15. SQL 演習

（データベース演習）

URL: https://www.kkaneko.jp/cc/de/index.html

金子邦彦

ここで使用する SQL

- **テーブル定義**

  CREATE TABLE ...

- **問い合わせ（クエリ）**

  SELECT ... FROM ...

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

- **行の挿入**

  INSERT INTO ...

- **問い合わせ結果の保存**

  CREATE TABLE AS ...

- **トランザクションの開始とロールバック**

  START TRANSACTION, ROLLBACK

15-1. SQL まとめ

データベースシステム

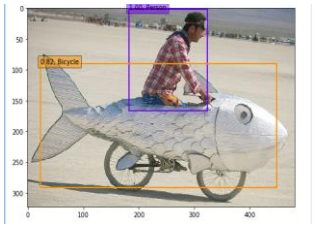
・データベース管理システムは，データベースの管理等の機能を持ったソフトウエア
・オンラインでデータを共有するときに，特に適する

取引

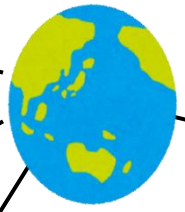
記入

データ保存

センサー連携



人工知能応用



ネットワーク

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

- **データベースシステム**の一種
- データの形は**テーブル**（**リレーション**ともいう）

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	name	price
1	orange	50
2	apple	100
3	melon	500

属性 id, name, price

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

属性 book, who, what, at

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システムの特徴

データの形は**テーブル**

- **機能が豊富**
- **扱いは容易．SQL を利用．**
- **リレーショナルデータベース設計**の基礎は**体系化されている**：**異状**，**正規化**
- **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）

