ic-1. リレーショナルデータベースの基本まとめ

（データベース）

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

金子邦彦

BY NC SA

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

アウトライン

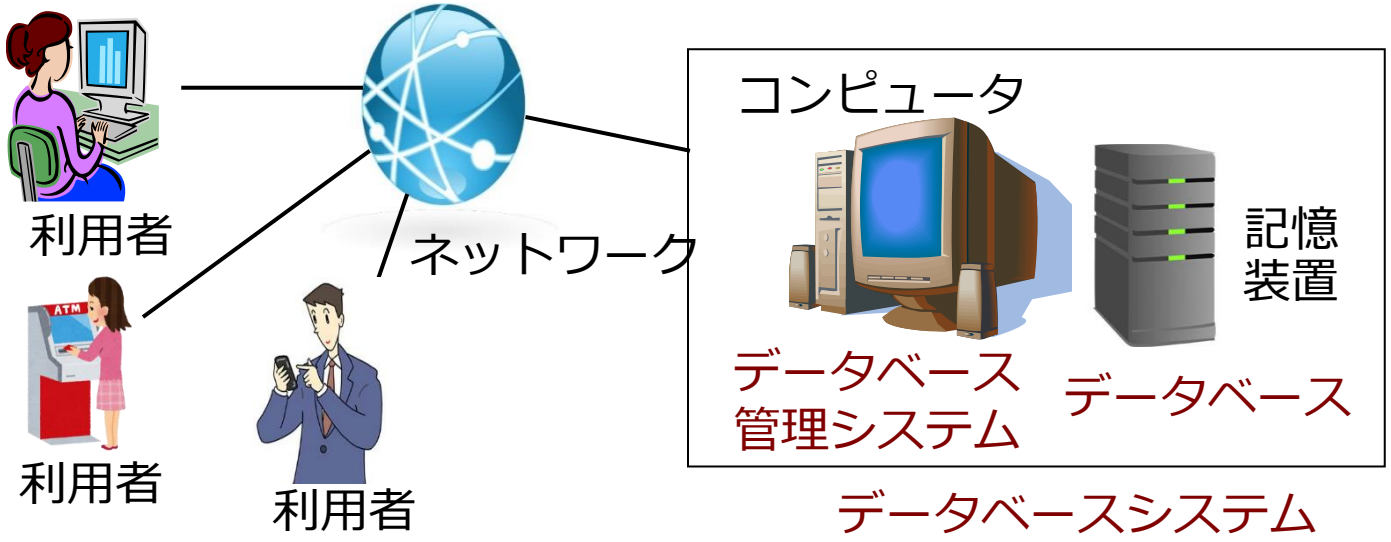
1. リレーショナルデータベースシステムの概要
2. SQL
3. SQL の実行
4. データベースの設計

1. 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

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（ソフトウエア）



データベースシステム

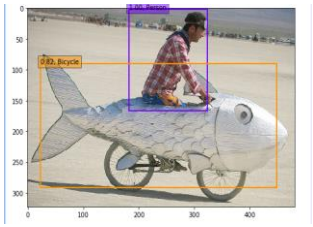
・データベース管理システムは，データベースの管理等の機能を持ったソフトウエア
・オンラインでデータを共有するときに，特に適する

取引

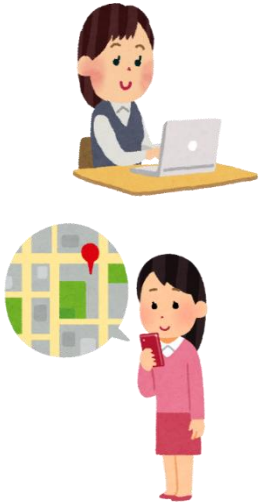
記入

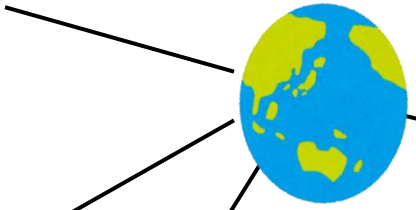
データ保存

センサー連携



人工知能応用





データベースシステム

ネットワーク

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
3	1002	3	東海道新幹線	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
5	1004	2	東北新幹線	ジョウエツシンカンセン

