

# dd-1. リレーショナルデータベースの特徴

リレーショナルデータベースの  
基本（短縮版）（全7回）

基本を把握したい人へ

<https://www.kkaneko.jp/data/dd/index.html>

金子邦彦



謝辞：この資料では「かわいいフリー素材集 いらすとや」のイラストを使用しています

# 7 回の内容



1. リレーショナルデータベースの特徴
2. テーブルとテーブル定義
3. 問い合わせと SQL
4. 種々の制約, 関連
5. 貸出記録のテーブル
6. さまざまな SQL 問い合わせ
7. テーブルの分解と結合

リレーショナルデータベースの基本を学び, 全体を把握する. SQL の実習付き.

URL: <https://www.kkaneko.jp/data/dd/index.html>

# 第1回のアウトライン



- データベース
- データベースシステム
- リレーショナルデータベースシステム

# データベースは、我々の生活に必須



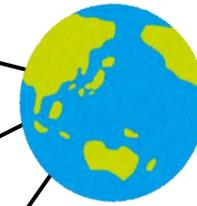
取引



記入



データ保存

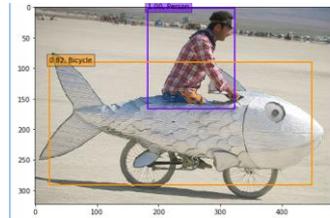


ネットワーク

データベース



センサー連携



人工知能応用



# データベースシステム

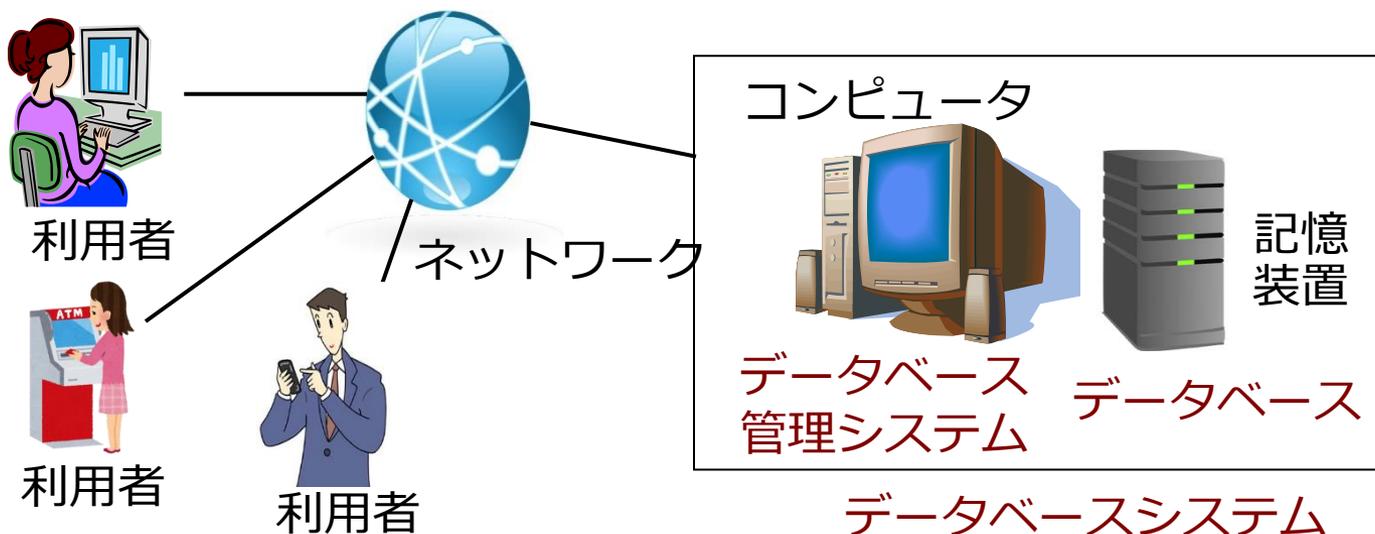


データベースシステムは、データベースを扱う IT のシステム

データベースシステム

= データベース (データの集まり)

+ データベース管理システム (ソフトウェア)

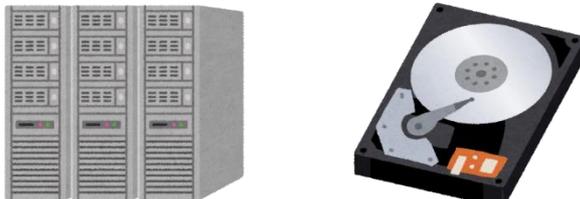


# リレーショナルデータベースシステム



- データベースシステムの種類
- データの形は**テーブル**（**リレーション**ともいう）
- **SQL** の機能

コンピュータ



記憶装置

リレーショナルデータベース管理システム

リレーショナルデータベース

ID	名前	単価
1	みかん	50
2	りんご	100
3	りんご	150

ID	購入者	商品ID	数量
1	X	1	10
2	Y	2	5

たくさんの**テーブル**が格納される

あわせて

リレーショナルデータベースシステム

# リレーショナルデータベースは表計算ではない



## データベース

+

路線コード番号	事業者コード番号	路線名称一般	路線名称一般カナ
1	1001	3 中央新幹線	チュウオウシンカンセン
2	1002	3 東海道新幹線	トウカイドウシンカンセン
3	1003	4 山陽新幹線	サンヨウシンカンセン
4	1004	2 東北新幹線	トウホクシンカンセン

## データベース 管理システム

||

## データベース システム

データ共有, 検索, セキュリティ

## エクセルのデータ

+

	A	B	C	D
1	路線コード番号	事業者コード番号	路線名称一般	路線名称一般カナ
2	1001	3 中央新幹線	チュウオウシンカンセン	チュウオウシンカンセン
3	1002	3 東海道新幹線	トウカイドウシンカンセン	トウカイドウシンカンセン
4	1003	4 山陽新幹線	サンヨウシンカンセン	サンヨウシンカンセン
5	1004	2 東北新幹線	ジョウエイシンカンセン	ジョウエイシンカンセン

## エクセル (ソフトウェア)



||

## 表計算のシステム

表計算

# リレーショナルデータベースシステムの特徴



- データの形はテーブル（リレーションともいう）
- **SQL** の機能がある
- **データベース**の扱いは容易. 学びやすい
- データベース設計の基礎は体系化されている：**ERモデル, 異状, 従属, 正規化, 正規形**
- 普及度はナンバーワン
- **リレーショナルデータベース管理システム**にはさまざまある. MySQL, マイクロソフト Access, Oracle, SQL Server, PostgreSQL, SQLite3, Firebird など. （無料で使えるものもある）

