9. データベース設計の実践

正規化の目的と手順、種々の正規形、SQLを用いた正規化

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

金子邦彦

謝辞：この資料では「いらすとや」のイラストを使用しています

9. データベース設計の実践

正規化の目的と手順、種々の正規形、SQLを用いた正規化

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

金子邦彦

謝辞：この資料では「いらすとや」のイラストを使用しています

① 理論と実践の両立

② データベース設計の重要性の理解

③ データベース運用能力の向上

アウトライン

1. イントロダクション
2. データベース設計の概要
3. 正規化
4. 問い合わせ結果からのテーブル生成
5. SQL を用いた正規化
6. 正規形のバリエーション

SQLFiddle のサイトにアクセス

Webブラウザを使用

1. ウェブブラウザを開く
2. アドレスバーにSQLFiddleのURLを入力

http://sqlfiddle.com/

URLが分からないときは、Googleなどの**検索エンジン**を利用。
「**SQLFiddle**」と**検索**し、表示された結果からSQLFiddleの
ウェブサイトをクリック。

SQLFiddle の画面

- **左側のパネル: テーブル定義、データの追加**など。SQLのCREATE TABLE や INSERT INTO などを入力。

- **右側のパネル: SQL問い合わせ。**SELECT などを入力。

SQL Fiddle MySQL 5.6

```
CREATE TABLE employees (
    id INTEGER,
    name TEXT,
    age INTEGER,
    salary INTEGER,
    department_id INT);
INSERT INTO employees VALUES (1, 'Alice', 30, 50000, 1);
INSERT INTO employees VALUES (2, 'Bob', 40, 60000, 1);
INSERT INTO employees VALUES (3, 'Charlie', 35, 70000, 2);
```

Build Schema | Edit Fullscreen | Browser | [;]

```
SELECT * FROM employees;
```

Run SQL | Edit Fullscreen | [;]

- **実行ボタン**

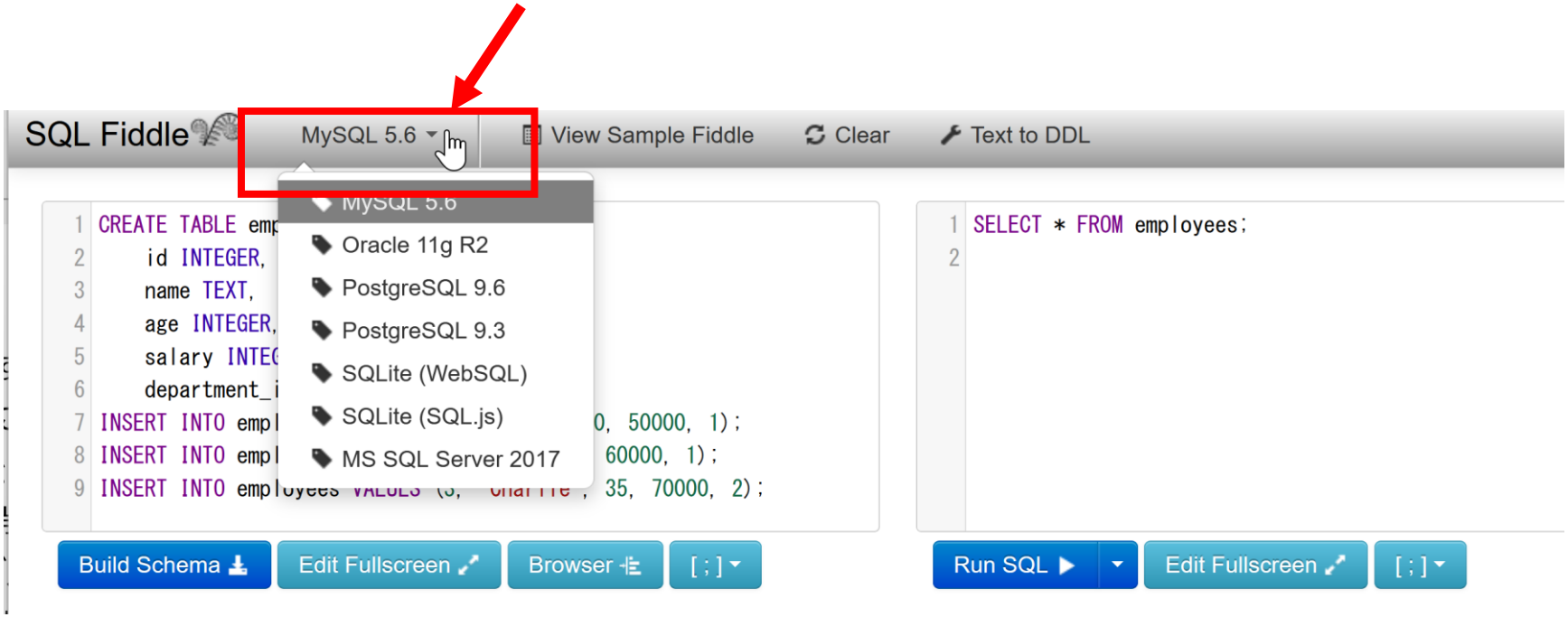
- **実行ボタン**

id	name	age	salary	department_id
1	Alice	30	50000	1
2	Bob	40	60000	1
3	Charlie	35	70000	2