＋

エクセル
（ソフトウエア）

||

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**

- **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）

テーブル定義

テーブル

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

「**book**」と「**who**」と
「**what**」と「**at**」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

属性名

テーブルの本体

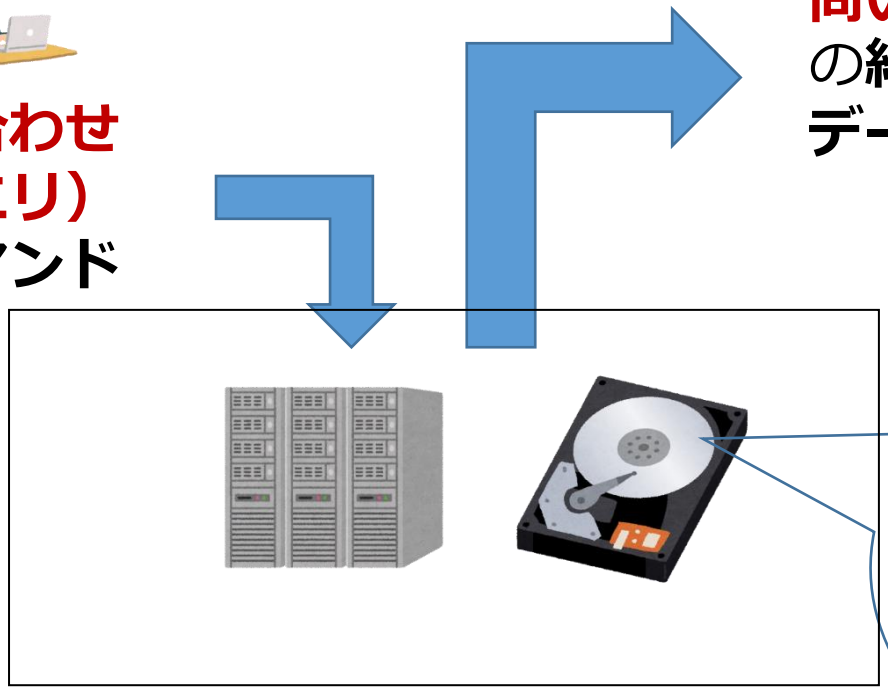
TEXT **TEXT** **TEXT** **DATETIME**

それぞれの**属性**の**データ型**

2. SQL

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**

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**

