テーブルの分解 (table decomposition)

URL: http://www.db.is.kyushu-u.ac.jp/rinkou/addb/4.html

table 'results'

name	teacher_name	student_name	score
Database	K	KK	85
Database	K	AA	75
Database	K	LL	90
Programming	A	KK	85
Programming	A	LL	75

```
SELECT name, teacher
FROM results;
```

name	teacher_name
Database	K
Database	K
Database	K
Programming	A
Programming	A

Query result is one new table

table 'results'

name	teacher_name	student_name	score
Database	K	KK	85
Database	K	AA	75
Database	K	LL	90
Programming	A	KK	85
Programming	A	LL	75

```
SELECT DISTINCT name, teacher
FROM results;
```

重複した値を持つ行は 1つの行になる
Duplicate Records are Eliminated from the Query Result

name	teacher_name
Database	K
Programming	A

Query result is one new table

テーブルの分解 (Table Decomposition)

■ 分解前

results(name, teacher_name, student_name, score)

name	teacher_name	student_name	score
Database	K	KK	85
Database	K	AA	75
Database	K	LL	90
Programming	A	KK	85
Programming	A	LL	75

■ 分解後

A(name, teacher_name), B(name, student_name, score)

name	teacher_name
Database	K
Programming	A

name	student_name	score
Database	KK	85
Database	AA	75
Database	LL	90
Programming	KK	85
Programming	LL	75

テーブルの分解 (Table Decomposition)

■ 分解前

results(name, teacher_name, student_name, score)

■ 分解後

A(name, teacher_name), B(name, student_name, score)

results の全ての属性が，分解後のテーブルに含まれていること.
All attributes in 'results' must be included in the decomposed tables.

テーブルの分解と結合

A(name, teacher_name), B(name, student_name, score)

name	teacher_name
Database	K
Programming	A

name	student_name	score
Database	KK	85
Database	AA	75
Database	LL	90
Programming	KK	85
Programming	LL	75

```
SELECT A.name, A.teacher_name, B.student_name, B.score
FROM A, B
WHERE A.name = B.name
```

結合問い合わせにより元のテーブルに戻るので，この分解は無損失分解である．
The original table 'results' is constructed.
The **decomposition** is **LOSSLESS**.

name	teacher_name	student_name	score
Database	K	KK	85
Database	K	AA	75
Database	K	LL	90
Programming	A	KK	85
Programming	A	LL	75

- テーブルの分解は，必ずしも情報無損失ではない
  table decompositions are not always lossless
- データベース設計には 2つの異なる方針がある
  Two alternatives in database design
  1. テーブルを情報無損失分解する
     Decompose tables losslessly
  2. テーブルをなるべく分解しない
     Do not decompose tables
- テーブルを情報無損失分解することで，データの管理がしやすくなる場合がある
  Lossless table decomposition is useful to make database management ease

- **テーブル(table)**
  リレーショナルデータベース での「テーブル」は，多重集合 (muti set)を基礎とする体系になっているということに注意して欲しい．**「テーブル」は，同じ値をもった行が複数回現れることが許される**．
- SQL における**DISTINCT 指定**
  SQL では，SELECT と選択リスト内の間に「DISTINCT」と書く場合がある．これを **DISTINCT 指定**という．

DISTINCT 指定を行うと，問い合わせの結果に**重複行があるとき1つを残して他を除去**する．重複行というときは，**行全体が同じ値**だという意味である．下記の例では，同じ注文者名が別の商品番号の注文を行った場合には重複行ではない．

```
SELECT DISTINCT 商品番号, 注文者名
FROM 注文
```

DISTINCT 指定があるとき，重複行があるとき1つを残して他を除去するから，SQL の評価結果は多重集合にはなることは無い．必ず，リレーションになる．一方で，DISTINCT 指定がないときは，SQL の評価結果が多重集合になる場合がある．

補足説明しておく．下記のテーブル results を例に説明する．

```
name       | teacher_name | student_name | score
-----------------------------------------------
Database   | K            | KK           | 85
Database   | K            | AA           | 75
Database   | K            | LL           | 90
Programming | A           | KK           | 85
Programming | A           | LL           | 75
```

このテーブル results から，**属性 name と teacher_name だけを抜き出して**新しいテーブルを作ったとする．下記のように重複行が現れる．つまり新しく出来たテーブルは**多重集合**である．

```
name       | teacher_name
--------------------------
Database   | K
Database   | K
Database   | K
Programming | A
Programming | A
```

上記のテーブルを作るための SQL は次の通りである．

```
SELECT name, teacher_name
FROM   results;
```

一方で，**射影** results[name, student_name] といったときは，属性 name と teacher_name だけを抜き出した上で，**重複行があるとき1つを残して他を除去**する．結果は次の通りである．

```
name       | teacher_name
--------------------------
Database   | K
Programming | A
```

射影を行うための SQL は，次のように **DISTINCT 指定**が付く．

```
SELECT DISTINCT name, teacher_name
FROM   results;
```

- **テーブルの分解**

  あるテーブルのテーブル名が R，属性名が A1, A2, …, An であるとする．また，このテーブルには **重複行が無い**ものと仮定する．このテーブルの，m 個のテーブル X1, X2, ..., Xm への分解は次のように定義される．

  X1, X2, …, Xm をRの全属性集合{A1, A2,…, An}の部分集合で，

  $$X1 \cup X2 \cup \cdots \cup Xm = \{A1, A2, \cdots, An\}$$

  なる条件を満たすとき（XiとXjは共通集合をもってよい），テーブルを m 個の射影 R[X1], R[X2], …, R[Xm] で置き換えることを，このテーブルの分解という．

  ※ 重複行を持つようなテーブルについては，分解を考えないことにする（重複行を持つテーブルについては，無損失分解であることの定義が「難しい」から）．

- **単純値 atomic value**

  分解不可能な値のこと．直積集合の要素や，べき集合の要素は単純値ではないと考えるのが普通である．

  - 直積集合の例：属性名「family name」のドメインと，属性名「given name」のドメインの直積集合
  - べき集合の例：ドメイン {みかん, りんご, バナナ}から構成されるべき集合は，{φ, {みかん}, {りんご}, {バナナ}, {みかん, りんご}, {りんご, バナナ}, {バナナ, みかん}, {みかん, りんご, バナナ}}

演習

演習で行うこと

- Sqliteman の起動と終了 (Start Sqliteman)
- Sqliteman で既存のデータベースを開く (Open an existing database using Sqliteman)
- SQL を用いたテーブル定義と一貫性制約の記述 (Table defintion and integrity constraint specification using SQL)

  リレーショナル・スキーマ (relational schema): **results(name, teacher_name, student_name, score)**

  SQL プログラム:

```
CREATE TABLE results (
    name         TEXT  NOT NULL,
    teacher_name TEXT  NOT NULL,
    student_name TEXT  NOT NULL,
    score        INTEGER NOT NULL CHECK ( score >= 0 AND score <= 100 ),
    UNIQUE (name, student_name),
    UNIQUE (teacher_name, student_name) );
```

- SQL を用いたテーブルへの行の挿入 (Insert rows into a table using SQL)

```
BEGIN TRANSACTION;
INSERT INTO results VALUES( 'Database', 'K', 'KK', 85 );
INSERT INTO results VALUES( 'Database', 'K', 'AA', 75 );
INSERT INTO results VALUES( 'Database', 'K', 'LL', 90 );
INSERT INTO results VALUES( 'Programming', 'A', 'KK', 85 );
INSERT INTO results VALUES( 'Programming', 'A', 'LL', 75 );
COMMIT;
```

- Sqliteman を用いたデータのブラウズ (Browse Data using Sqliteman)