# リレーショナルデータベースシステムの機能



	機能	SQL のキーワード
テーブル定義	テーブル定義	CREATE TABLE
	データ型	CHAR, TEXT, INTEGER, REAL, DATETIME, BIT, NULL
	自動インクリメント	AUTOINCREMENT
	主キー	PRIMARY KEY
	参照整合性制約	FOREIGN KEY, REFERENCES
問い合わせ (クエリ)	射影, 選択, 結合	SELECT FROM WHERE
	重複行除去 (分解でも)	DISTINCT
	比較, 範囲指定, パターンマッチ, AND/OR	=, <, >, <>, !=, <=, >=, BETWEEN, LIKE, AND, OR, IS NULL, IS NOT NULL
	集計・集約	GROUP BY, MAX, MIN, COUNT, AVG, SUM
	並べ替え (ソート)	ORDER BY
	副問い合わせ	IN
	データ操作	挿入, 削除, 更新
トランザクション	開始, コミット, ロールバック	BEGIN TRANSACTION, COMMIT, ROLLBACK

# dd-2. テーブルと テーブル定義

リレーショナルデータベースの  
基本（短縮版）（全7回）

基本を把握したい人へ

<https://www.kkaneko.jp/data/dd/index.html>

金子邦彦



謝辞：この資料では「かわいいフリー素材集 いらすとや」のイラストを使用しています

# 第2回のアウトライン



- テーブル
- テーブル定義
- データ型
- 主キー
- NULL
- 一貫性制約

# テーブルの例



テーブル名: products

id	name	price
1	orange	50
2	apple	100
3	melon	500

# テーブルの性質 ①

## リレーショナルデータベースならではの決まり事

テーブル名：福山駅行き

時	分
8	0
8	20
8	45
12	30
17	20
17	40

リレーショナルデータベースで  
扱えないテーブルの例

時	分
8	0, 20, 45
12	30
17	20, 40

- ☑ リレーショナルデータベースでは、1つのセルに1つの値

- ☐ 1つのセルに複数の値を入れることはない

# テーブルの性質 ②



## リレーショナルデータベースならではの決まり事

テーブル名：福山駅行き

リレーショナルデータベースで  
扱えないテーブルの例

時	分
8	0
8	20
8	45
12	30
17	20
17	40

時	分
8	0
	20
	45
12	30
17	20
	40

リレーショナルデータベース  
では、1つのセルに1つの値

マルチカラムにはしない

# テーブル定義



テーブル名: products

テーブル

id	name	price
1	orange	50
2	apple	100
3	melon	500

「id」と「name」と  
「price」の属性

# リレーショナルデータベースの構築手順



データベース  
設計



データベース  
生成

※ 最初、デー  
タベースは空



id	購入者	商品 ID	数量

id	name	price

テーブル定義

※ 最初、テーブルは空



id	購入者	商品 ID	数量
1	X	1	10
2	Y	2	5

id	name	price
1	orange	50
2	apple	100
3	melon	500

テーブル生成

# テーブル定義



テーブル名: products

テーブル定義では,

- ・ **テーブル名**
- ・ **属性の属性名**
- ・ **属性のデータ型**

などを設定して, **テーブル**を**定義**する

id	name	price
1	orange	50
2	apple	100
3	melon	500

```
CREATE TABLE products (  
  id INTEGER PRIMARY KEY NOT NULL,  
  name TEXT NOT NULL,  
  price REAL);
```

# 属性のデータ型



id	name	price
1	orange	50
2	apple	100
3	melon	500

属性名

テーブル  
の本体



整数

INTEGER



長いテキスト

TEXT



浮動小数点数

REAL

← SQL のキーワード

それぞれの属性のデータ型

# 属性のデータ型



主なデータ型	SQL のキーワード	
空値	NULL	
短いテキスト	CHAR	文字列
長いテキスト	TEXT	文字列
数値	INTEGER, REAL	整数や浮動小数 点数
日付／時刻	DATETIME	日付や時刻など
ブール値	BIT, BOOL	

※ **整数**は **INTEGER**, **浮動小数点数** (小数付きの数) は **REAL**

※ **短いテキスト**は半角 **255文字分**までが目安  
それ以上になる可能性があるときは**長いテキスト**

# 主キーと候補キー



- 候補キー

通し番号, 学生番号のように, 1つのテーブルの中で同じ値が2回以上出ないと前もって分かっている**属性**あるいは**属性の集まり**

- **主キー**

**候補キー**のうち管理に適するもの

id	name	price
1	orange	50
2	apple	100
3	melon	500

**主キー**

# リレーショナルデータベースの NULL



- **NULL** は「ヌル」あるいは「ナル」と読む
- **リレーショナルデータベース**で **NULL** は、次の場合に使う
  1. **未定, 未知, 不明** (分からない場合)
  2. **非存在** (もともと存在しない場合)

# テーブル定義と一貫性制約



## 【SQL プログラム】

```
CREATE TABLE products (  
  id INTEGER PRIMARY KEY NOT NULL,  
  name TEXT NOT NULL,  
  price REAL);
```

id:

**主キー** (PRIMARY KEY),  
**NULL になることはない** (NOT NULL)

name:

**NULL になることはない** (NOT NULL)

テーブルの制約について記述.

データベースの一貫性を維持するのに役立つ.

# dd-3. 種々の制約, 関連

リレーショナルデータベースの  
基本（短縮版）（全7回）

基本を把握したい人へ

<https://www.kkaneko.jp/data/dd/index.html>

金子邦彦



謝辞：この資料では「かわいいフリー素材集 いらすとや」のイラストを使用しています

# 第3回のアウトライン



- カラム制約
- テーブル制約
- 参照整合性制約
- 関連

# カラム制約, デフォルト値, 自動インクリメント

カラム制約は, 単一属性についての一貫性制約

## ◆ カラム制約

**PRIMARY KEY** 主キー

**NOT NULL** 非空

**UNIQUE** 一意

**FOREIGN KEY ... REFERENCES** 参照整合性制約

**CHECK** 更新時にチェック

※ CHECK は SQLite 3 固有の機能

## ◆ デフォルト値, 自動インクリメント

**DEFAULT** デフォルト値

**AUTOINCREMENT** 自動インクリメント (オートナンバー)

# テーブル制約



- 複数の属性に関わるような**一貫性制約**は**テーブル制約**の形で記述

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

## 【テーブル制約の書き方】

- **PRIMARY KEY** (<属性名の並び>) 主キー
- **UNIQUE** (<属性名の並び>) 一意
- **CHECK** (<式>) 更新時にチェック