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）

- 「**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**

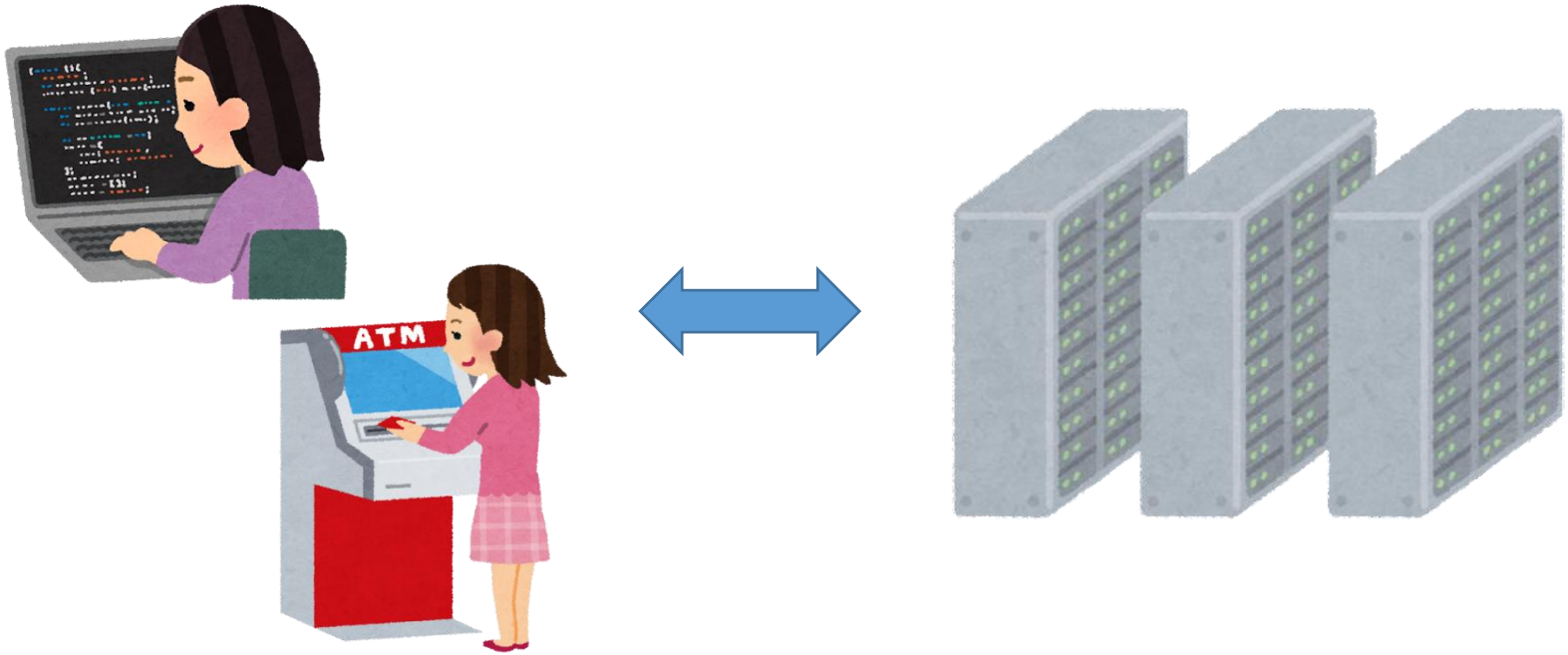
SQL

- **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



SQL の特徴

- **SQL** は，**リレーショナルデータベースシステム**の**<u>標準言語</u>**
- 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

- さまざまな機能を組み合わせて使うことができる
- 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 による問い合わせ（クエリ）の例

誰が何回貸出，返却したか

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

```
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';