- DISTINCT を含む SQL の例 (SQL with DISTINCT)

```
SELECT DISTINCT name, teacher_name
FROM results;

SELECT name, teacher_name
FROM results;

SELECT DISTINCT score
FROM results;

SELECT score
FROM results;
```

- SQL を用いたテーブルの分解 (table decomposition using SQL)

```
CREATE TABLE A AS
SELECT DISTINCT name, teacher_name
FROM results;

CREATE TABLE B AS
SELECT DISTINCT name, student_name, score
FROM results;
```

- 結合問い合わせ (join query)

```
SELECT A.name, A.teacher_name, B.student_name, B.score
FROM A, B
WHERE A.name = B.name;
```

- 情報無損失分解にならない分解 (Inspect lossless decomposition)

```
CREATE TABLE C AS
SELECT DISTINCT name, student_name
FROM results;

CREATE TABLE D AS
SELECT DISTINCT teacher_name, student_name, score
FROM results;

SELECT C.name, D.teacher_name, C.student_name, D.score
FROM C, D
WHERE C.student_name = D.student_name;
```

SQLite の SQL 演習に関連する部分

SQLite の SQL の説明は http://www.hwaci.com/sw/sqlite/lang.html (English Web Page) にある.

- **SELECT DISTINCT ...**
  同一の値を持つ行が複数あるとき, 1つを残して除去する

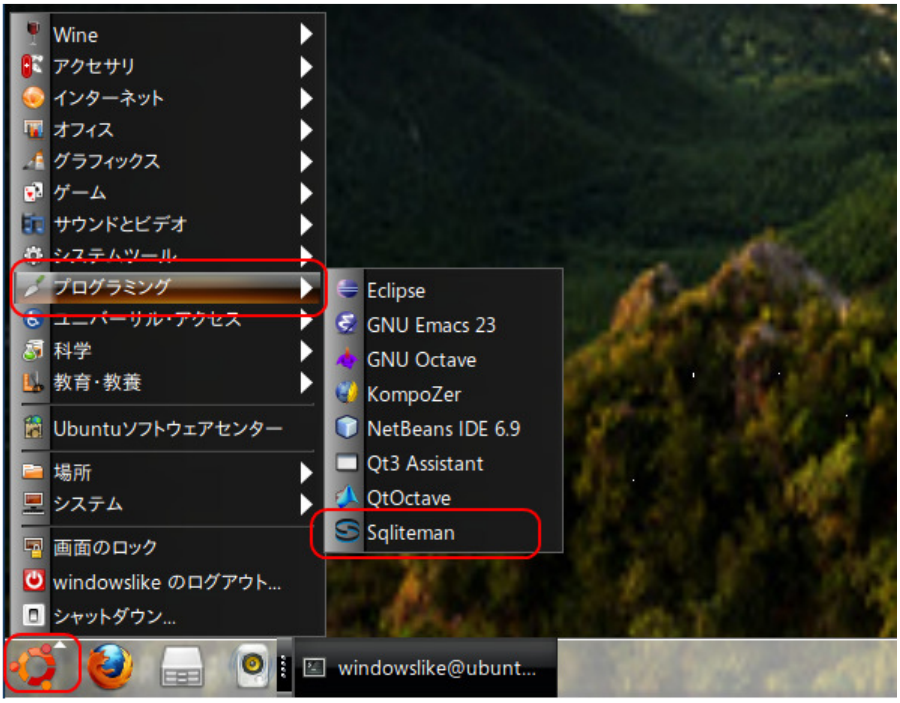
- **CREATE TABLE AS <table-name> SELECT ...**
  SQL 問い合わせ結果から新しいテーブルを作る

Sqliteman の起動と終了 (Start Sqliteman)

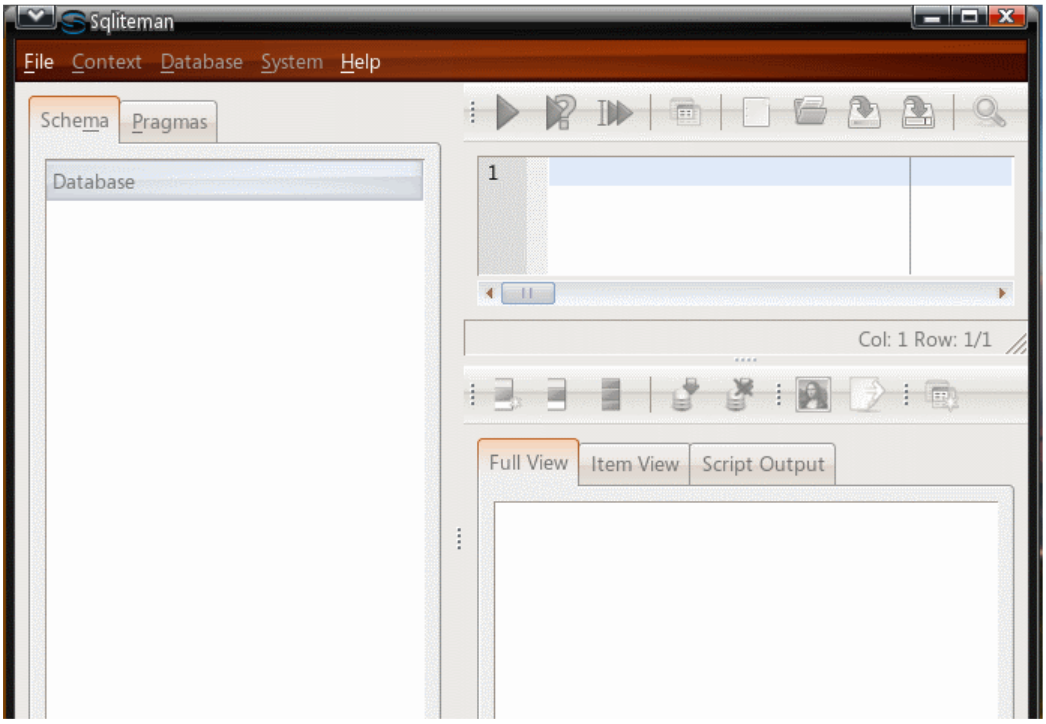
1. **Sqliteman の起動** (Start Sqliteman)

   ■ **Ubuntu での SQLiteman の起動例**

   「**プログラミング**」→「**Sqliteman**」と操作する.



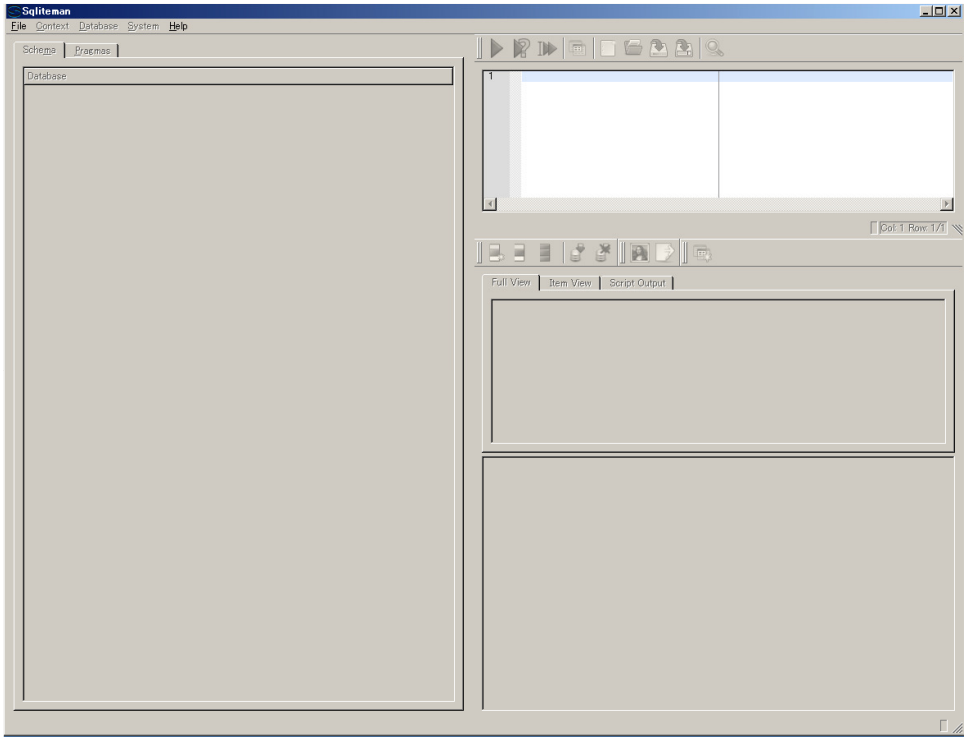
SQLiteman の新しいウインドウが開く. (A New window appears)