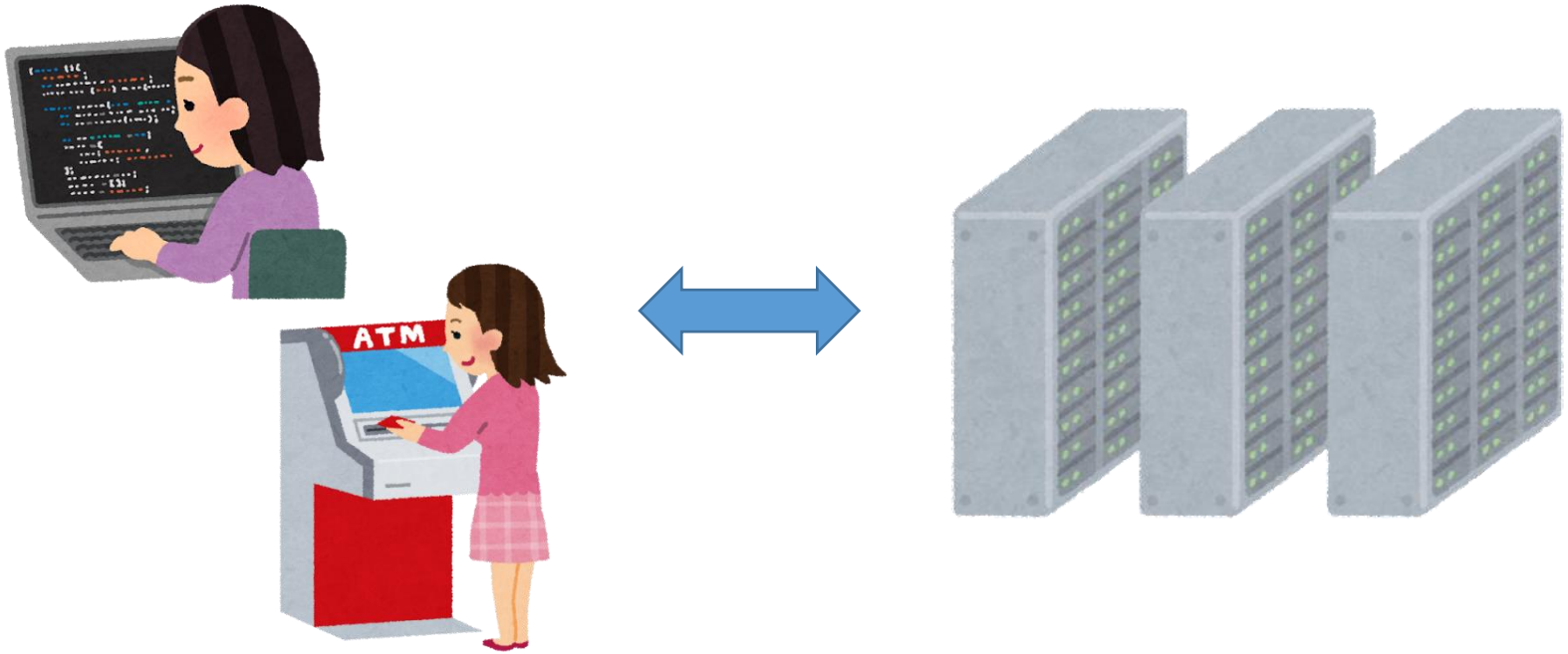
SQL

- **SQL** は、**リレーショナルデータベースシステム**の
  さまざまな機能を使える言語

  問い合わせ（クエリ）、

  テーブル定義、

  その他の操作



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
	オートナンバー	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	INSERT INTO, DELETE FROM WHERE, UPDATE SET WHERE
トランザクション	開始、コミット、ロールバック	BEGIN TRANSACTION, COMMIT, ROLLBACK

データベース設計の基礎: ER図, 異状, 従属, 正規化, 正規形

SQL の特徴

- 豊富な機能
- 簡単簡潔
- すべての**リレーショナルデータベース管理システム**で通用する**共通**言語

  リレーショナルデータベース管理システムの例

  Access, SQL Server, Oracle, MySQL, PostgreSQL,

  SQLite, Firebird, ・・・
- コマンドなので、自動実行も簡単．あとからの確認も簡単

問い合わせ（クエリ）

- 「**問い合わせ（クエリ）**」とは、

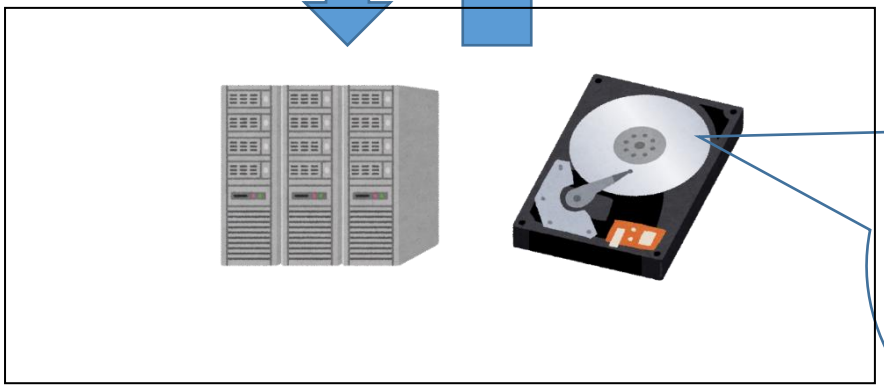
データベースの**検索**、**集計・集約**、**ソート（並べ替え**）を行うこと

- **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

問い合わせ（クエリ）の結果は，テーブル形式のデータ



リレーショナルデータベースシステム

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

		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

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（クエリ）の例

① SELECT * FROM 商品;

② SELECT 名前, 単価 FROM 商品;

③ SELECT 名前, 単価 FROM 商品 WHERE 単価 > 80;

④ SELECT 受講者, COUNT(*) FROM 成績 GROUP BY 受講者;

⑤ SELECT * FROM 米国成人調査データ ORDER BY 年齢;

⑥ SELECT * FROM T, S;

⑦ SELECT * FROM T, S WHERE a = b;