結果ウィンドウ

SQLFiddle でのデータベース管理システムの選択（高度な機能）

データベース管理システムの選択
（この授業では MySQL を使用）



① 理論と実践の両立

② データベース設計の重要性の理解

③ データベース運用能力の向上

アウトライン

1. イントロダクション
2. データベース設計の概要
3. 正規化
4. 問い合わせ結果からのテーブル生成
5. SQL を用いた正規化
6. 正規形のバリエーション

SQLFiddle のサイトにアクセス

Webブラウザを使用

1. ウェブブラウザを開く
2. アドレスバーにSQLFiddleのURLを入力

http://sqlfiddle.com/

URLが分からないときは、Googleなどの**検索エンジン**を利用。
「**SQLFiddle**」と**検索**し、表示された結果からSQLFiddleの
ウェブサイトをクリック。

SQLFiddle の画面

- **左側のパネル: テーブル定義、データの追加**など。SQLのCREATE TABLE や INSERT INTO などを入力。

- **右側のパネル: SQL問い合わせ。**SELECT などを入力。

SQL Fiddle MySQL 5.6

```
CREATE TABLE employees (
    id INTEGER,
    name TEXT,
    age INTEGER,
    salary INTEGER,
    department_id INT);
INSERT INTO employees VALUES (1, 'Alice', 30, 50000, 1);
INSERT INTO employees VALUES (2, 'Bob', 40, 60000, 1);
INSERT INTO employees VALUES (3, 'Charlie', 35, 70000, 2);
```

```
SELECT * FROM employees;
```

Build Schema | Edit Fullscreen | Browser | [;]

Run SQL | Edit Fullscreen | [;]

- **実行ボタン**

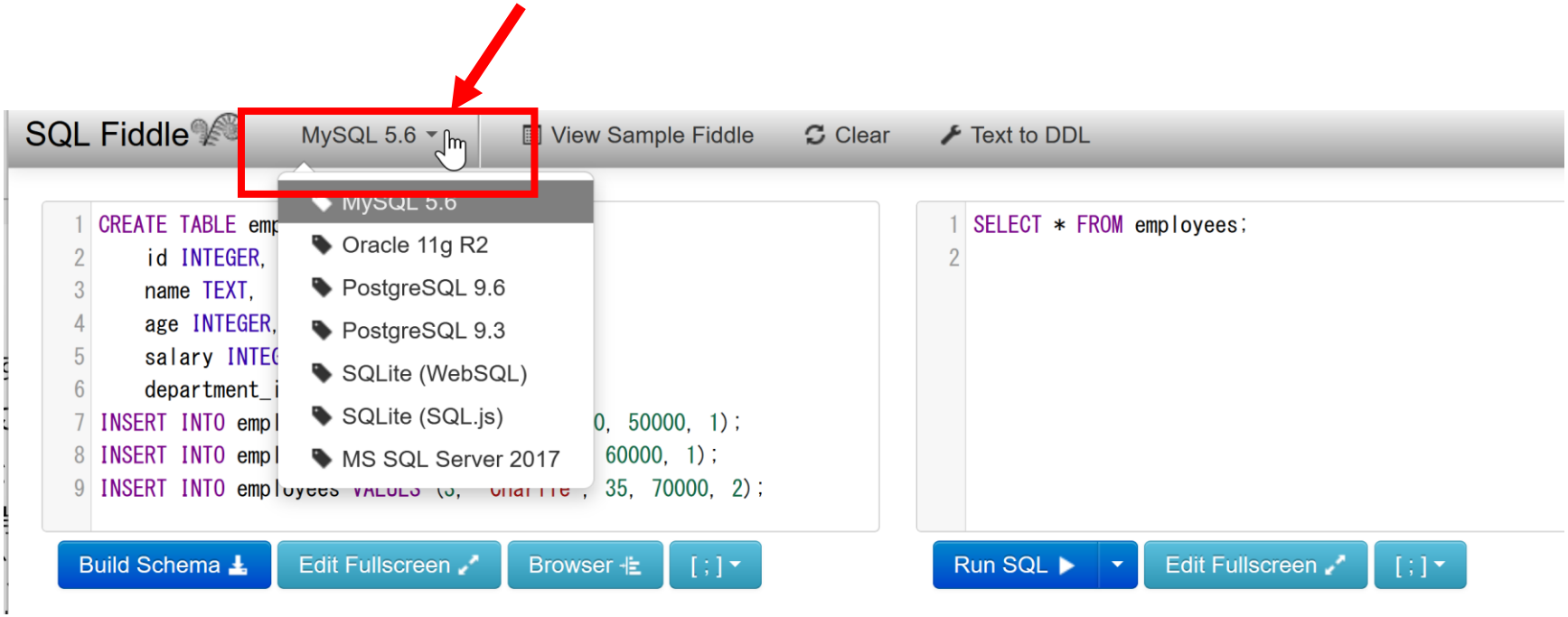
- **実行ボタン**

id	name	age	salary	department_id
1	Alice	30	50000	1
2	Bob	40	60000	1
3	Charlie	35	70000	2