■ **Windows での SQLiteman の起動例**

「Sqliteman」のアイコンをダブルクリック (double click "Sqliteman.exe")

Sqliteman のウインドウが開く. (A New window appears)

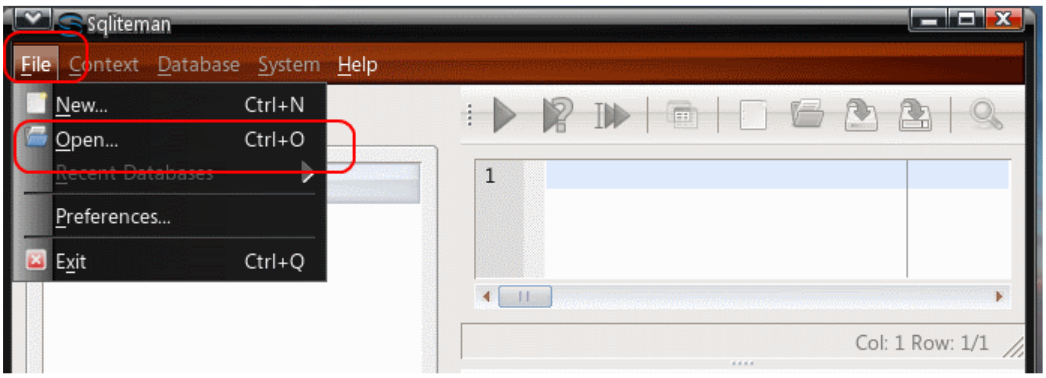


Sqliteman で既存のデータベースを開く(Open an existing database using Sqliteman)

すでに作成済みのデータベースを，下記の手順で開くことができる．

以下の手順で，**既存のデータベースファイルを開く**．(Open an existing database file)

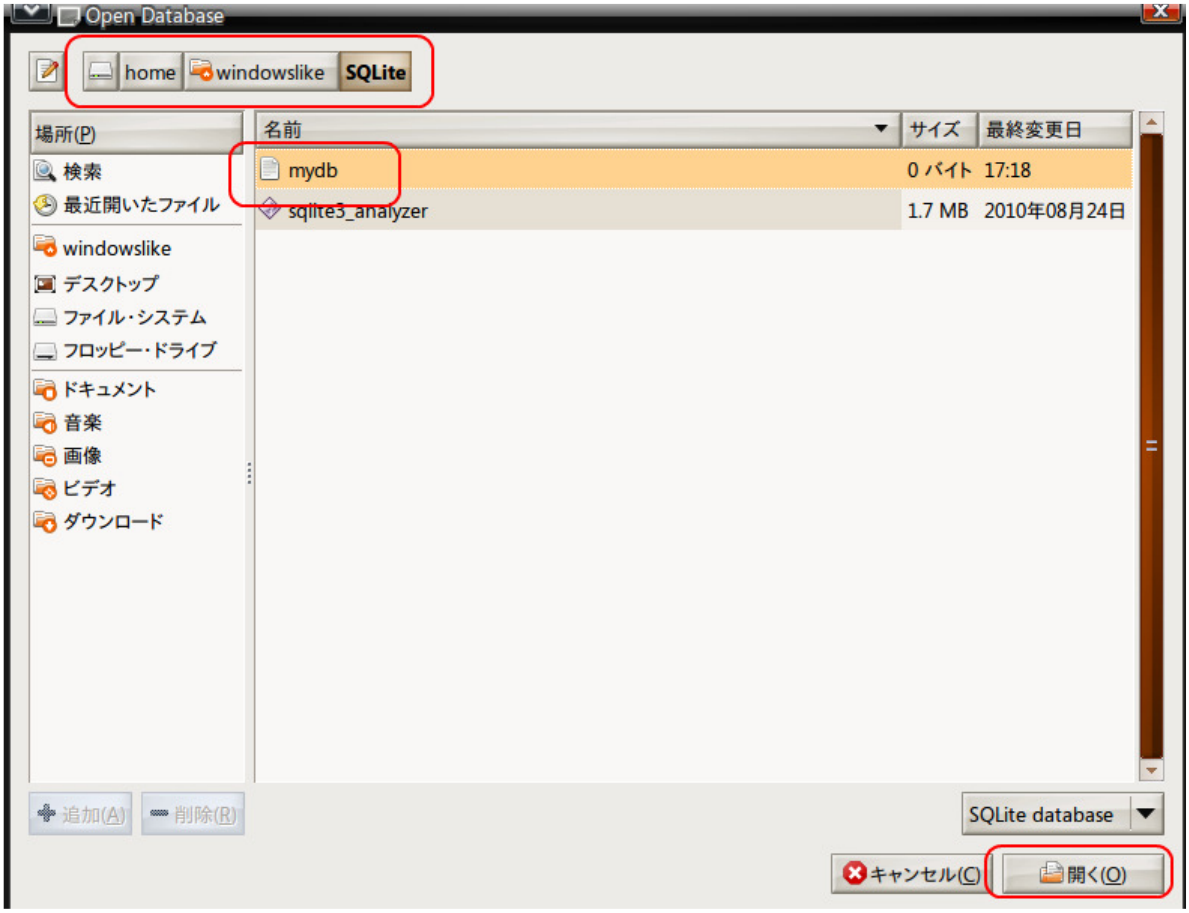
1. 「**File**」→「**Open**」



2. **データベースファイルを開く**(Open Database File)

■ **Ubuntu での実行例**(「SQLite/mydb」を開く場合)

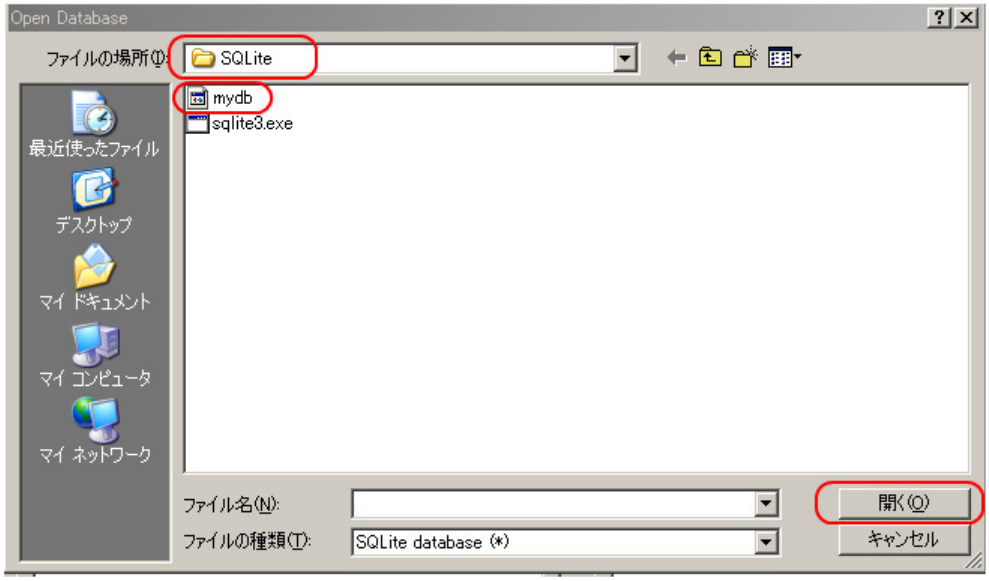
データベースファイル **SQLite/mydb** を選び，「開く」をクリック (Click '開く' after choosing the database file "SQLite/mydb")