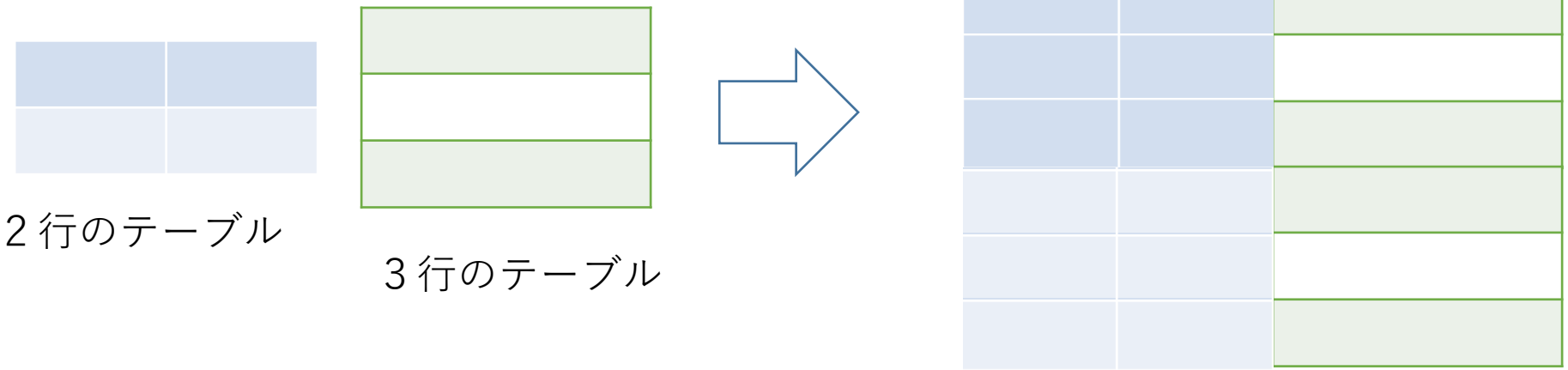
⑧ SELECT * FROM 授業 WHERE 教室名 IN ('一階', '二階');

⑨ SELECT DISTINCT 学生番号 FROM 成績 WHERE 科目名
IN (SELECT 科目名 FROM 成績 WHERE 学生番号 = 101);

結合と結合条件

- **結合**：　2 つのテーブルを1つにまとめる

SELECT * FROM S, T　・・・ 結合条件なしで S と T を**結合**



2 行のテーブル

3 行のテーブル

2 × 3 で，6 行のテーブル

- **結合条件**

SELECT * FROM S, T　WHERE a = b;