※ CHECK は SQLite 固有の機能

# テーブルの例

テーブル名：**社員**

id	氏名	住所	所属
1	x	aaa	1
2	y	bbb	2
3	z	ccc	1

**x** と **z** は, **XX** に所属  
**y** は, **YY** に所属

テーブル名：**部**

id	部名	所在地
1	XX	AAAAA
2	YY	BBBBB

テーブル名: **products**

id	name	price
1	orange	50
2	apple	100
3	melon	500

# テーブル間の関連

- **テーブル**は互いに**関連**しあっている

テーブル名：**社員**      テーブル名：**部**      テーブル名：**products**



x と z は, XX に所属  
y は, YY に所属

- テーブル**社員**の**複数行**が, テーブル**部**の**1行**と**関連**
- テーブル**部**の**複数行**が, テーブル**products**の**複数行**と**関連**

# 関連の種類



- **一対一**

**テーブル**の**1**行が、別の**テーブル**の**1**行と関連

- **一対多**

**テーブル**の**1**行が、別の**テーブル**の**複数行**と関連

- **多対多**

**テーブル**の**複数行**が、別の**テーブル**の**複数行**と関連

# 参照整合性制約のイメージ



productsから  
お選びください

枝豆はないん  
ですか？



あるテーブルの**ある属性に格納できる**データには制約がつく  
場合がある

# 参照整合性制約



参照整合性制約の  
付いた属性

テーブル名：購入

id	購入者	商品ID	数量
1	X	1	10
2	Y	2	5

テーブル名：products

id	name	price
1	orange	50
2	apple	100
3	melon	500

テーブル「購入」の商品IDの値は、必ず、  
テーブル「products」のidの中から選ぶ

参照整合性制約

主キー

# テーブル定義の例



テーブル名：購入

id	購入者	商品ID	数量
1	X	1	10
2	Y	2	5

主キー

参照整合性制約

テーブル名：products

id	name	price
1	orange	50
2	apple	100
3	melon	500

主キー

```
CREATE TABLE 購入 (  
  id INTEGER PRIMARY KEY,  
  購入者 TEXT,  
  商品ID INTEGER,  
  数量 INTEGER,  
  FOREIGN KEY(商品ID) REFERENCES  
  products(id));
```

```
CREATE TABLE products (  
  id INTEGER PRIMARY  
  KEY NOT NULL,  
  name TEXT NOT NULL,  
  price REAL);
```

# dd-4. 問い合わせ と SQL

リレーショナルデータベースの  
基本（短縮版）（全7回）

基本を把握したい人へ

<https://www.kkaneko.jp/data/dd/index.html>

金子邦彦



謝辞：この資料では「かわいいフリー素材集 いらすとや」のイラストを使用しています

# 第4回のアウトライン



- 問い合わせ（クエリ）
- 問い合わせ（クエリ）の結果はテーブルである
- SQLによる問い合わせ（クエリ）の例
- SQLの特徴

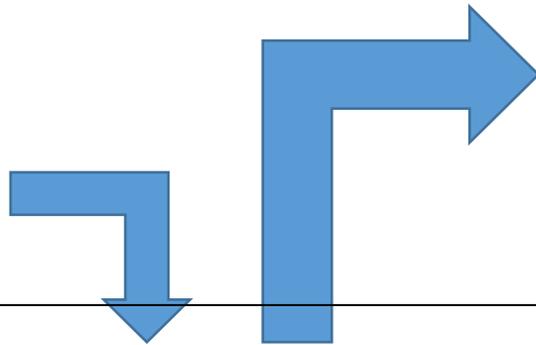
# 問い合わせ (クエリ)

- **問い合わせ (クエリ)** は、**データベースの検索**、**集計・集約**、**ソート (並べ替え)** などを行う
- **リレーショナルデータベースでの問い合わせ (クエリ)** の結果は、**テーブル形式のデータ**

# 問い合わせ（クエリ）の仕組み



問い合わせ  
（クエリ）  
のコマンド



リレーショナル  
データベースシステム

問い合わせ（クエリ）  
の結果は、**テーブル形式の**  
データ

ID	商品	単価
1	みかん	50
2	りんご	100
3	りんご	150

ID	購入者	商品ID	数量
1	X	1	10
2	Y	2	5

データの種類ごとに分かれ  
た、たくさんの**テーブル**

# SQLによる問い合わせの例



- ① **SELECT \* FROM** products;
- ② **SELECT** name, price **FROM** products;
- ③ **SELECT** name, price **FROM** products **WHERE** price > 80;

SQLは簡潔で単純

# SQL の特徴



- **SQL は、リレーショナルデータベースシステムの標準言語**
- 豊富な機能

問い合わせ (クエリ)	射影、選択、結合	SELECT FROM WHERE
	重複行除去 (分解でも)	DISTINCT
	比較, 範囲指定, パターンマッチ, AND/OR	=, <, >, <>, !=, <=, >=, BETWEEN, LIKE, AND, OR, IS NULL, IS NOT NULL
	集計・集約	GROUP BY, MAX, MIN, COUNT, AVG, SUM
	並べ替え (ソート)	ORDER BY
	副問い合わせ	IN

- 簡単簡潔
- コマンドなので、自動実行も簡単. あとからの確認も簡単

# SQL の利用イメージ



SQL の作成,  
編集, 実行



SQLプログラム 1

SQLプログラム 2

SQLプログラム 3



SQLコマンドのプログラムを  
準備しておくことも可能



リレーショナル  
データベースシステム

一般利用者は、リレーショナル  
データベースの利用で、  
SQLのことを意識しないことも多い

# dd-5. 貸出記録の テーブル

リレーショナルデータベースの  
基本（短縮版）（全7回）

基本を把握したい人へ

<https://www.kkaneko.jp/data/dd/index.html>

金子邦彦



# 第5回のアウトライン



次のことについて, Paiza.IO を用いた演習

- SQL によるテーブル定義
- SQL によるレコードの挿入
- SQL による問い合わせ (クエリ) の例

# 作成するテーブル



- 図書 (**book**) は, 次の3冊とする  
赤, 青, 緑
- 貸出者(**who**), 貸出か返却か(**what**), 日時(**at**)を記録する

## テーブル名: tosyo

book	who	what	at
赤	XX	貸出	2021-05-11 13:30:18
赤	XX	返却	2021-05-11 13:30:18
青	YY	貸出	2021-05-11 13:30:18
緑	ZZ	貸出	2021-05-11 13:30:18

