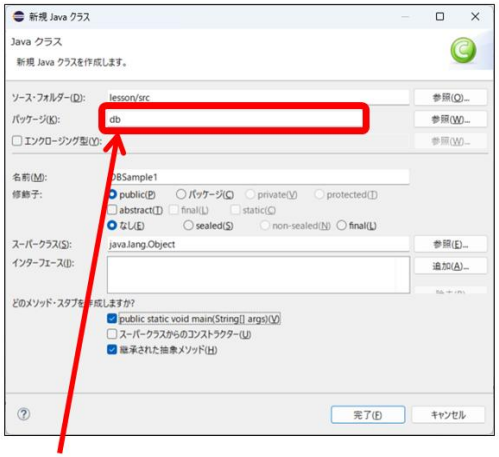
プログラムの作成と実行



プロジェクト名Javaを右クリックして「新規 → クラス」を選択

83

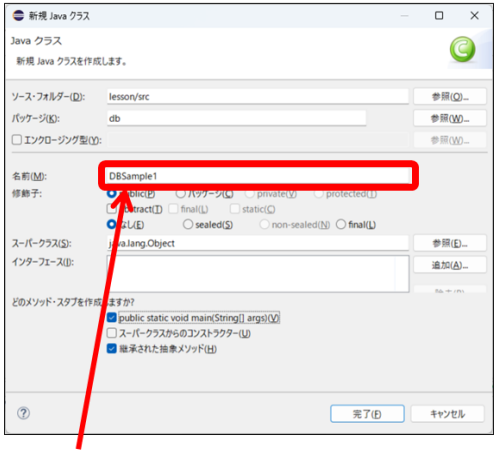
プログラムの作成と実行



パッケージに「db」と入力

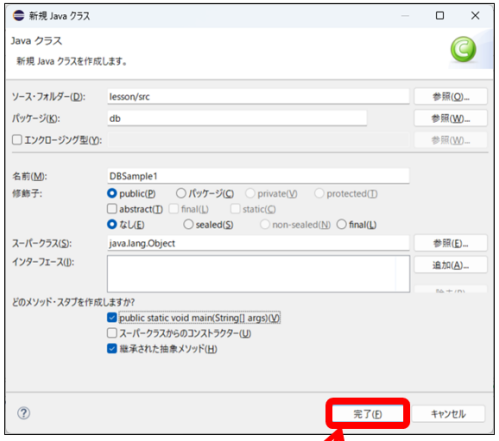
84

プログラムの作成と実行



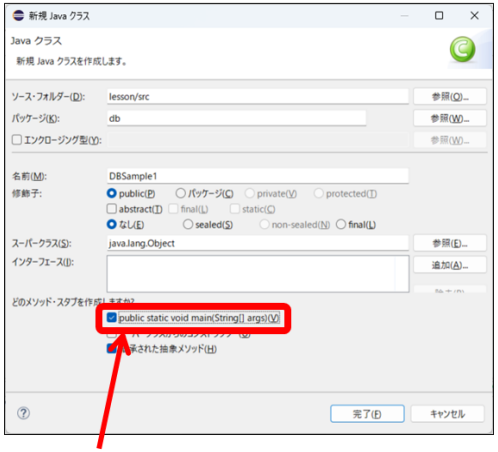
名前に「DBSample1」と入力

プログラムの作成と実行



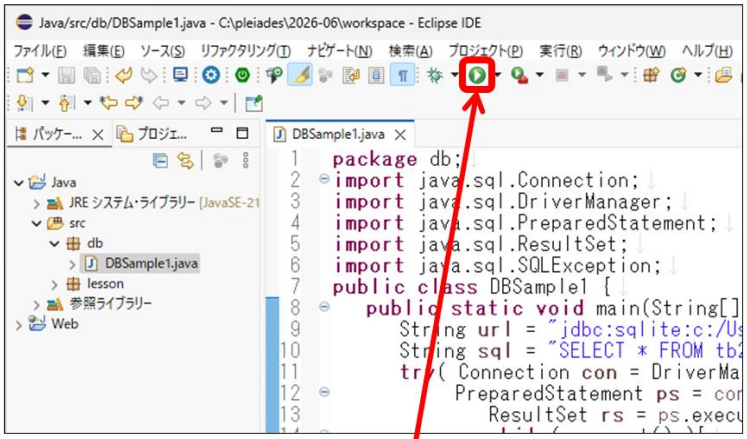
「完了」をクリック

プログラムの作成と実行



「public static void main(・・・)」にチェック

実行ボタンを押して実行



実行ボタン「 」をクリックして実行

プログラムの作成と実行

DBSample1.java

```
1  package db;
2  import java.sql.*;
3  public class DBSample1 {
4     public static void main(String[] args) {
5        String url = "jdbc:sqlite:c:/Users/jcreation/test.db";
6        String sql = "SELECT * FROM tb2 WHERE id = ?";
7        try( Connection con = DriverManager.getConnection(url);
8             PreparedStatement stmt = con.prepareStatement(sql) ){
9              stmt.setInt(1, 2); //追加(1個目の?を2に置き換え)
10             ResultSet rs = stmt.executeQuery(); //引数なし
11             while( rs.next() ){
12                System.out.println(
13                      rs.getInt(1) + "¥t" + rs.getString(2));
14          }
15       }catch(SQLException e){
16          e.printStackTrace();
17       }
18    }
19 }
```

コンソール　サーバー

```
<終了> DBSample1 [Java アプリケーション] C:¥pleiades¥j
2    高橋
```

プログラムを作成して実行

SQL文実行の例

●挿入、更新、削除の例を示す。これらの操作にはメソッドexecuteUpdate()を使用する。メソッドの戻り値は更新のあった行数となる。

注. executeUpdate()では表の作成や削除も行える。

DBSample2.java(挿入・更新・削除)

```
package db;
import java.sql.*;
public class DBSample2 {
   public static void main(String[] args) {
      update("INSERT INTO tb2(name) VALUES('山田')");
      update("UPDATE tb2 SET name = '綿鍋' WHERE id = 1");
      update("DELETE FROM tb2 WHERE id = 2");
   }
   static void update(String sql){
      String url = "jdbc:sqlite:c:/Users/jcreation/test.db";