・・・ 結合条件は「a = b」

結合はどういう場合に役に立つのか

違うテーブルに分かれているデータを，**１つにまとめたい**とき

- **結合**は**２つのテーブルを１つにまとめる**
- **結合**を繰り返すことにより，３つ以上のテーブルを１つにまとめることも可能

「商品」と「購入」の関連

商品

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500

購入

購入者	商品番号
X	1
X	3
Y	2

Xさんは、**1**の**みかん**と，
3の**メロン**を買った
Yさんは、**2**の**りんご**を買った

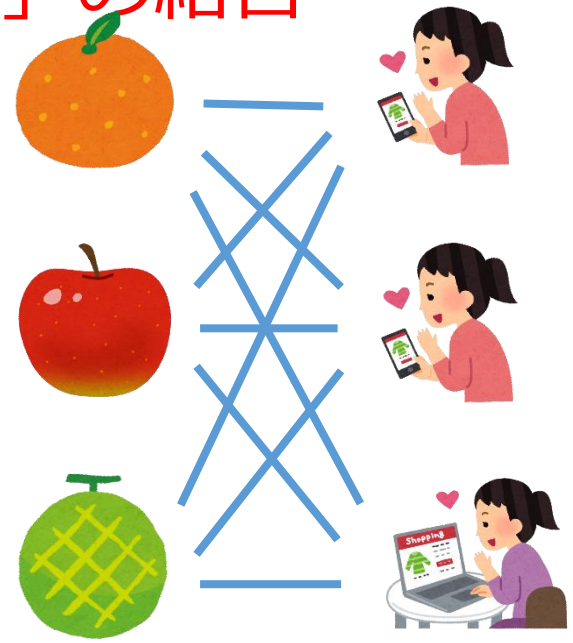
購入テーブルの情報

商品テーブルの情報

「商品」と「購入」の結合

商品

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500



購入者	商品番号
X	1
X	3
Y	2

ペアは９通り

結合の結果　⇒
（結合条件が無い場合）

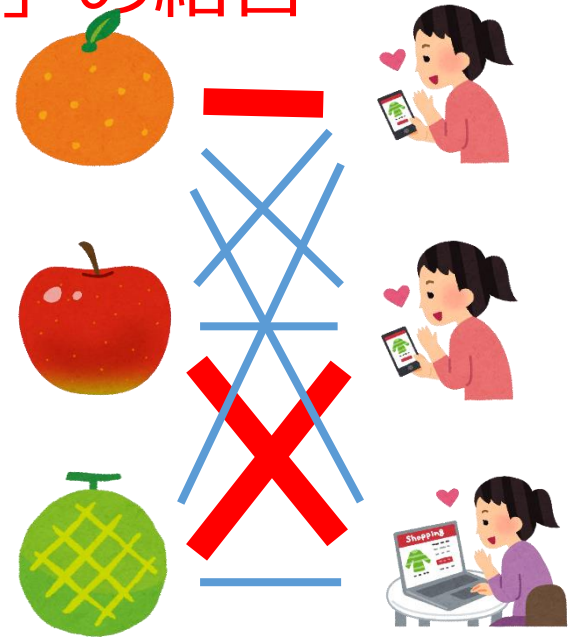
ID	商品名	単価	購入者	商品番号
1	みかん	50	X	1
1	みかん	50	X	3
1	みかん	50	Y	2
2	りんご	100	X	1
2	りんご	100	X	3
2	りんご	100	Y	2
3	メロン	500	X	1
3	メロン	500	X	3
3	メロン	500	Y	2

```
SELECT * FROM 商品, 購入;
```

「商品」と「購入」の結合

商品

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500



購入者	商品番号
X	1
X	3
Y	2

結合条件があると
選択が行われる ⇒

ID	商品名	単価	購入者	商品番号
1	みかん	50	X	1
2	りんご	100	Y	2
3	メロン	500	X	3

```
SELECT * FROM 商品, 購入
WHERE ID = 商品番号;
```

SQL を用いた新しい行の挿入

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

ロールバックとは

- **ロールバック**は，データベースを，**トランザクション開始時点**に**戻すこと**
- **リレーショナルデータベース管理システム**の**標準機能**
- **ロールバックしたトランザクションだけが元に戻る（他の利用者に影響を与えることはない）**

ロールバック (rollback)

操作１、操作２、操作３
と操作していて、
最初に戻したくなった

ユーザ

トランザクション開始

操作１

操作２

操作３

rollback

データベース管理システム

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	名前	単価

テーブル定義

「こういうテーブルを使いたい」と設定するだけなので、テーブルは**空**

ID	購入者	商品ID	数量
1	X	1	10
2	Y	2	5

ID	名前	単価
1	みかん	50
2	りんご	100
3	りんご	150

テーブル生成

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

テーブル名: tosyo

book	who	what	at
赤	XX	貸出	2023-05-11 13:30:18
赤	XX	返却	2023-05-11 13:30:18
青	YY	貸出	2023-05-11 13:30:18
緑	ZZ	貸出	2023-05-11 13:30:18

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

属性のデータ型

book	who	what	at
赤	XX	貸出	2023-05-11 13:30:18
赤	XX	返却	2023-05-11 13:30:18
青	YY	貸出	2023-05-11 13:30:18
緑	ZZ	貸出	2023-05-11 13:30:18
TEXT	TEXT	TEXT	DATETIME

属性名

テーブル
の本体

それぞれの**属性**の**データ型**

属性のデータ型

Access の主なデータ型	SQL のキーワード	
	NULL	空値
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
Yes／No	**bit, bool**	ブール値