結果ウィンドウ

SQLFiddle でのデータベース管理システムの選択（高度な機能）

データベース管理システムの選択
（この授業では MySQL を使用）



9-1. イントロダクション

リレーショナルデータベースの仕組み

- データを**テーブル**と呼ばれる**表形式で保存**
- **テーブル間**は**関連**で結ばれる。複雑な構造を持ったデータを効率的に管理することを可能に。

商品

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

関連

購入

購入者	商品番号
X	1
X	3
Y	2

商品テーブルと購入テーブル

商品

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

関連

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

結合の例

商品

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

関連

購入

購入者	商品番号
X	1
X	3
Y	2

- 商品テーブルと購入テーブルを**結合**して、**購入者がどの商品を購入したかのデータ**を取得。
- **結合条件**は、**商品テーブルのID属性と購入テーブルの商品番号属性が等しい**場合に結合

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

```
SELECT * FROM 商品
JOIN 購入
ON 商品.ID = 購入.商品番号;
```

結合の例

商品

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

関連

購入

購入者	商品番号
X	1
X	3
Y	2

結合のためのSQL

```
SELECT * FROM 商品
JOIN 購入
ON 商品.ID = 購入.商品番号;
```

結合条件

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

SQLの実行により
新しく生成される
テーブル

結合条件の指定

結合のためのSQL

```
SELECT * FROM 商品
JOIN 購入
ON 商品.ID = 購入.商品番号;
```

} **結合条件**

- **商品テーブルの「ID」と購入テーブルの「商品番号」属性が等しい**という結合条件

商品.ID = 購入.商品番号

- **「等しい値を持つ」という結合条件の表し方**

テーブル1.属性3 = テーブル2.属性4

リレーショナルデータベースの重要性

1. **データの整合性**: リレーショナルデータベースは、**データの整合性を保持するための機能**を有する。これにより、誤ったデータや矛盾したデータが保存されるのを防ぐことができる。
2. **柔軟な問い合わせ（クエリ）能力**: リレーショナルデータベースの**SQL**（Structured Query Language）の使用により、**複雑な検索やデータの抽出**が可能になる。
3. **トランザクション**の機能: 一連の操作全体を一つの単位として取り扱うことができる機能。これにより、**データの一貫性と信頼性が向上**する。
4. **セキュリティ**: **アクセス権限の設定**などにより、セキュリティを確保。

データの安全な保管、効率的なデータ検索・操作、ビジネスや研究の意思決定をサポート。

リレーショナルデータベースの設計における考慮事項

次のことを考慮して設計を行う

- **データの冗長性を排除**する**正規化**
- **データの整合性を保証**する**制約**

  例：関連するテーブル間の参照に関する制約

  例：同じ値が２度現れないという制約

正規化とその重要性

- **正規化**は、**リレーショナルデータベースのテーブルを適切な形に再構成**することで、**データの冗長性を排除**し、**データの整合性**を向上させるプロセス。
- 正規化は、データベース設計において欠かせない

正規化の例

正規化前

名前	昼食	料金
A	そば	250
B	**カレーライス**	**400**
C	**カレーライス**	**400**
D	うどん	250

冗長なデータがある

正規化後

名前	昼食
A	そば
B	カレーライス
C	カレーライス
D	うどん

昼食	料金
そば	250
カレーライス	400
うどん	250

冗長なデータがない

正規化により、元のテーブルにあった**冗長性を排除.**

正規化前の問題

更新

カレーライスが
400円から 350円に値下げ

名前	昼食	料金
A	そば	250
B	カレーライス	~~400~~ **350**
C	カレーライス	400
D	うどん	250

「400」をすべて「350」に変更する必要があるが、変更を間違ったとする

昼食の値段が１つのはずなのに、350, 400 の違った値段の記録があり、不整合がある

正規化による問題解消

カレーライスが
400円から 350円に値下げ

更新

名前	昼食
A	カレーライス
B	うどん
C	カレーライス

昼食	値段
カレーライス	~~400~~ **350**
うどん	250

「400」をすべて「350」に変更するのは1か所で済む。
間違いが起きにくい。

正規化の例

生徒名	クラス	教科	成績
田中	3年A組	数学	85
田中	3年A組	英語	90
佐藤	3年B組	数学	88
佐藤	3年B組	英語	92

正規化

生徒ID	生徒名	クラス
1	田中	3年A組
2	佐藤	3年B組

生徒ID	教科	成績
1	数学	85
1	英語	90
2	数学	88
2	英語	92

正規化により、元のテーブルにあった**冗長性を排除.**

SQL によるテーブル定義

- **テーブル名**：**T**
- **属性名**：**名前、昼食、料金**
- **属性のデータ型**：**テキスト、テキスト、数値**
- データの整合性を保つための**制約**：**なし**

```
CREATE TABLE T (
名前 TEXT,
昼食 TEXT,
料金 INTEGER);
```