      try( Connection con = DriverManager.getConnection(url);
           PreparedStatement ps = con.prepareStatement(sql) ){
              ps.executeUpdate();
      }catch(SQLException e){
         e.printStackTrace();
      }
   }
}
```

実行結果(コマンドプロンプトで確認)

```
C:¥Users¥jcreation> sqlite3 test.db
sqlite> select * from tb2;
1|綿鍋
3|山田
```


問題5

① 表 tb2 の列 name の値を、idの降順で表示するプログラムを作成せよ。

② 表 tb2 に列 name の値が「上田」の行を挿入せよ。ここで id は指定しないものとする。

③ 表 tb2 において列 id の値が 1 の行の name の値を「森山」に変更せよ。

④ 表tb2から列 id の値が 3 の行を削除せよ。

問題①解答例

```
package db;
import java.sql.*;
public class DBQ1 {
   public static void main(String[] args) {
      String url = "jdbc:sqlite:c:/Users/jcreation/test.db";
      String sql = "SELECT name FROM tb2 ORDER BY id DESC";
      try( Connection con = DriverManager.getConnection(url);
           PreparedStatement ps = con.prepareStatement(sql) ){
              ResultSet rs = ps.executeQuery();
              while( rs.next() ){
                 System.out.println(rs.getString(1));
         }
      }catch(SQLException e){
         e.printStackTrace();
      }
   }
}
```

実行結果

```
山田
綿鍋
```

93

問題②③④解答例

```
package db;
import java.sql.*;
public class DBQ234 {
   public static void main(String[] args) {
      update("INSERT INTO tb2(name) VALUES('上田')");
      update("UPDATE tb2 SET name = '森山' WHERE id = 1");
      update("DELETE FROM tb2 WHERE id = 3");
   }
   static void update(String sql){
      String url = "jdbc:sqlite:c:/Users/jcreation/test.db";
      try( Connection con = DriverManager.getConnection(url);
           PreparedStatement ps = con.prepareStatement(sql) ){
              ps.executeUpdate();
      }catch(SQLException e){
         e.printStackTrace();
      }
   }
}
```

実行結果(コマンドプロンプトで確認)

```
C:¥Users¥jcreation> sqlite3 test.db
sqlite> select * from tb2;
1|森山
4|上田
```

94