■ **Windows での実行例**(「C:¥SQLite¥mydb」を開く場合)

データベースファイル **C:¥SQLite¥mydb** を選び,「開く」をクリック (Click '開く' after choosing the database file "C:¥SQLite¥mydb")

要するに, /home/<ユーザ名>/SQLite の**下**の mydb を選ぶ.



SQL を用いたテーブル定義と一貫性制約の記述 (Table definition and integrity constraint specification using SQL)

SQL を用いて, **results テーブルを定義し, 一貫性制約を記述**する. (Define 'results' table and specify integrity constrants of the table using SQL)

リレーショナル・スキーマ (relational schema): **results(name, teacher_name, student_name, score)**

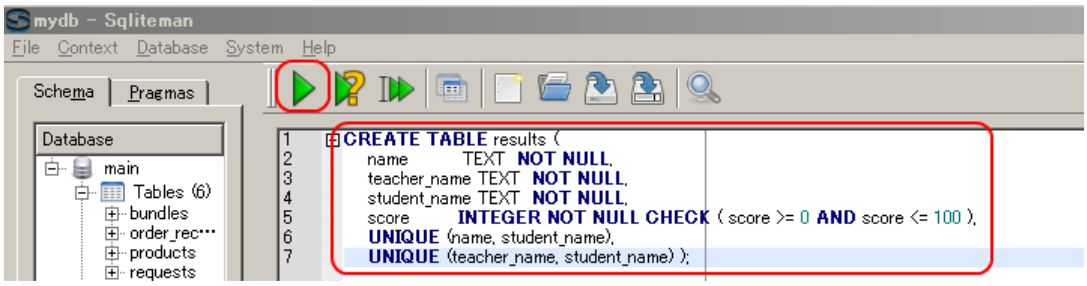
1. **results テーブルの定義** (Define a table)

   次の SQL を入力し,「Run SQL」のアイコンをクリック (Write the following SQL, and click "Run SQL" icon).

```
CREATE TABLE results (
    name         TEXT  NOT NULL,
    teacher_name TEXT  NOT NULL,
    student_name TEXT  NOT NULL,
    score        INTEGER NOT NULL CHECK ( score >= 0 AND score <= 100 ),
```

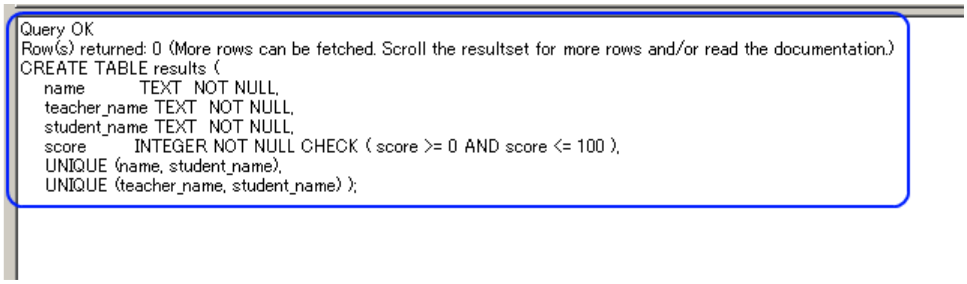
```
    UNIQUE (name, student_name),
    UNIQUE (teacher_name, student_name) );
```

※「SQL Editor」のウインドウには，SQL プログラムを書くことができる．In the '**SQL string**' window, you can write down **SQL program(s)**.



2. **コンソールの確認** (Inspect console)

   エラーメッセージが出ていないことを確認



SQL を用いたテーブルへの行の挿入 (Insert rows into a table using SQL)

次のような **results** テーブルを作る．(Construct table 'results')

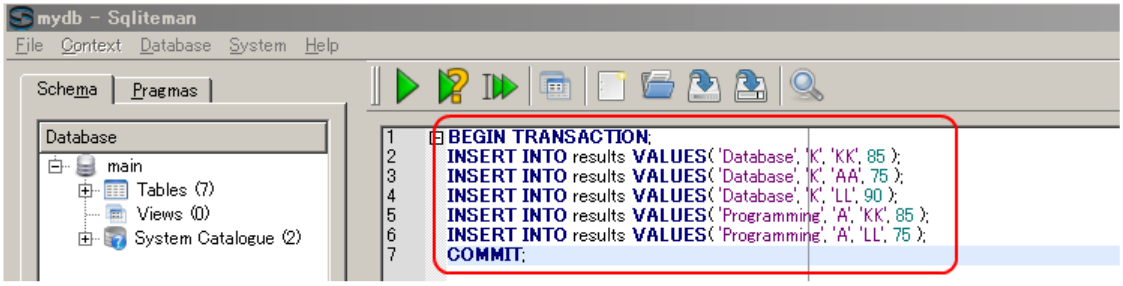
```
name        | teacher_name |  student_name | score
-----------------------------------------------
Database    |   K          |  KK           | 85
Database    |   K          |  AA           | 75
Database    |   K          |  LL           | 90
Programming |   A          |  KK           | 85
Programming |   A          |  LL           | 75
```

以下の手順で，SQL を用いて **results テーブルへの行の挿入**を行う (Insert rows into table 'results' using SQL)

1. **SQL プログラムの記述**

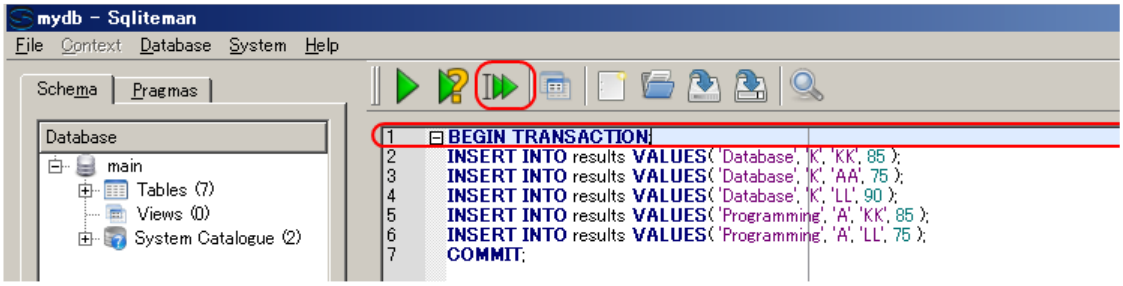
   「INSERT INTO …」は行の挿入．ここには **5つの SQL 文**を書き，「BEGIN TRANSACTION」と「COMMIT」で囲む．("INSERT INTO …" means inserting a row into a table. Four SQL statements are wrote).

```
BEGIN TRANSACTION;
INSERT INTO results VALUES( 'Database', 'K', 'KK', 85 );
INSERT INTO results VALUES( 'Database', 'K', 'AA', 75 );
INSERT INTO results VALUES( 'Database', 'K', 'LL', 90 );
INSERT INTO results VALUES( 'Programming', 'A', 'KK', 85 );
INSERT INTO results VALUES( 'Programming', 'A', 'LL', 75 );
COMMIT;
```