演習のため、正規化前のテーブルを作成

データ追加のSQL

名前	昼食	料金
A	そば	250
B	カレーライス	400
C	カレーライス	400
D	うどん	250

```
INSERT INTO T VALUES('A', 'そば', 250);
INSERT INTO T VALUES('B', 'カレーライス', 400);
INSERT INTO T VALUES('C', 'カレーライス', 400);
INSERT INTO T VALUES('D', 'うどん', 250);
```

演習のため、正規化前のテーブルを作成

DISTINCT は重複行の除去

SELECT 昼食 **FROM** T;

結果

昼食
そば
カレーライス
カレーライス
うどん

SELECT DISTINCT 昼食 **FROM** T

結果

昼食
そば
カレーライス
うどん

演習１．テーブル定義とデータの追加

【トピックス】

1. SQL によるテーブル定義
2. SQL によるデータの追加
3. 問い合わせ（クエリ）による確認

Webブラウザを使用

① アドレスバーにSQLFiddleのURLを入力

http://sqlfiddle.com/

URLが分からないときは、Googleなどの**検索エンジン**を利用。「**SQLFiddle**」と**検索**し、表示された結果からSQLFiddleのウェブサイトをクリック。

② **左側のパネル**に、**テーブル定義**と**データの追加**を行う SQL を入れる。

```
CREATE TABLE T (
名前 TEXT,
昼食 TEXT,
料金 INTEGER);
INSERT INTO T VALUES('A', 'そば', 250);
INSERT INTO T VALUES('B', 'カレーライス', 400);
INSERT INTO T VALUES('C', 'カレーライス', 400);
INSERT INTO T VALUES('D', 'うどん', 250);
```

③ 「Build Schema」をクリック

SQL Fiddle　MySQL 5.6　View Sample Fiddle　Clear　Text to DDL

```
CREATE TABLE T (
名前 TEXT,
昼食 TEXT,
料金 INTEGER);
INSERT INTO T VALUES('A', 'そば', 250);
INSERT INTO T VALUES('B', 'カレーライス', 400);
INSERT INTO T VALUES('C', 'カレーライス', 400);
INSERT INTO T VALUES('D', 'うどん', 250);
```

Build Schema　Edit Fullscreen　Browser　[;]

Please build schema.

Run SQL　Edit Fullscreen　[;]

Please build schema.

④ **右側のパネル**に、**問い合わせ（クエリ）**を行う SQL を入れる。

```
SELECT * FROM T;
```

⑤ 「**Run SQL**」をクリック

SQL 文が**実行**され、結果が表示される。

⑥ 下側のウインドウで、**結果を確認。**

⑦ **右側のパネル**に、**問い合わせ（クエリ）**を行う SQL を入れる。

```
SELECT 昼食 FROM T;
SELECT DISTINCT 昼食 FROM T;
```

⑧ 「**Run SQL**」をクリック

SQL 文が**実行**され、結果が表示される。

⑨ 下側のウインドウで、**結果を確認。**

自習１．データの追加

目的: 新しい行をテーブルに追加

テーブルTに新しい行である「E ラーメン 500」を追加する SQL 文を解答してください

自習２．データの選択

目的: 基本的なSELECTを復習する

料金が「300」よりも大きい行を選択する SQL文を解答してください

解答例

自習 1

INSERT INTO T VALUES('E', 'ラーメン', 500);

自習 2

SELECT * FROM T WHERE 料金 > 300;

9-2. データベース設計の概要

データベース設計

<u>データベース設計は、データベースの構造を定めるプロセス</u>

- テーブル名、属性、データ型、制約、索引
- テーブル間の関係性

考慮事項

- **データの冗長性の減少：正規化**
- **データの整合性の保証：制約**
  例：関連するテーブル間の参照に関する制約
  例：同じ値が２度現れないという制約
- **データの検索や操作の効率化：索引、データベースの構造の簡素化**

データベース設計の重要性

- 冗長性の減少と異状の防止
- 整合性の保証
- 性能の確保
- 拡張性（将来のデータベース拡張への対応）
- SQL が単純になるような簡素なデータベースの構造
- セキュリティ