```
3
```

SQLは簡潔で単純

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

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 プログラムを準備しておき，
呼び出すことが可能.
・他のアプリの中に SQL プログラム
を埋め込むことが可能

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

リレーショナル
データベースシステム

3. SQL の実行

アウトライン

- SQL による**テーブル定義**
- SQL による**行の挿入**
- SQL による**問い合わせ（クエリ）**

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	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

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	日時，日付，時刻など

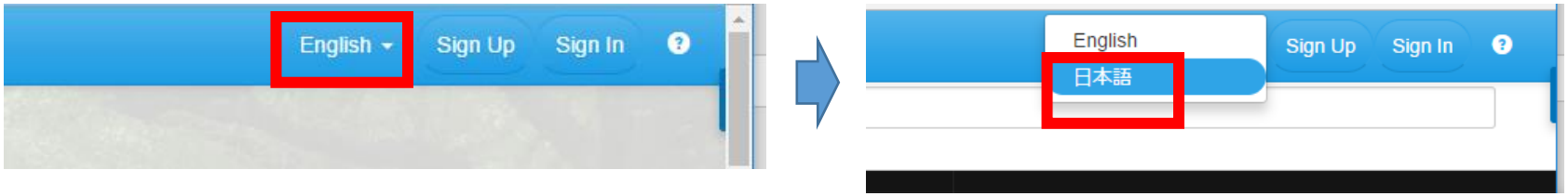
Paiza.IO の使い方

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

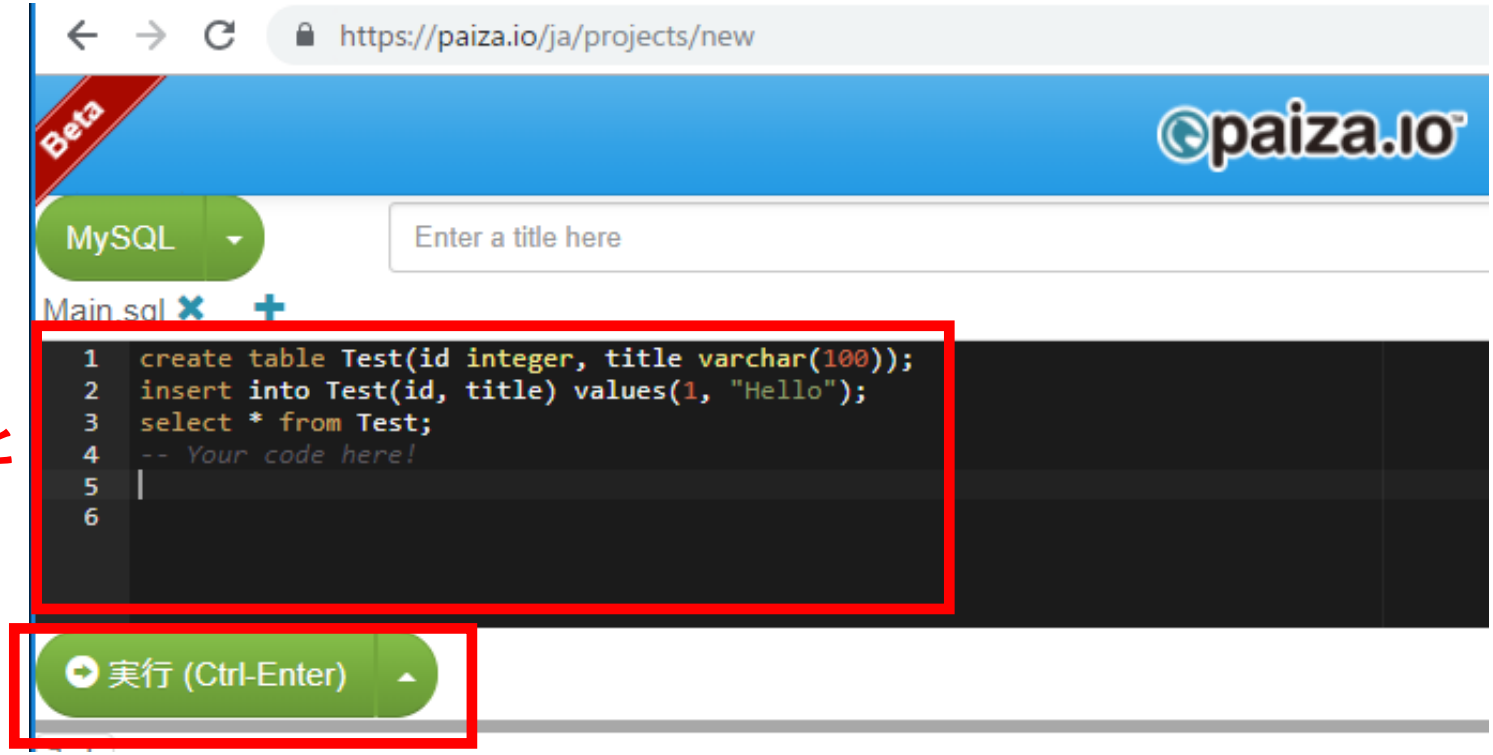


④ 「コード作成を試してみる」をクリック

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
```

テーブル定義

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

行の挿入と確認

```
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;
```

NOW() は MySQL の機能で
現在日時の取得
（９時間遅れの世界標準時）

集計・集約

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;

```
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';

```
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** ...

- 行の挿入

  INSERT INTO ...

4. リレーショナルデータベースの設計

はじめに

- **データベース**で一番困ることは，データベースの**異状**である.