※ **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**

15-2. 演習（Paiza.IO を使用）

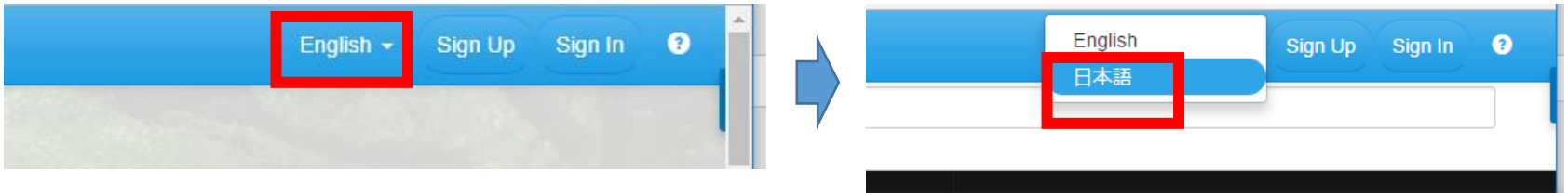
Paiza.IO の使い方

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



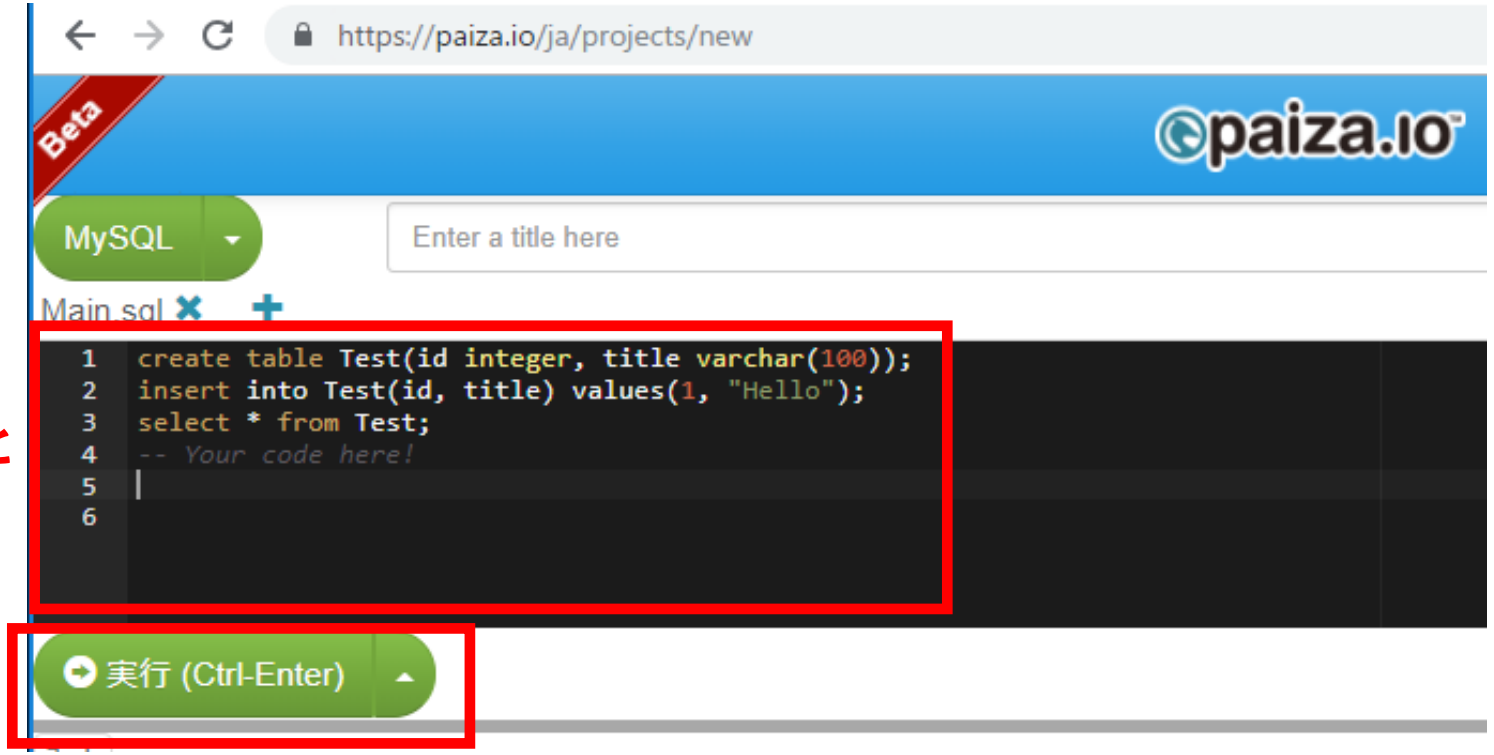
④「コード作成を試してみる」をクリック

⑤「**MySQL**」を選ぶ（**左上のボタンをクリック**するとメニューが出る）

プログラムの
編集画面

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
```

使用するテーブル

テーブル products

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

テーブル sales

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

ここで使用する SQL

- **テーブル定義**

  CREATE TABLE ...

- **問い合わせ（クエリ）**

  SELECT ... FROM ...

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

- **行の挿入**

  INSERT INTO ...

- **問い合わせ結果の保存**

  CREATE TABLE AS ...

- **トランザクションの開始とロールバック**

  START TRANSACTION, ROLLBACK

テーブル定義 products

１から5行目に、次の **SQL を入れ**、「実行」をクリック。
エラーメッセージが出ないことを確認
※　エラーメッセージが出たときは、SQLを修正してから、
　　再度実行する