まとめ

- **データベース設計**は、**冗長性の減少と異状の防止**、**整合性の保証**、**性能の確保**など、さまざまな面から重要

- **正規化**により**データの冗長性を排除**することは、データベース設計において重要

9-3. 正規化

正規化

目的

- **データベースの構造を最適化**
- **効率的なデータベース管理**を実現

方針

テーブルの数を減らすことよりも、**データの冗長性（重複）を減らす**ことを行う

正規化のメリット

- **データの冗長性の減少**： データの重複を減らす
- **異状の防止**：更新、削除、挿入時の異状を防ぐ
- **効率の向上**：データベース全体のサイズが縮小すると、ストレージの効率化とデータベース操作の高速化が期待できる
- **信頼性の確保**：異状の防止により、データの信頼性の向上。
- **管理の容易化**：データベースの管理が容易になる

正規化と情報無損失

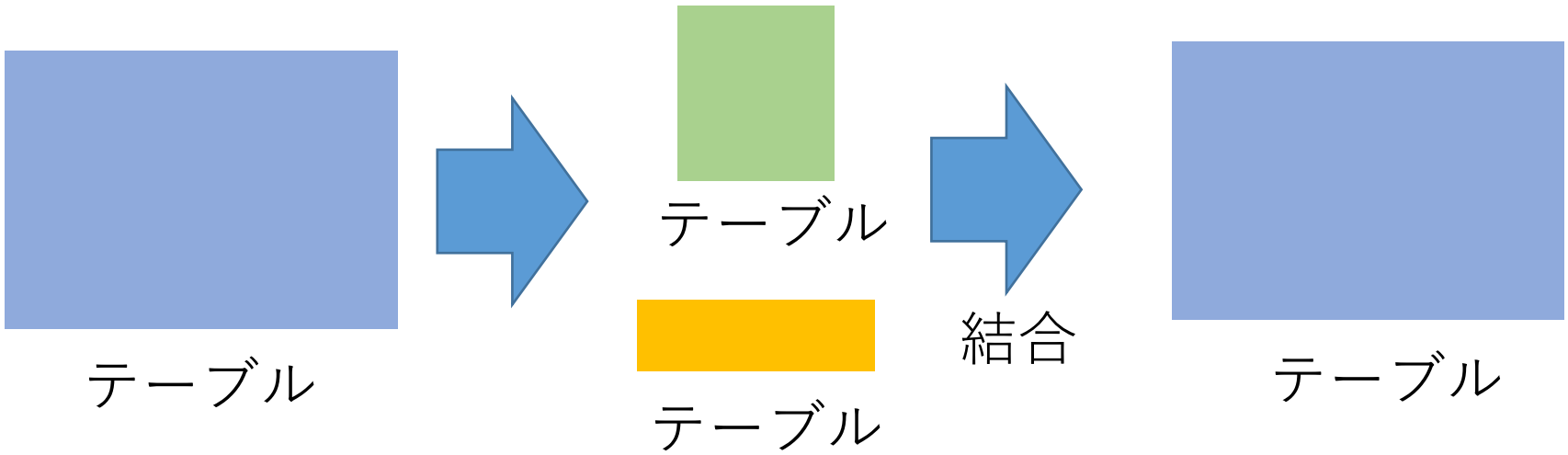
情報無損失の原則

- **正規化**においては、**元のデータベースの情報が失われたり**、**余計な情報が追加されたりしない**ことが重要

情報無損失の確認方法

- **正規化を施した後のテーブル群**から、**正規化する前のテーブルを正確に復元できるか**どうかを検証
- 正規化が、データを損なわないことを保証

正規化と情報無損失



正規化前

正規化後

正規化後の
テーブル群を使い
正規化する前の
テーブルを**復元**可能

テーブル分割のバリエーション

テーブルをどのように分割するかについては、様々考えられるが、**情報無損失で、データの冗長性が減少している**ことが重要

名前	昼食	料金
A	そば	250
B	カレーライス	400
C	カレーライス	400
D	うどん	250

分割①

名前	昼食
A	そば
B	カレーライス
C	カレーライス
D	うどん

昼食	料金
そば	250
カレーライス	400
うどん	250

分割②

名前	昼食
A	そば
B	カレーライス
C	カレーライス
D	うどん

名前	料金
A	250
B	400
C	400
D	250

分割③

昼食	料金
そば	250
カレーライス	400
うどん	250

名前	料金
A	250
B	400
C	400
D	250

分割①

名前	昼食	料金
A	そば	250
B	カレーライス	400
C	カレーライス	400
D	うどん	250

分割①

名前	昼食
A	そば
B	カレーライス
C	カレーライス
D	うどん

テーブル名: X とする

昼食	料金
そば	250
カレーライス	400
うどん	250

テーブル名: Y とする

結合のコマンド

```
SELECT X.名前, X.昼食, Y.料金
FROM X JOIN Y ON X.昼食 = Y.昼食;
```

このコマンドの実行により元に戻る

名前	昼食	料金
A	そば	250
B	カレーライス	400
C	カレーライス	400
D	うどん	250

- **情報無損失である：OK**
- **データの冗長性が減少している：OK**

分割②

名前	昼食	料金
A	そば	250
B	カレーライス	400
C	カレーライス	400
D	うどん	250

分割②

名前	昼食
A	そば
B	**カレーライス**
C	**カレーライス**
D	うどん

名前	料金