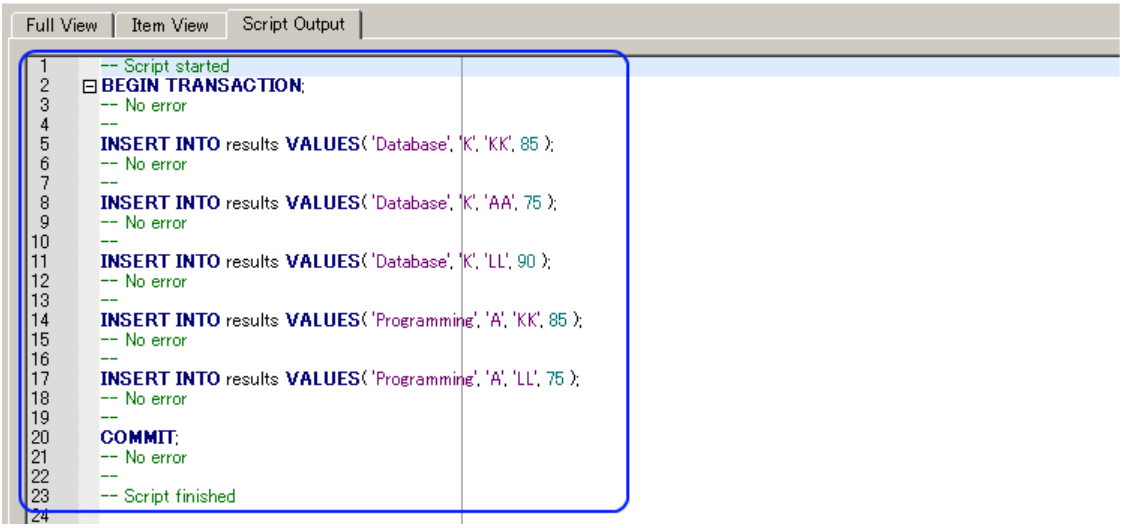


2. **複数の SQL 文の一括実行** (Run multiple SQL statements)

   複数の SQL 文を一括実行したいので，**カーソルを先頭行に移動**した後に，「Run multiple SQL statements …」のボタンをクリックする．「Move the cursor to the top statement. Click "Run multiple SQL statements from current cursor position in one batch" icon)



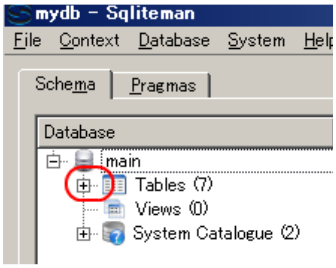
3. **「Script Output」ウインドウの確認** (Inspect "Script Output" window)
   エラーメッセージが出ていないことを確認



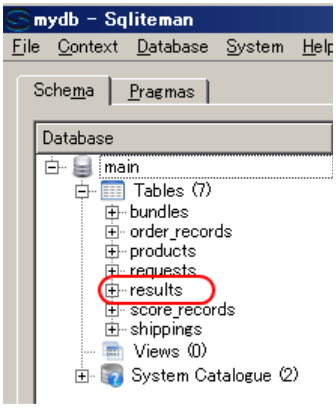
Sqliteman を用いたデータのブラウズ (Browse Data using Sqliteman)

- **results テーブル**

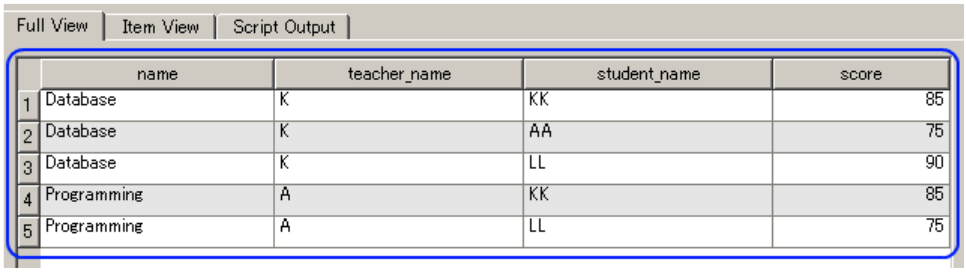
  まず, オブジェクト・ブラウザ (Object Browser) の中の「**Tables**」を展開 (Click 'Tables')



  次に, テーブル **results**を選ぶ (Select table 'results')



  テーブル **results**が表示される (table 'results' appears)



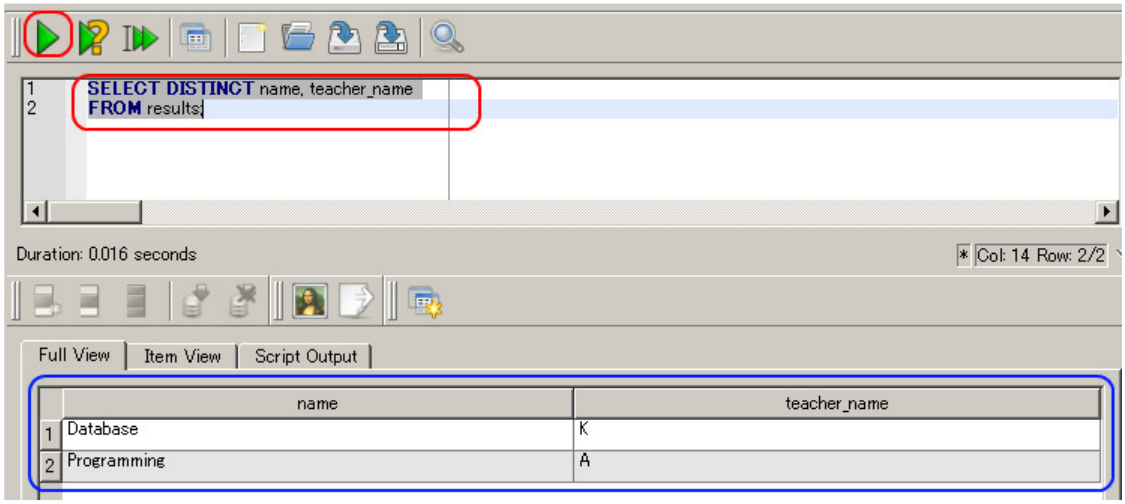
	name	teacher_name	student_name	score
1	Database	K	KK	85
2	Database	K	AA	75
3	Database	K	LL	90
4	Programming	A	KK	85
5	Programming	A	LL	75

※ もし，データに間違いがあれば，このウインドウで修正できる (If you find any mistakes, you can modify the data using this window).