```
1  create table products (
2    id integer,
3    name text,
4    price integer
5  ) engine = innoDB;
```

○ engine = innoDB; は，MySQLでトランザクションの機能を有効にするために付けている

データ型
テキスト（文字列） **text**
数値 **integer**, **real**

ここで，補足説明

SQLプログラムは**消さず**に，
下に書き加えるようにしてください

資料のスクリーンショットでは「行番号」
も付けているので，参考にしてください

テーブル定義 sales

6から 11行目に、次の **SQL を入れ**、「実行」をクリック。
エラーメッセージが
出ないことを確認

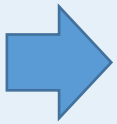
```
 6  create table sales (
 7    id integer,
 8    customer text,
 9    pid integer,
10    num integer
11  ) engine == innoDB;
```

○ engine = innoDB; は， MySQLで
トランザクションの機能を有効に
するために付けている

ここで，補足説明

行の挿入

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500



ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500
4	レモン	80

```
insert into 商品 values(4, 'レモン', 80);
```

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

新しい行の挿入

12から 20行目に、次の **SQL を入れ**、「実行」をクリック。確認

```
12  insert into products values(1, 'orange', 50);
13  insert into products values(2, 'apple', 100);
14  insert into products values(3, 'melon', 500);
15  insert into sales values(1, 'X', 1, 2);
16  insert into sales values(2, 'Y', 1, 3);
17  insert into sales values(3, 'X', 3, 1);
18  insert into sales values(4, 'Y', 2, 4);
19  select * from products;
20  select * from sales;
```

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

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

結合（2つのテーブルのすべてのペア）

① 次の SQL 問い合わせを実行し確認

```
21  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（行）の順序が違っている場合がある

結合（結合条件あり）

② 次の SQL 問い合わせを実行し確認

```
23  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）

③ 次の SQL 問い合わせを実行し確認．昇順.

```
24  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）

④ 次の SQL 問い合わせを実行し確認．降順.

```
25  select * from products order by price desc;
```

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

数え上げ

⑤ 次の SQL 問い合わせを実行し確認

```
26  select customer, count(*) from sales group by customer;
```

customer	count(*)
X	2
Y	2

範囲指定

⑥ 次の SQL 問い合わせを実行し確認

```
27  select * from products where price between 50 and 200;
```

id	name	price
1	orange	50
2	apple	100

重複除去

⑦ 次の SQL 問い合わせを実行し確認

```
28  select distinct customer from sales;
```

```
customer
X
Y
```

ここで，補足説明

次のSQL は，SQL 問い合わせ（クエリ）の結果を，**新しいテーブルに保存**する

```
create table T as select name from products where name = 'orange';
```

【書き方】

create table ＜テーブル名＞ as ＜SQL 問い合わせ＞

問い合わせの結果をテーブルに保存

⑧ 次の SQL 問い合わせを実行し確認

```
29  create table T as select name from products where name = 'orange';
30  select * from T;
```

name
orange

ロールバック

⑨ 次の SQL 問い合わせを実行し確認

```
31  start transaction;
32  insert into sales values(5, 'Z', 1, 1);
33  select * from sales;
34  rollback;
35  select * from sales;
```

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

rollback は,
start transaction 以降の
データベース操作を取り消す

15-3. テーブルの分解と結合

- テーブルの分解と結合の演習

- 今まで入れた SQL プログラムは，すべて消しても問題ありません

① テーブル定義

次の **SQL を入れる.**

```
1  create table scores (
2    id integer,
3    name text,
4    teacher_name text,
5    student_name text,
6    score integer
7  );
```

データ型

テキスト（文字列） **text**

数値 **integer**, **real**

② 実行

エラーメッセージが出ないことを確認. 表示はない.

※　エラーメッセージが出たときは、SQLを修正してから、**再度**実行する

③ 行の挿入

次の **SQL** を**書き加える**.

```
 8  insert into scores values(1, 'database', 'k', 'kk', 85);
 9  insert into scores values(2, 'database', 'k', 'aa', 75);
10  insert into scores values(3, 'database', 'k', 'nn', 90);
11  insert into scores values(4, 'programming', 'a', 'kk', 85);
12  insert into scores values(5, 'programming', 'a', 'nn', 75);
```

④ 実行

エラーメッセージが出ないことを確認. 表示はない.