A	250
B	**400**
C	**400**
D	250

データの冗長性

更新のとき、複数個所書き換え必要な場合あり

名前	料金
A	250
B	~~400~~ **350**
C	~~400~~ **350**
D	250

- **情報無損失である：OK**
- **データの冗長性が減少<u>していない</u>：NG**

分割③

名前	昼食	料金
A	そば	250
B	カレーライス	400
C	カレーライス	400
D	うどん	250

分割③

昼食	料金
そば	250
カレーライス	400
うどん	250

名前	料金
A	250
B	400
C	400
D	250

- **情報無損失<u>でない</u>：NG**
- **データの冗長性が減少している：OK**

正規化のまとめ

- **正規化**は、**データベースの構造を最適化**、**効率的なデータベース管理**を実現
- テーブルの数を減らすことよりも、**データの冗長性（重複）を減らす**ことを行う
- メリット：**データの冗長性の減少、異状の防止、効率の向上、信頼性の確保、管理の容易化**
- **正規化では、情報無損失で、データの冗長性が減少**するように、**テーブルを分割**する

9-4. 問い合わせ結果からのテーブル生成

問い合わせ結果からのテーブル生成
CREATE TABLE … AS

CREATE TABLE … AS は、SQL **の SELECT文の結果**に基づいて**新しいテーブルを作成**

テーブル T

名前	昼食	料金
A	そば	250
B	カレーライス	400
C	カレーライス	400
D	うどん	250

```
CREATE TABLE X AS
SELECT DISTINCT 名前, 昼食
FROM T;
```

テーブル X

昼食	料金
そば	250
カレーライス	400
うどん	250

SQLFiddle での実行画面

単純な問い合わせ

```
CREATE TABLE 名簿 (
  id INTEGER,
  name TEXT );

INSERT INTO 名簿 VALUES (1, 'tokugawa');
INSERT INTO 名簿 VALUES (2, 'toyotomi');
```

```
SELECT name FROM 名簿;
```

Build Schema　Edit Fullscreen　Run SQL　Edit Fullscre

name
tokugawa
toyotomi

SQLFiddle での実行画面

CREATE TABLE … AS の使用

```
1 CREATE TABLE 名簿 (
2   id INTEGER,
3   name TEXT );
4
5 INSERT INTO 名簿 VALUES (1, 'tokugawa');
6 INSERT INTO 名簿 VALUES (2, 'toyotomi');
7 CREATE TABLE A AS SELECT name FROM 名簿;
```

```
1 SELECT * FROM A;
```

Build Schema　Edit Fullscreen　Run SQL　Edit Full

name
tokugawa
toyotomi

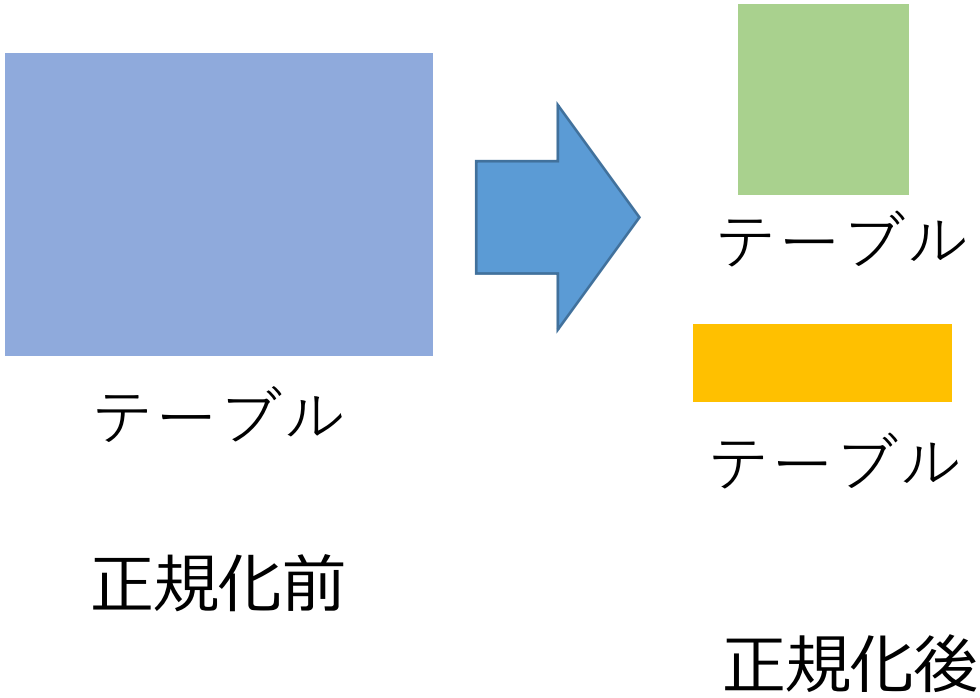
SQLFiddle では CREATE TABLE … AS は
左側のパネルに