DISTINCT を含む SQL の例 (SQL with DISTINCT)

DISTINCT を付ける (**with** DISTINCT)

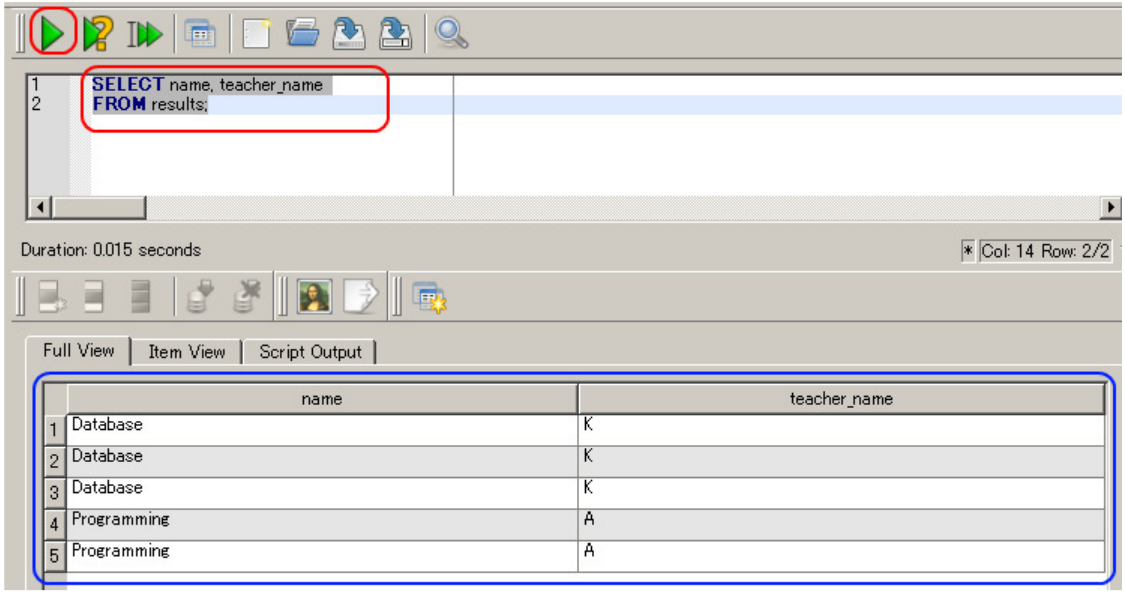
```
SELECT DISTINCT name, teacher_name
FROM results;
```



	name	teacher_name
1	Database	K
2	Programming	A

DISTINCT を付けない (**without** DISTINCT)

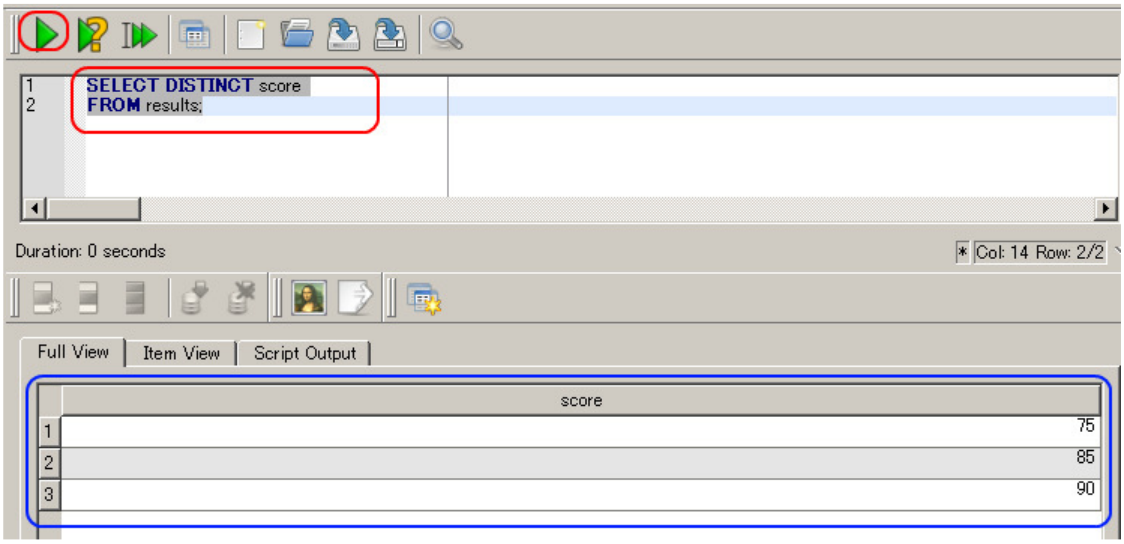
```
SELECT name, teacher_name
FROM results;
```



	name	teacher_name
1	Database	K
2	Database	K
3	Database	K
4	Programming	A
5	Programming	A

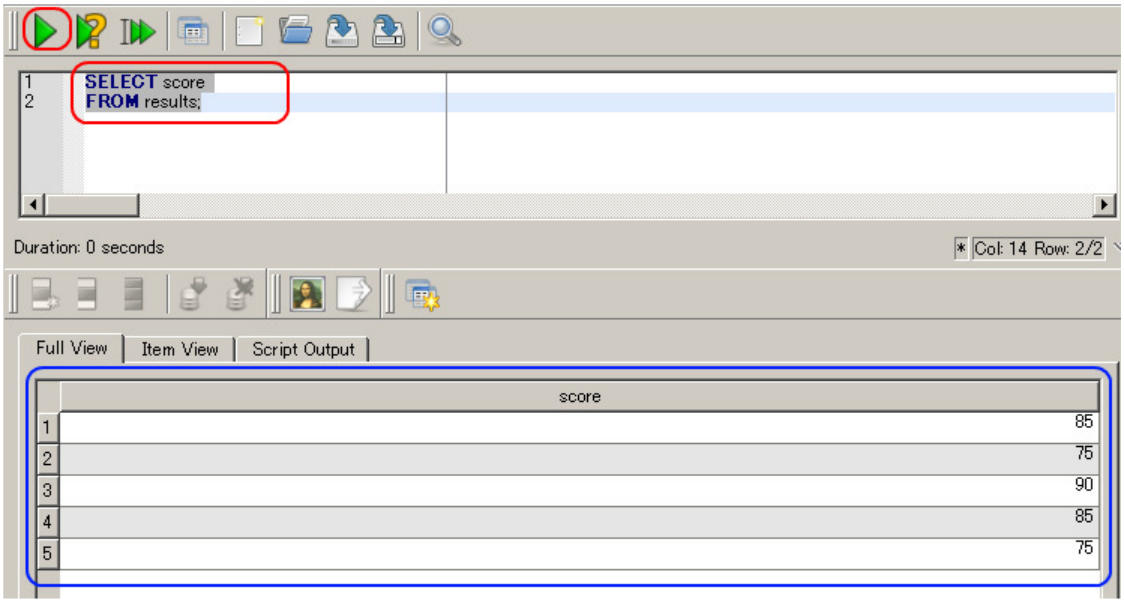
DISTINCT を付ける (**with** DISTINCT)

```
SELECT DISTINCT score
FROM results;
```



DISTINCT を付けない (**without** DISTINCT)

```
SELECT score
FROM results;
```



SQL を用いたテーブルの分解 (table decomposition using SQL)

results(name, teacher_name, student_name, score)を 2つのテーブル

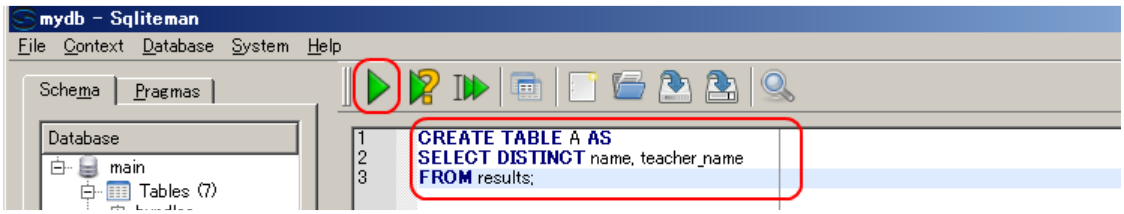
- A(name, teacher_name)
- B(name, student_score, score)

に分解. **テーブルを分解**するときは**重複行を除去するという規則がある**ことに注意. DISTINCTを使う.