実習では, at には, プログラム  
実行日時を記録する

# テーブル定義のSQL



```
CREATE TABLE tosyo (  
  book TEXT,  
  who TEXT,  
  what TEXT,  
  at DATETIME);
```

**赤, 青, 本**  
貸出者  
**貸出, 返却**  
日時

SQL のキーワード	
TEXT	文字列
DATETIME	日付や時刻など

# 新しいレコードの挿入



テーブル名: products

id	name	price
1	orange	50
2	apple	100
3	melon	500

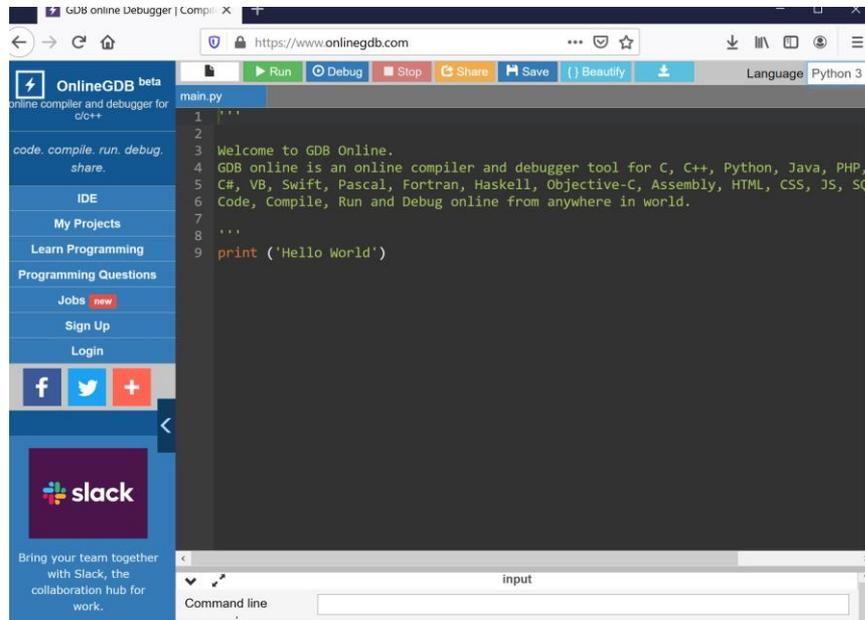


id	name	price
1	orange	50
2	apple	100
3	melon	500
4	apple	150

**INSERT INTO** products **VALUES**(4, 'apple', 150);

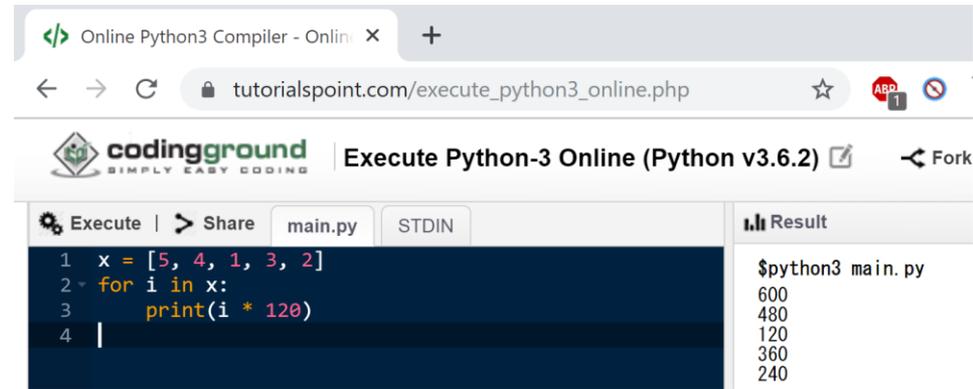
テーブル名 値の並び. 半角のカンマ「,」で区切る  
※ 文字列は半角の「'」で囲む

# プログラム作成ができるウェブサービス (オンラインの開発環境) の例



GDB online

<https://www.onlinegdb.com/>



Coding Ground

<https://www.tutorialspoint.com/codingground.htm>

# プログラム作成ができるウェブサービス (オンラインの開発環境) の例

A screenshot of the Paiza.IO web interface. The browser address bar shows "paiza.io/ja/projects/new". The page has a blue header with the Paiza.IO logo, a "Beta" badge, and navigation links for "新規コード" (New Code) and "一覧" (List). Below the header, there is a green "Python3" dropdown menu and a text input field with the placeholder "Enter a title here". A tab labeled "Main.py" is visible. The main content area is a dark code editor with the following text:

```
1 # coding: utf-8
2 # Your code here!
3
4
```