9-5. SQL を用いた正規化

SQL を用いたデータベースの正規化

SQLを使用して、データベース内のテーブルを**正規化**

情報無損失で、**データの冗長性が減少**するように**テーブルを分割**する



SQL を用いたデータベースの正規化

名前	昼食	料金
A	そば	250
B	カレーライス	400
C	カレーライス	400
D	うどん	250

正規化前

```
CREATE TABLE X AS SELECT
DISTINCT 名前, 昼食 FROM T;
```

名前	昼食
A	そば
B	カレーライス
C	カレーライス
D	うどん

```
CREATE TABLE Y AS SELECT
DISTINCT 昼食, 料金 FROM T;
```

昼食	料金
そば	250
カレーライス	400
うどん	250

正規化後

演習２で行うこと

正規化前

名前	昼食	料金
A	そば	250
B	**カレーライス**	**400**
C	**カレーライス**	**400**
D	うどん	250

冗長なデータがある

正規化後

名前	昼食
A	そば
B	カレーライス
C	カレーライス
D	うどん

昼食	料金
そば	250
カレーライス	400
うどん	250

冗長なデータがない

正規化により、元のテーブルにあった**冗長性を排除.**

演習２．テーブルの分割による正規化

【トピックス】

1. テーブルの分割
2. 正規化
3. DISTINCT による重複除去
4. CREATE TABLE … AS によるテーブル生成

Webブラウザを使用

① アドレスバーにSQLFiddleのURLを入力

http://sqlfiddle.com/

URLが分からないときは、Googleなどの**検索エンジン**を利用。「**SQLFiddle**」と**検索**し、表示された結果からSQLFiddleのウェブサイトをクリック。

② **左側のパネル**に、**テーブル定義**と**データの追加**を行う SQL を入れる。（演習１から、**<u>末尾の２行が増えている</u>**ので注意）

```
CREATE TABLE T (
名前 TEXT,
昼食 TEXT,
料金 INTEGER);
INSERT INTO T VALUES('A', 'そば', 250);
INSERT INTO T VALUES('B', 'カレーライス', 400);
INSERT INTO T VALUES('C', 'カレーライス', 400);
INSERT INTO T VALUES('D', 'うどん', 250);
CREATE TABLE X AS SELECT DISTINCT 名前, 昼食 FROM T;
CREATE TABLE Y AS SELECT DISTINCT 昼食, 料金 FROM T;
```

③ 「Build Schema」をクリック

SQL Fiddle　MySQL 5.6　View Sample Fiddle　Clear　Text to DDL

```
CREATE TABLE T (
名前 TEXT,
昼食 TEXT,
料金 INTEGER);
INSERT INTO T VALUES('A', 'そば', 250);
INSERT INTO T VALUES('B', 'カレーライス', 400);
INSERT INTO T VALUES('C', 'カレーライス', 400);
INSERT INTO T VALUES('D', 'うどん', 250);
CREATE TABLE X AS SELECT DISTINCT 名前, 昼食 FROM T;
CREATE TABLE Y AS SELECT DISTINCT 昼食, 料金 FROM T;
```

Build Schema　Edit Fullscreen　Browser　[;]

Please build schema.

Run SQL　Edit Fullscreen　[;]

Please build schema.

④ **右側のパネル**に、**問い合わせ（クエリ）**を行う SQL を入れる。

```
SELECT * FROM X;
SELECT * FROM Y;
```

⑤ 「**Run SQL**」をクリック

SQL 文が**実行**され、結果が表示される。

⑥ 下側のウインドウで、**結果を確認。**

```
1 CREATE TABLE T (
2 名前 TEXT,
3 昼食 TEXT,
4 料金 INTEGER);
5 INSERT INTO T VALUES('A', 'そば', 250);
6 INSERT INTO T VALUES('B', 'カレーライス', 400);
7 INSERT INTO T VALUES('C', 'カレーライス', 400);
8 INSERT INTO T VALUES('D', 'うどん', 250);
9 CREATE TABLE X AS SELECT DISTINCT 名前, 昼食 FROM T;
10 CREATE TABLE Y AS SELECT DISTINCT 昼食, 料金 FROM T;
```

```
1 SELECT * FROM X;
2 SELECT * FROM Y;
```

名前	昼食
A	そば
B	カレーライス
C	カレーライス
D	うどん

Record Count: 4; Execution Time: 31ms View Execution Plan link

昼食	料金
そば	250
カレーライス	400
うどん	250

Record Count: 3; Execution Time: 2ms View Execution Plan link

演習３．正規化における情報無損失

【トピックス】

1. **正規化**
2. **結合**

Webブラウザを使用

① アドレスバーにSQLFiddleのURLを入力

http://sqlfiddle.com/

URLが分からないときは、Googleなどの**検索エンジン**を利用。「**SQLFiddle**」と**検索**し、表示された結果からSQLFiddleのウェブサイトをクリック。