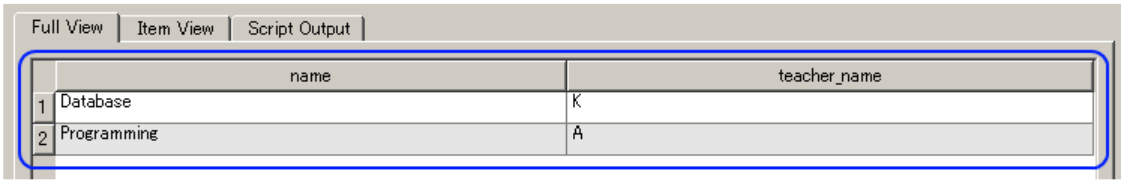
'results' table is decomposed into two tables A and B. Duplicate rows in A and B are eliminated when decomposition (because of the definition of "table decomposition"). Use 'DISTINCT'.

SQL の実行 (Execute SQL)

```
CREATE TABLE A AS
SELECT DISTINCT name, teacher_name
FROM results;
```

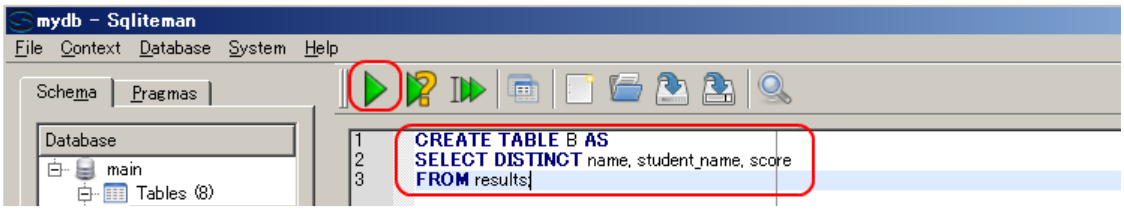


データのブラウズ機能を使いテーブル A を確認 (Browse Table A)

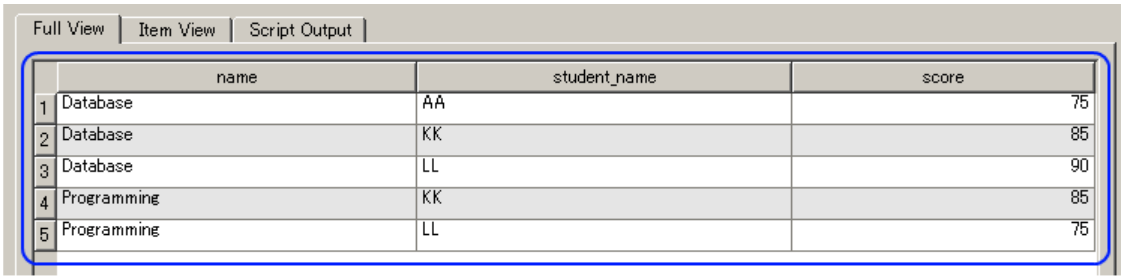


SQL の実行 (Execute SQL)

```
CREATE TABLE B AS
SELECT DISTINCT name, student_name, score
FROM results;
```



データのブラウズ機能を使いテーブル B を確認 (Browse Table B)



結合問い合わせ (join query)

元のテーブル results が，分解後の2つのテーブル A, B から復元できることを確認しておこう．テーブル results は，テーブル A と B から復元できるから，データベース設計として．

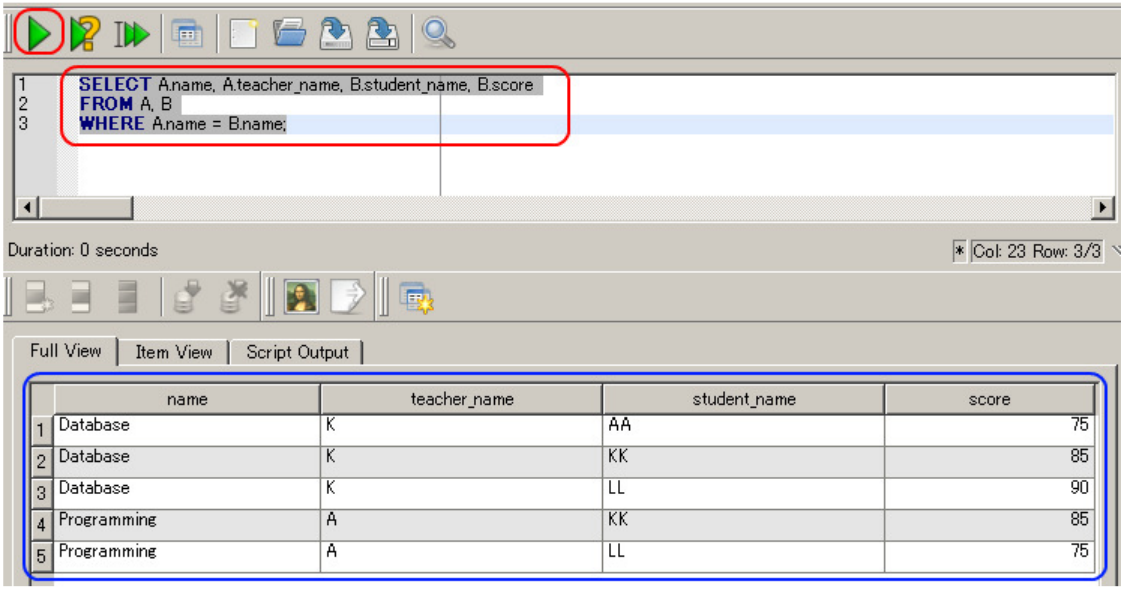
- テーブル results をデータベースに格納するか，あるいは，
- テーブル results でなく，テーブル A と B をデータベースに格納する

という 2つの案がありえることになる．(では，どちらが良いのか？　今度の授業で明らかにしていく)

Examine that the original table 'results' can be constructed from the two tables **A and B**. It means that there are two alternatives in database design.

1. Store 'results' into database, or
2. Store 'A' and 'B' instead of 'results' in database.

```
SELECT A.name, A.teacher_name, B.student_name, B.score
FROM A, B
WHERE A.name = B.name;
```



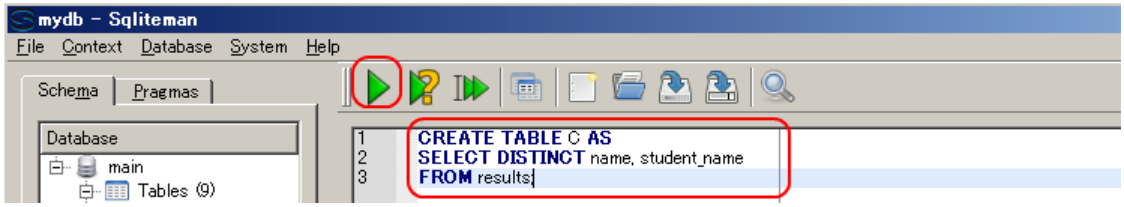
情報無損失分解にならない分解 (Inspect lossless decomposition)

今度は，テーブルの分解の仕方を変える．分解後のテーブルから，もとのテーブルに戻らない場合がある．このことを確認しておく．

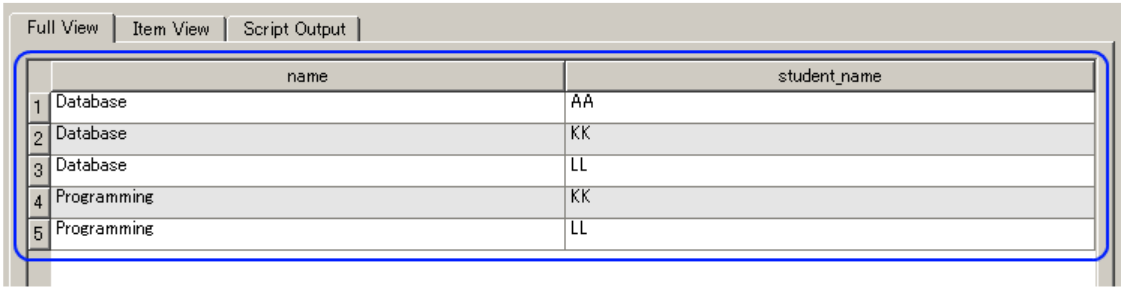
Table decompositions are **not always LOSSLESS**.