Paiza.IO

<https://paiza.io/>

# 実習



① ウェブブラウザを起動する

② 次の URL を開く

<https://paiza.io/>



③ もし、表示が英語になっていたら、日本語に切り替える



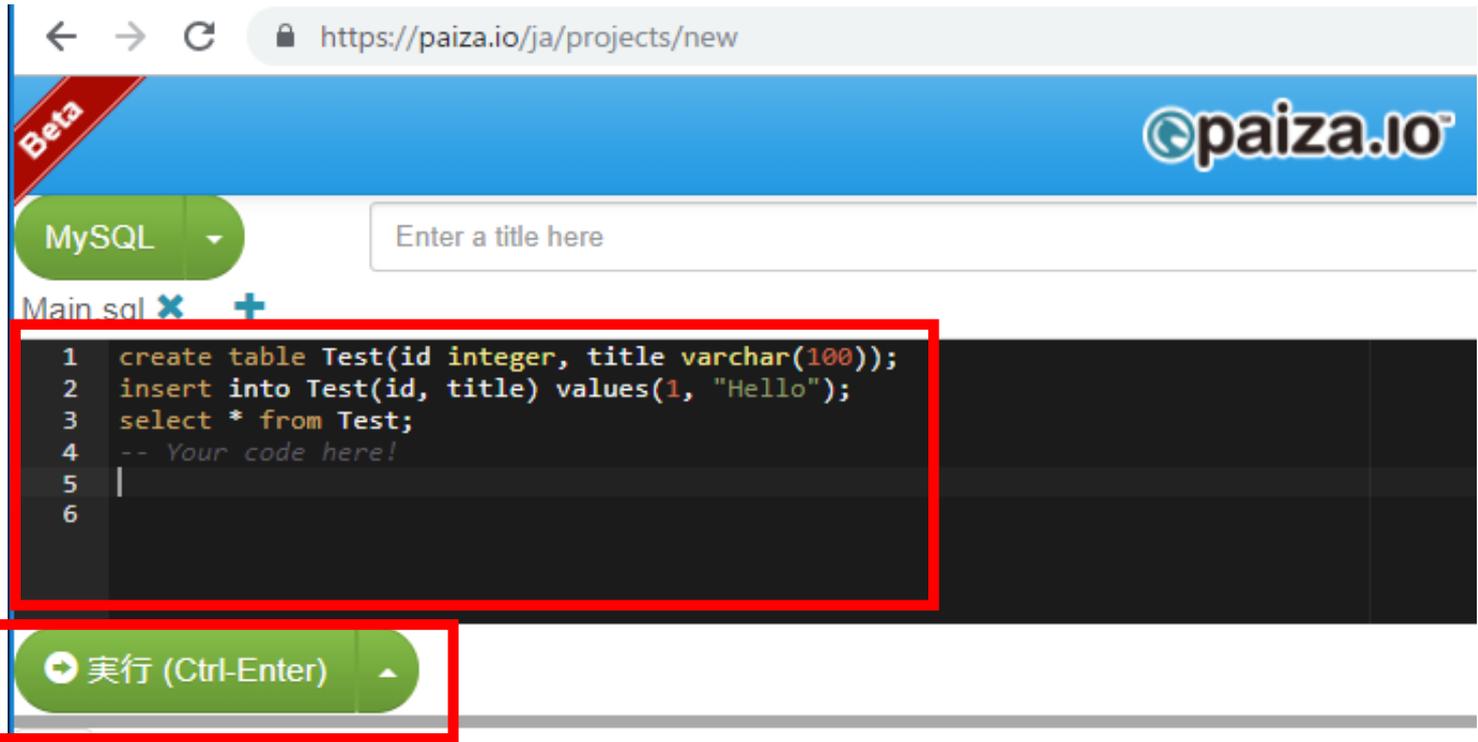
#### ④ 「コード作成を試してみる」をクリック



#### ⑤ 「MySQL」を選ぶ (左上のボタンをクリックするとproductsが出る)



プログラムの  
編集画面



MySQL

Main.sql

```
1 create table Test(id integer, title varchar(100));
2 insert into Test(id, title) values(1, "Hello");
3 select * from Test;
4 -- Your code here!
5 |
6
```

プログラムを  
書き換えること  
ができる

実行ボタン

編集画面を確認する。

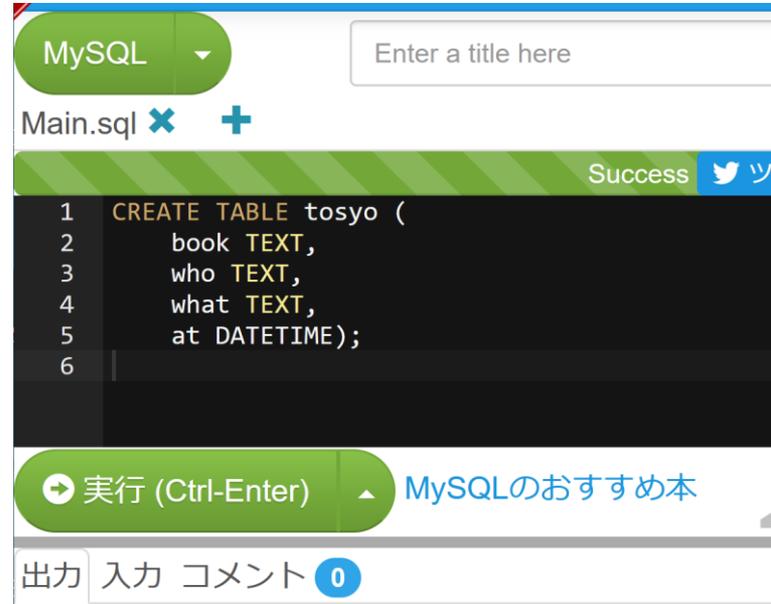
すでに、SQLが入っているが、使わないので消す。

```
1 create table Test(id integer, title varchar(100));
2 insert into Test(id, title) values(1, "Hello");
3 select * from Test;
4 -- Your code here!
5 |
6
```

# テーブル定義 tosyo

1から6行目に、次の **SQL** を書いて、「**実行**」をクリック。エラーメッセージが出ないことを確認。

```
CREATE TABLE tosyo (  
  book TEXT,  
  who TEXT,  
  what TEXT,  
  at DATETIME);
```

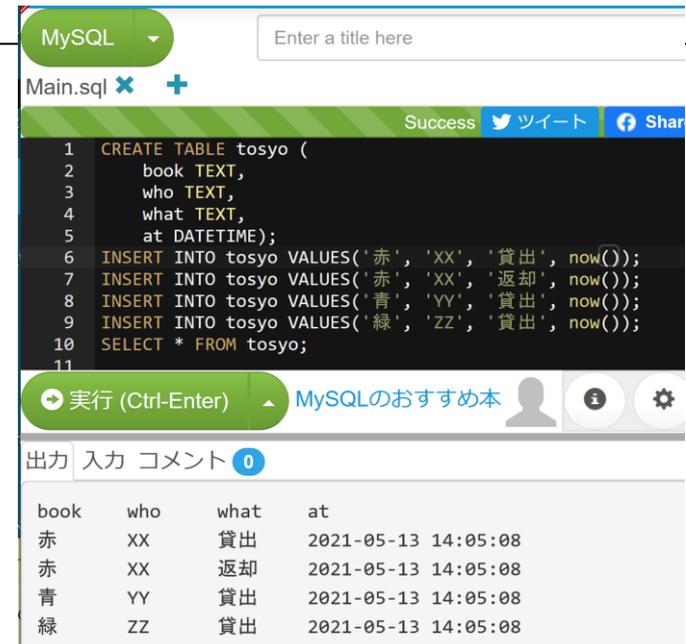


```
MySQL  
Enter a title here  
Main.sql x +  
Success ツ  
1 CREATE TABLE tosyo (  
2   book TEXT,  
3   who TEXT,  
4   what TEXT,  
5   at DATETIME);  
6  
実行 (Ctrl-Enter) MySQLのおすすめ本  
出力 入力 コメント 0
```

# テーブルへのレコードの挿入と確認

6から10行目に、次のSQLを書き加えて、「実行」をクリック。結果を確認

```
INSERT INTO tosyo VALUES('赤', 'XX', '貸出', now());  
INSERT INTO tosyo VALUES('赤', 'XX', '返却', now());  
INSERT INTO tosyo VALUES('青', 'YY', '貸出', now());  
INSERT INTO tosyo VALUES('緑', 'ZZ', '貸出', now());  
SELECT * FROM tosyo;
```



The screenshot shows a MySQL web interface with the following content:

- MySQL dropdown menu and "Enter a title here" input field.
- Tab for "Main.sql" with a close and add button.
- Success notification bar with "ツイート" and "Shar" buttons.
- SQL code editor showing 11 lines of code, including the INSERT and SELECT statements.
- Execution button labeled "実行 (Ctrl-Enter)" and a "MySQLのおすすめ本" link.
- Output table with columns: book, who, what, at.

book	who	what	at
赤	XX	貸出	2021-05-13 14:05:08
赤	XX	返却	2021-05-13 14:05:08
青	YY	貸出	2021-05-13 14:05:08
緑	ZZ	貸出	2021-05-13 14:05:08

now() は MySQL の機能で  
現在日時を取得  
(9時間遅れの世界標準時が取得  
されることもある)

# 貸し出し記録の集計・集約

11, 12行目に, 次の **SQL** を書き加えて, 「実行」をクリック. 結果を確認

```
SELECT who, COUNT(*) FROM tosyo GROUP BY who;  
SELECT COUNT(*) FROM tosyo WHERE what='貸出';
```

誰が何回貸出, 返却したか

```
SELECT who, COUNT(*) FROM tosyo GROUP BY who;
```

who	COUNT(*)
XX	2
YY	1
ZZ	1

貸出の回数は全部で何回か

```
SELECT COUNT(*) FROM tosyo WHERE what='貸出';
```

```
COUNT(*)  
3
```

# ここで使用した SQL



- テーブル定義

**CREATE TABLE ...**

- 問い合わせ

**SELECT ... FROM ...**

**SELECT ... FROM ... WHERE ...**

- レコードの挿入

**INSERT INTO ...**

# dd-6. さまざまな SQL 問い合わせ

リレーショナルデータベースの  
基本（短縮版）（全7回）

基本を把握したい人へ

<https://www.kkaneko.jp/data/dd/index.html>

金子邦彦



謝辞：この資料では「かわいいフリー素材集 いらすとや」のイラストを使用しています

# 第6回のアウトライン



次のことについて, Paiza.IO を用いた演習

- 問い合わせ (クエリ)
- 結合, 結合条件
- 並べ替え (ソート)
- 数え上げ
- 範囲指定
- 重複除去

# 実習



① ウェブブラウザを起動する

② 次の URL を開く

<https://paiza.io/>



③ もし、表示が英語になっていたら、日本語に切り替える



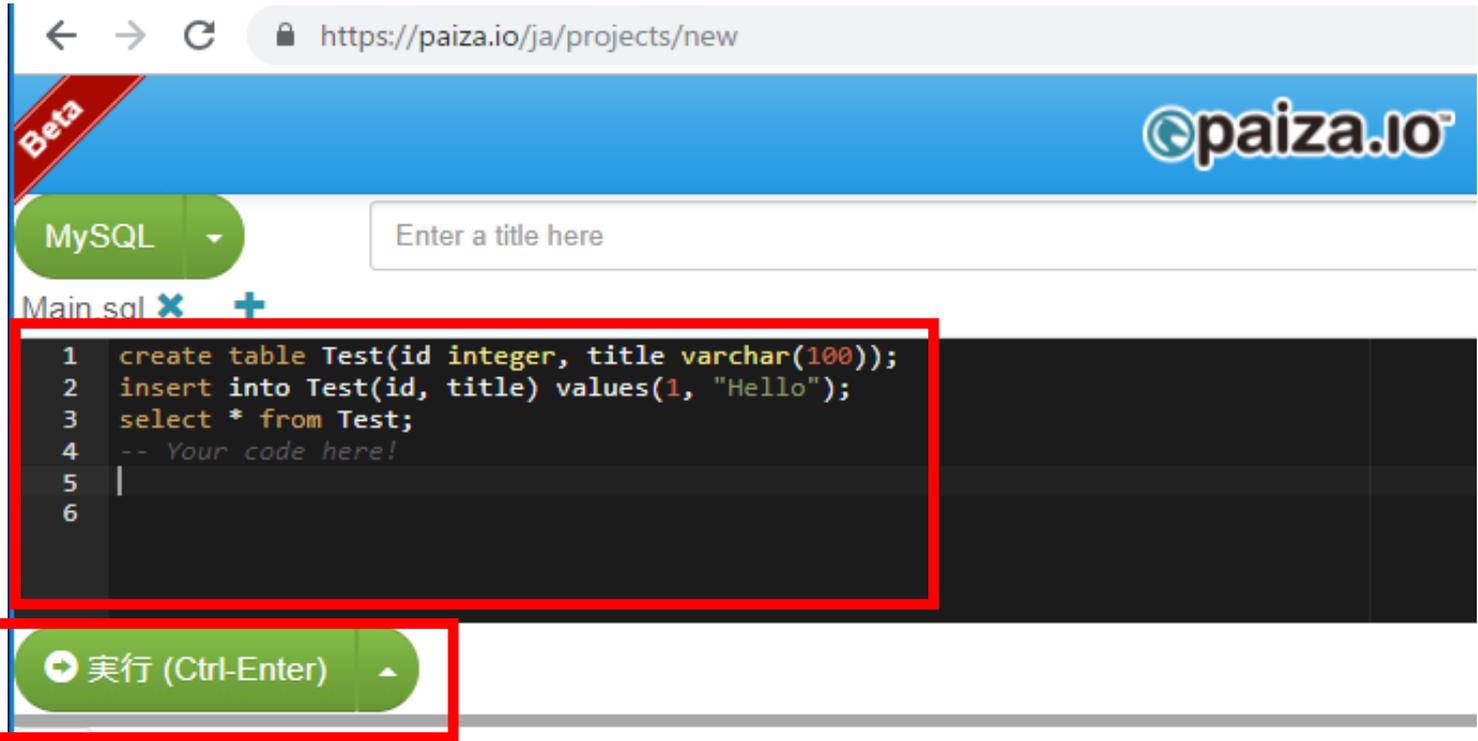
#### ④ 「コード作成を試してみる」をクリック



#### ⑤ 「MySQL」を選ぶ (左上のボタンをクリックするとproductsが出る)



プログラムの  
編集画面



MySQL

Enter a title here

Main sql

```
1 create table Test(id integer, title varchar(100));
2 insert into Test(id, title) values(1, "Hello");
3 select * from Test;
4 -- Your code here!
5 |
6
```

実行 (Ctrl-Enter)

プログラムを  
書き換えること  
ができる

実行ボタン

編集画面を確認する。

すでに、SQLが入っているが、使わないので消す。

```
1 create table Test(id integer, title varchar(100));
2 insert into Test(id, title) values(1, "Hello");
3 select * from Test;
4 -- Your code here!
5 |
6
```