【異状とは】

データベース内のデータが、

◆　**つじつまの合わない状態**

あるいは

◆　**記録したいデータが記録できない状態**

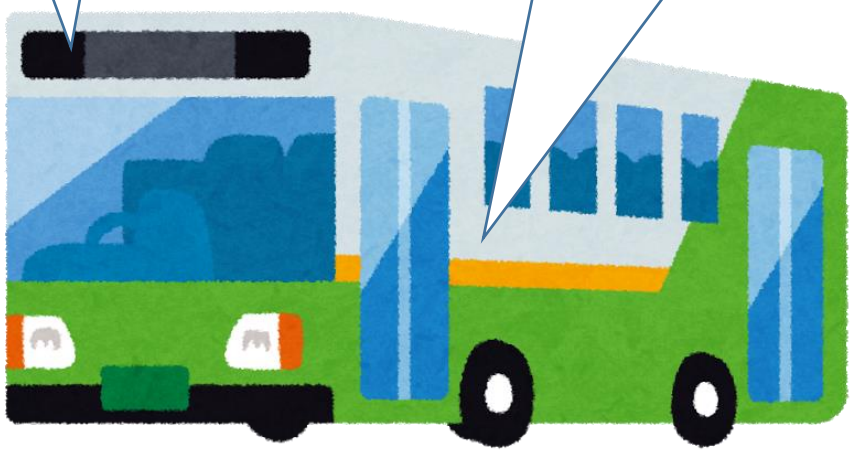
になり、しかも、修復できない状態に陥ること

- **異状**は完全に防止できるわけではないが，正規化により，ある程度防ぐことができる

異状の例

このバスは無料です

このバスは運賃１０００円です



異状が起きやすいリレーショナルデータベースの例

名前	朝食	値段
A	カレーライス	400
B	うどん	250
C	カレーライス	400

テーブル

- **カレーライス**は、４００円
- **うどん**は、２５０円
- **A**さんは**カレーライス**を食べた
- **B**さんは**うどん**を食べた
- **C**さんは**カレーライス**を食べた

情報

異状が起きやすいリレーショナルデータベースの例

情報の更新

カレーライスが
400円から 350円に値下げ

名前	朝食	値段
A	カレーライス	~~400~~ 350
B	うどん	250
C	カレーライス	~~400~~ 350

テーブル

- **カレーライス**は、 ~~４００~~ 350 円
- **うどん**は、 ２５０円
- **A**さんは**カレーライス**を食べた
- **B**さんは**うどん**を食べた
- **C**さんは**カレーライス**を食べた

情報

異状が起きやすいリレーショナルデータベースの例

情報の更新

カレーライスが
400円から 350円に値下げ

名前	朝食	値段
A	カレーライス	400
B	うどん	250
C	カレーライス	~~400~~ 350

書き換え忘れして
しまうかも！

テーブル

異状が起きやすいリレーショナルデータベースの例

情報の更新

カレーライスが
400円から 350円に値下げ

名前	朝食	値段
A	カレーライス	**400**
B	うどん	250
C	カレーライス	**350**

異状が起きている

テーブル

◆朝食の値段が１つのはずなのに、違った値段が記録されていてつじつまが合わない

テーブル分解

名前	朝食	値段
A	カレーライス	400
B	うどん	250
C	カレーライス	400

分解

名前	朝食
A	カレーライス
B	うどん
C	カレーライス

朝食	値段
カレーライス	400
うどん	250

分解後、情報は**失われていない**

分解後のテーブルは，異状が起きにくい

名前	朝食
A	カレーライス
B	うどん
C	カレーライス

朝食	値段
カレーライス	~~400~~ 350
うどん	250

テーブル

カレーライスが

400円から 350円に値下げ

異状はない

設計変更による異状の防止

名前	朝食	値段
A	カレーライス	~~400~~ 350
B	うどん	250
C	カレーライス	~~400~~ 350

異状が起きやすいデータベース

冗長なデータがある

カレーライスの値下げのとき、
片方を書き忘れると → 異状

情報は同じ

名前	朝食
A	カレーライス
B	うどん
C	カレーライス

朝食	値段
カレーライス	~~400~~ 350
うどん	250

異状が起きにくいデータベース

冗長なデータがない

１つのテーブルを、**複数**の**テーブルに分解**することで、**異状の防止ができる**場合がある.