※　エラーメッセージが出たときは、SQLを修正してから、**再度**実行する

⑤ 問い合わせ

次の **SQL** を**書き加える**.

```
13  select * from scores;
```

⑥ 実行

確認.

※　エラーメッセージが出たときは、SQLを修正してから、**再度**実行する

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

⑦ 問い合わせ

次の **SQL** を**書き加える**.

```
14  select name, teacher_name from scores;
```

⑧ 実行

確認.

※　エラーメッセージが出たときは、SQLを修正してから、**再度**実行する

name	teacher_name
database	k
database	k
database	k
programming	a
programming	a

⑨ 重複行除去

次の **SQL** を**書き加える**.

```
14  select distinct name, teacher_name from scores;
```

⑩ 実行

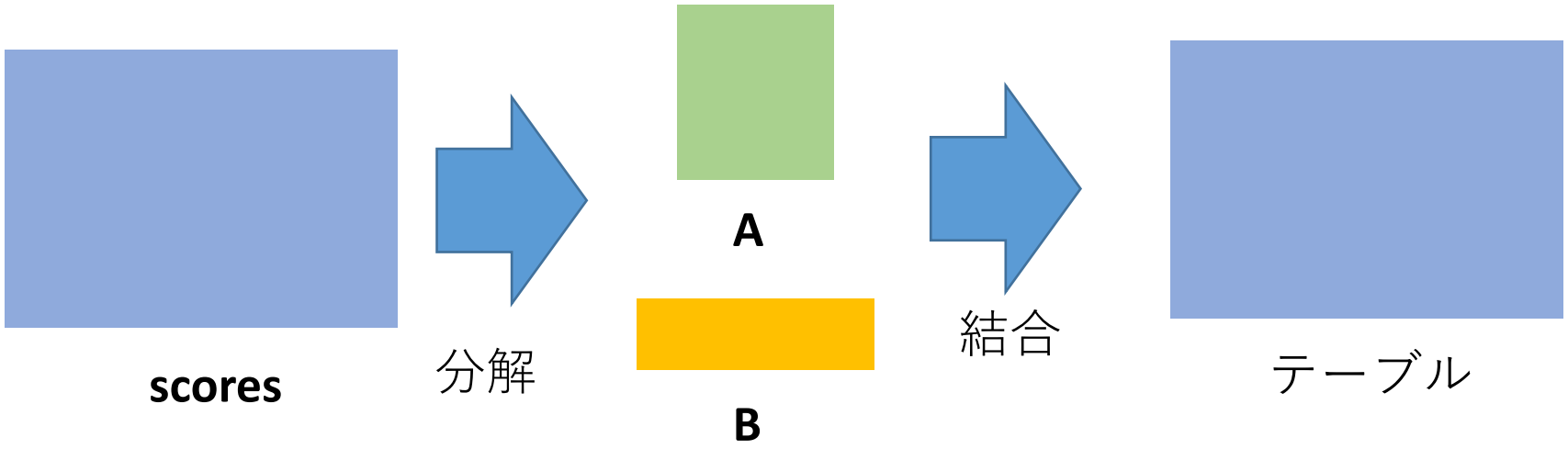
確認.

※　エラーメッセージが出たときは、SQLを修正してから、**再度**実行する

name	teacher_name
database	k
programming	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

Access: INTO
その他のシステム(世界標準): create table … as

⑪ テーブルの分解のため，テーブル A の作成

次の **SQL** を**書き加える**.

```
15  create table A as select distinct name, teacher_name from scores;
16  select * from A;
```

⑫ 実行

確認.

※　エラーメッセージが出たときは、SQLを修正してから、
再度実行する

name	teacher_name
database	k
programming	a

⑬ テーブルの分解のため，テーブル B の作成

次の **SQL** を**書き加える**.

```
17  create table B as select distinct id, name, student_name, score from scores;
18  select * from B;
```

⑭ 実行

確認.

※　エラーメッセージが出たときは、SQLを修正してから、**再度**実行する

id	name	student_name	score
1	database	kk	85
2	database	aa	75
3	database	nn	90
4	programming	kk	85
5	programming	nn	75

⑮ テーブル A, B の結合

次の **SQL** を**書き加える**.

```
19  select B.id, A.name, A.teacher_name, B.student_name, B.score
20  from A, B
21  where A.name = B.name;
```

⑯ 実行

確認.

※　エラーメッセージが出たときは、SQLを修正してから、**再度**実行する

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