# 作成するテーブル



テーブル名: products

id	name	price
1	orange	50
2	apple	100
3	melon	500

テーブル名: sales

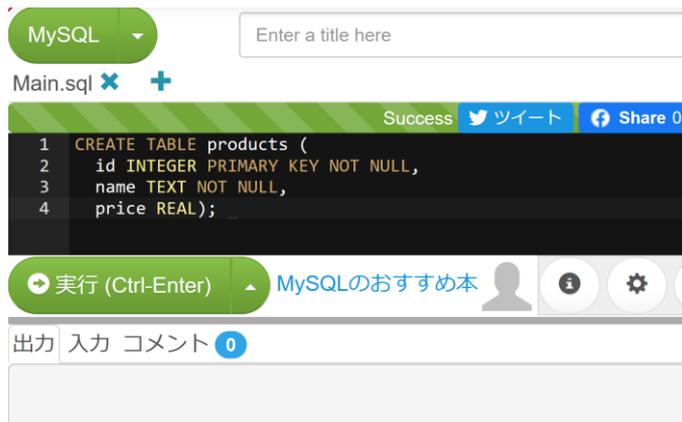
id	customer	pid	num
1	X	1	2
2	Y	1	3
3	X	3	1
4	Y	2	4

# テーブル定義 products

1から4行目に、次の **SQL** を書いて、「実行」をクリック。エラーメッセージが出ないことを確認

## 【SQL プログラム】

```
CREATE TABLE products (  
  id INTEGER PRIMARY KEY NOT NULL,  
  name TEXT NOT NULL,  
  price REAL);
```



MySQL

Main.sql

Success ツイート Share 0

```
1 CREATE TABLE products (  
2   id INTEGER PRIMARY KEY NOT NULL,  
3   name TEXT NOT NULL,  
4   price REAL);
```

実行 (Ctrl-Enter) MySQLのおすすめ本

出力 入力 コメント



## テーブル定義 sales

5から9行目に、次の **SQL** を書き加えて、「実行」をクリック。エラーメッセージが出ないことを確認

### 【SQLプログラム】

```
CREATE TABLE sales (  
  id INTEGER PRIMARY KEY NOT NULL,  
  customer TEXT NOT NULL,  
  pid INTEGER NOT NULL,  
  num INTEGER NOT NULL);
```

```
5 CREATE TABLE sales (  
6   id INTEGER PRIMARY KEY NOT NULL,  
7   customer TEXT NOT NULL,  
8   pid INTEGER NOT NULL,  
9   num INTEGER NOT NULL);
```

# レコードの挿入と確認



10から 13行目に, 次の **SQL** を書き加えて, 「実行」をクリック. 結果を確認

## 【SQLプログラム】

```
INSERT INTO products VALUES( 1, 'orange', 50 );  
INSERT INTO products VALUES( 2, 'apple', 100 );  
INSERT INTO products VALUES( 3, 'melon', 500 );  
SELECT * FROM products;
```

id	name	price
1	orange	50
2	apple	100
3	melon	500

# テーブルへのレコードの挿入と確認

14から 18行目に、次の **SQL** を書き加えて、「実行」をクリック。結果を確認

## 【SQL プログラム】

```
INSERT INTO sales VALUES(1, 'X', 1, 2);  
INSERT INTO sales VALUES(2, 'Y', 1, 3);  
INSERT INTO sales VALUES(3, 'X', 3, 1);  
INSERT INTO sales VALUES(4, 'Y', 2, 4);  
SELECT * FROM sales;
```

id	customer	pid	num
1	X	1	2
2	Y	1	3
3	X	3	1
4	Y	2	4

```
1 CREATE TABLE products (  
2   id INTEGER PRIMARY KEY NOT NULL,  
3   name TEXT NOT NULL,  
4   price REAL);  
5 CREATE TABLE sales (  
6   id INTEGER PRIMARY KEY NOT NULL,  
7   customer TEXT NOT NULL,  
8   pid INTEGER NOT NULL,  
9   num INTEGER NOT NULL);  
10 INSERT INTO products VALUES( 1, 'orange', 50 );  
11 INSERT INTO products VALUES( 2, 'apple', 100 );  
12 INSERT INTO products VALUES( 3, 'melon', 500 );  
13 SELECT * FROM products;  
14 INSERT INTO sales VALUES(1, 'X', 1, 2);  
15 INSERT INTO sales VALUES(2, 'Y', 1, 3);  
16 INSERT INTO sales VALUES(3, 'X', 3, 1);  
17 INSERT INTO sales VALUES(4, 'Y', 2, 4);  
18 SELECT * FROM sales;
```

いまからは、**1行目から18行目に入れたプログラムをそのまま残して実行する**

## 【SQL プログラム】

```
SELECT * FROM products;
```

id	name	price
1	orange	50
2	apple	100
3	melon	500

## 【SQL プログラム】

```
SELECT * FROM products WHERE name = 'orange';
```

id	name	price
1	orange	50

## 【SQL プログラム】

```
SELECT * FROM products WHERE price > 70;
```

id	name	price
2	apple	100
3	melon	500

## 【SQL プログラム】

```
SELECT * FROM products, sales;
```

id	name	price	id	customer	pid	num
3	melon	500	1	X	1	2
2	apple	100	1	X	1	2
1	orange	50	1	X	1	2
3	melon	500	2	Y	1	3
2	apple	100	2	Y	1	3
1	orange	50	2	Y	1	3
3	melon	500	3	X	3	1
2	apple	100	3	X	3	1
1	orange	50	3	X	3	1
3	melon	500	4	Y	2	4
2	apple	100	4	Y	2	4
1	orange	50	4	Y	2	4

行（レコード）の順序が違っている場合がある

## 【SQL プログラム】

```
SELECT * FROM products, sales
WHERE products.id = sales.pid;
```

id	name	price	id	customer	pid	num
1	orange	50	1	X	1	2
1	orange	50	2	Y	1	3
3	melon	500	3	X	3	1
2	apple	100	4	Y	2	4

# 並べ替え (ソート)



## 【SQLプログラム】

```
SELECT * FROM products ORDER BY price;
```

```
..  
id      name      price  
1       orange    50  
2       apple     100  
3       melon     500
```