SQL の実行 (Execute SQL)

```
CREATE TABLE C AS
SELECT DISTINCT name, student_name
FROM results;
```



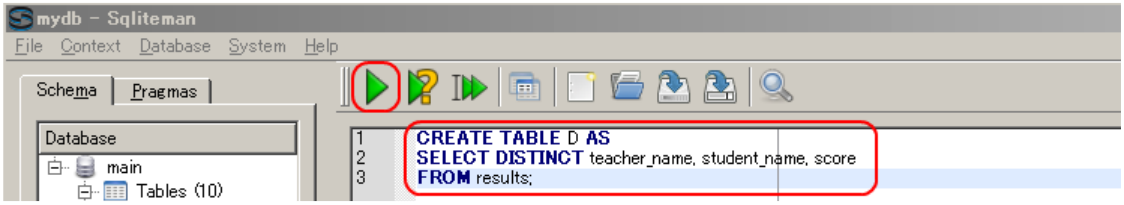
データのブラウズ (Browse Data)



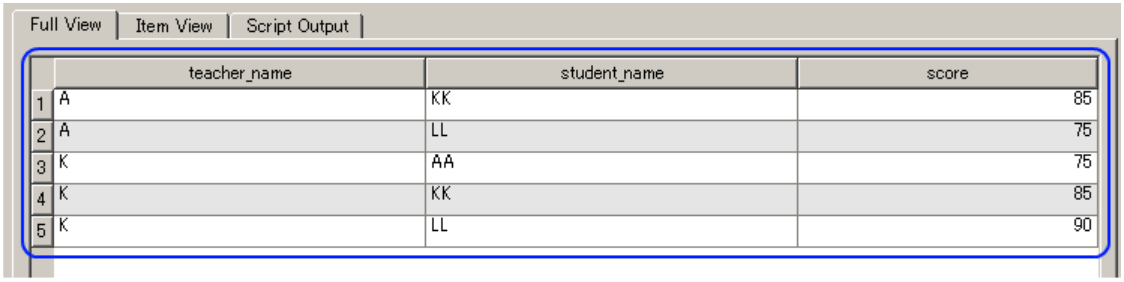
	name	student_name
1	Database	AA
2	Database	KK
3	Database	LL
4	Programming	KK
5	Programming	LL

SQL の実行 (Execute SQL)

```
CREATE TABLE D AS
SELECT DISTINCT teacher_name, student_name, score
FROM results;
```



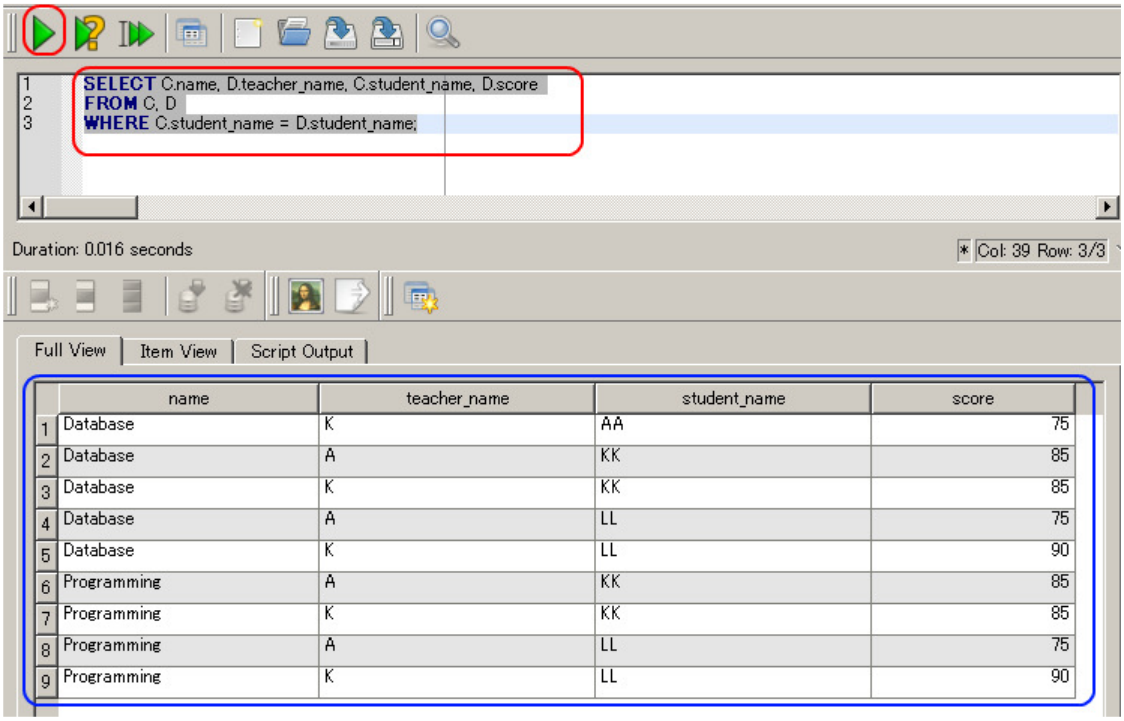
データのブラウズ (Browse Data)



	teacher_name	student_name	score
1	A	KK	85
2	A	LL	75
3	K	AA	75
4	K	KK	85
5	K	LL	90

テーブル C, D からは, テーブル results を構築できない ('results' table can not be constructed from C and D).

```
SELECT C.name, D.teacher_name, C.student_name, D.score
FROM C, D
WHERE C.student_name = D.student_name;
```



	name	teacher_name	student_name	score
1	Database	K	AA	75
2	Database	A	KK	85
3	Database	K	KK	85
4	Database	A	LL	75
5	Database	K	LL	90
6	Programming	A	KK	85
7	Programming	K	KK	85
8	Programming	A	LL	75
9	Programming	K	LL	90

演習問題と解答例

次の問いに答えよ. その後, 下記の解答例を確認せよ. Answer the following questions. Then, inspect answers described below.

問い (Questions)

1. 次の PTABLE テーブルに関する問題 (About the following 'PTABLE' table)

```
name    | type   | color
-------------------------
apple   | fruit  | red
apple   | fruit  | blue
rose    | flower | white
rose    | flower | red
rose    | flower | yellow
```

   1. 次の SQL の評価結果は何か? What is the result of the following SQL ?

```
SELECT DISTINCT name, type FROM PTABLE;
```

   2. 次の SQL の評価結果は何か? What is the result of the following SQL ?

```
SELECT DISTINCT name FROM PTABLE;
```

解答例 (Answers)

※ 問い合わせ結果は1つのテーブルになる. その属性名には, 元のテーブル名と属性名をドットでつなげたドット記法を用いている.

1.

```
SELECT DISTINCT name, type FROM PTABLE;

name    | type
------------------
apple   | fruit
rose    | flower
```

2.

```
SELECT DISTINCT name FROM PTABLE;

name
--------
apple
rose
```