# 並べ替え (ソート)



## 【SQL プログラム】

```
SELECT * FROM products ORDER BY price DESC;
```

id	name	price
3	melon	500
2	apple	100
1	orange	50

## 【SQL プログラム】

```
SELECT customer, COUNT(*) FROM sales  
GROUP BY customer;
```

customer	COUNT(*)
X	2
Y	2

## 【SQL プログラム】

```
SELECT * FROM products  
WHERE price BETWEEN 50 AND 200;
```

id	name	price
1	orange	50
2	apple	100

## 【SQL プログラム】

```
SELECT DISTINCT customer FROM sales;
```

customer

X

Y

# 問い合わせの結果をテーブルに保存



## 【SQL プログラム】

```
CREATE TABLE T AS  
SELECT DISTINCT name FROM products;  
SELECT * FROM T;
```

```
name  
orange  
apple  
melon
```

# dd-7. テーブルの 分解と結合

リレーショナルデータベースの  
基本（短縮版）（全7回）

基本を把握したい人へ

<https://www.kkaneko.jp/data/dd/index.html>

金子邦彦



謝辞：この資料では「かわいいフリー素材集 いらすとや」のイラストを使用しています

## 第7回のアウトライン

次のことについて, Paiza.IO を用いた演習

- テーブルの分解
- テーブルの結合

Paiza.IO の URL: <https://paiza.io/>

## テーブル定義 scores

1から6行目に、次の **SQL** を書いて、「実行」をクリック。エラーメッセージが出ないことを確認

### 【SQL プログラム】

```
CREATE TABLE scores (  
  id INTEGER PRIMARY KEY NOT NULL,  
  name TEXT NOT NULL,  
  teacher_name TEXT NOT NULL,  
  student_name TEXT NOT NULL,  
  score INTEGER);
```

```
1 CREATE TABLE scores (  
2   id INTEGER PRIMARY KEY NOT NULL,  
3   name TEXT NOT NULL,  
4   teacher_name TEXT NOT NULL,  
5   student_name TEXT NOT NULL,  
6   score INTEGER);
```

# レコードの挿入



7から11行目に、次の**SQL**を書き加え、「**実行**」をクリック。エラーメッセージが出ないことを確認

## 【SQLプログラム】

```
INSERT INTO scores VALUES(1, 'db', 'k', 'kk', 85);  
INSERT INTO scores VALUES(2, 'db', 'k', 'aa', 75);  
INSERT INTO scores VALUES(3, 'db', 'k', 'nn', 90);  
INSERT INTO scores VALUES(4, 'python', 'a', 'kk', 85);  
INSERT INTO scores VALUES(5, 'python', 'a', 'nn', 75);
```

```
7  INSERT INTO scores VALUES(1, 'db', 'k', 'kk', 85);  
8  INSERT INTO scores VALUES(2, 'db', 'k', 'aa', 75);  
9  INSERT INTO scores VALUES(3, 'db', 'k', 'nn', 90);  
10 INSERT INTO scores VALUES(4, 'python', 'a', 'kk', 85);  
11 INSERT INTO scores VALUES(5, 'python', 'a', 'nn', 75);
```

## 【SQL プログラム】

```
SELECT * FROM scores;
```

id	name	teacher_name	student_name	score
1	db	k	kk	85
2	db	k	aa	75
3	db	k	nn	90
4	python	a	kk	85
5	python	a	nn	75

## 【SQL プログラム】

```
SELECT name, teacher_name  
FROM scores;
```

name	teacher_name
db	k
db	k
db	k
python	a
python	a

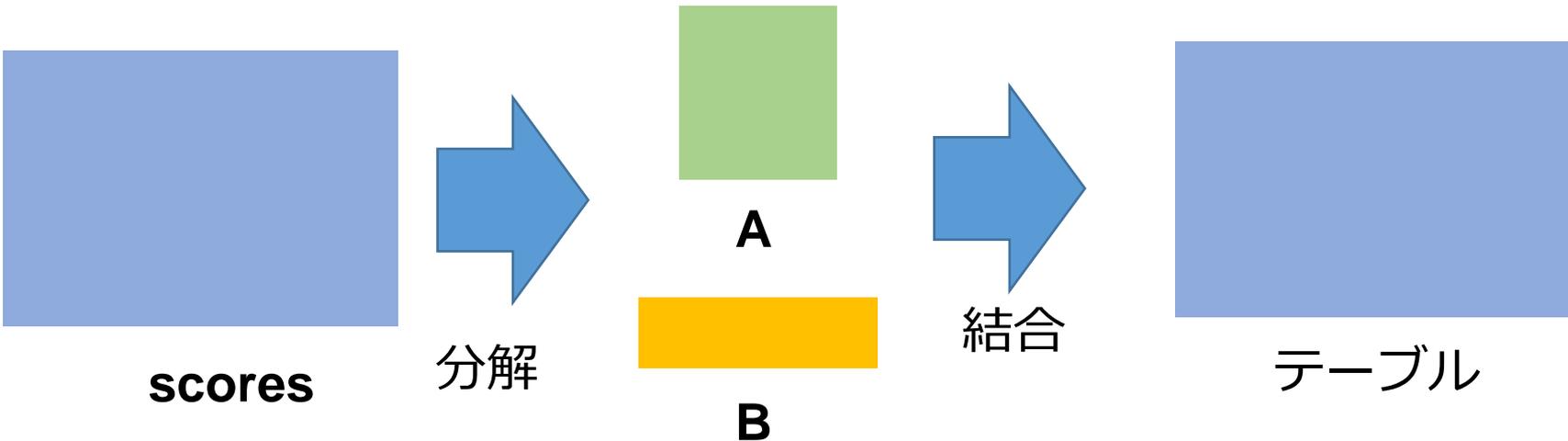
## 【SQL プログラム】

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

```
name      teacher_name  
db        k  
python    a
```

# テーブルの分解

いまから、テーブル scores を、テーブル A, B に**分解**する



**問い合わせの結果を、  
テーブルとして保存**

## テーブルへの保存の方法

マイクロソフト Access では: INTO

その他のシステム(世界標準): CREATE TABLE ... AS

# テーブルAの生成



## 【SQL プログラム】

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

```
name      teacher_name  
db        k  
python    a
```



# テーブル B の生成

テーブル A の生成の SQL を消さずに、次を書き加える

## 【SQL プログラム】

```
CREATE TABLE B AS  
SELECT DISTINCT id, name, student_name, score  
FROM scores;  
SELECT * FROM B;
```

id	name	student_name	score
1	db	kk	85
2	db	aa	75
3	db	nn	90
4	python	kk	85
5	python	nn	75



# テーブル A, B の結合

テーブル A の生成, テーブル B の生成の SQL を消さずに、次を書き加える

## 【SQL プログラム】

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

id	name	teacher_name	student_name	score
1	db	k	kk	85
2	db	k	aa	75
3	db	k	nn	90
4	python	a	kk	85
5	python	a	nn	75