② **左側のパネル**に、**テーブル定義**と**データの追加**を行う SQL を入れる。（演習 2 と同じ）

```
CREATE TABLE T (
名前 TEXT,
昼食 TEXT,
料金 INTEGER);
INSERT INTO T VALUES('A', 'そば', 250);
INSERT INTO T VALUES('B', 'カレーライス', 400);
INSERT INTO T VALUES('C', 'カレーライス', 400);
INSERT INTO T VALUES('D', 'うどん', 250);
CREATE TABLE X AS SELECT DISTINCT 名前, 昼食 FROM T;
CREATE TABLE Y AS SELECT DISTINCT 昼食, 料金 FROM T;
```

③ 「Build Schema」をクリック

SQL Fiddle　MySQL 5.6　View Sample Fiddle　Clear　Text to DDL

```
CREATE TABLE T (
名前 TEXT,
昼食 TEXT,
料金 INTEGER);
INSERT INTO T VALUES('A', 'そば', 250);
INSERT INTO T VALUES('B', 'カレーライス', 400);
INSERT INTO T VALUES('C', 'カレーライス', 400);
INSERT INTO T VALUES('D', 'うどん', 250);
CREATE TABLE X AS SELECT DISTINCT 名前, 昼食 FROM T;
CREATE TABLE Y AS SELECT DISTINCT 昼食, 料金 FROM T;
```

Build Schema　Edit Fullscreen　Browser　[;]

Please build schema.

Run SQL　Edit Fullscreen　[;]

Please build schema.

④ **右側のパネル**に、**問い合わせ（クエリ）**を行う SQL を入れる。

```
SELECT X.名前, X.昼食, Y.料金 FROM X JOIN Y ON X.昼食 = Y.昼食;
```

⑤ 「**Run SQL**」をクリック

SQL 文が**実行**され、結果が表示される。

⑥ 下側のウインドウで、**結果を確認。**

```
CREATE TABLE T (
名前 TEXT,
昼食 TEXT,
料金 INTEGER);
INSERT INTO T VALUES('A', 'そば', 250);
INSERT INTO T VALUES('B', 'カレーライス', 400);
INSERT INTO T VALUES('C', 'カレーライス', 400);
INSERT INTO T VALUES('D', 'うどん', 250);
CREATE TABLE X AS SELECT DISTINCT 名前, 昼食 FROM T;
CREATE TABLE Y AS SELECT DISTINCT 昼食, 料金 FROM T;
```

Build Schema | Edit Fullscreen | Browser | [;]

```
SELECT X.名前, X.昼食, Y.料金 FROM X JOIN Y ON X.昼食 = Y.昼食;
```

Run SQL | Edit Fullscreen | [;]

名前	昼食	料金
A	そば	250
B	カレーライス	400
C	カレーライス	400
D	うどん	250

自習３．学生情報のためのテーブル作成

目的: 学生とその受講情報を格納するテーブルを作成

次の SQL を実行することにより、S テーブルを作成。「SELECT * FROM S;」により確認。

```
CREATE TABLE S (
  StudentID INTEGER,
  StudentName TEXT,
  Course TEXT
);

INSERT INTO S VALUES (1, 'Alice', 'Math');
INSERT INTO S VALUES (1, 'Alice', 'Science');
INSERT INTO S VALUES (2, 'Bob', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Math');
INSERT INTO S VALUES (3, 'Charlie', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Science');

SELECT * FROM S;
```

自習４．冗長なデータの確認

目的: 学生について ID と名前のデータが冗長になっていることを確認

自習３に続いて、次の SQL の実行結果を確認

```
SELECT StudentID, StudentName FROM S;
```

自習５．重複なしの学生情報抽出

目的: DISTINCT キーワードを用いて重複を除去する。

自習４に続いて、次の SQL の実行結果を確認

```
SELECT DISTINCT StudentID, StudentName FROM S;
```

自習６．正規化

目的: DISTINCT キーワードを用いて重複を除去する。

自習４に続いて、次の SQL の実行結果を確認

```
CREATE TABLE U AS SELECT DISTINCT StudentID,
StudentName FROM S;

CREATE TABLE V AS SELECT DISTINCT StudentID,
Course FROM S;


SELECT * FROM U;

SELECT * FROM V;
```

自習 3

```sql
CREATE TABLE S (
  StudentID INTEGER,
  StudentName TEXT,
  Course TEXT
);

INSERT INTO S VALUES (1, 'Alice', 'Math');
INSERT INTO S VALUES (1, 'Alice', 'Science');
INSERT INTO S VALUES (2, 'Bob', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Math');
INSERT INTO S VALUES (3, 'Charlie', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Science');
```

Build Schema | Edit Fullscreen | Browser | [;]

```sql
SELECT * FROM S;
```

Run SQL | Edit Fullscreen | [;]

StudentID	StudentName	Course
1	Alice	Math
1	Alice	Science
2	Bob	History
3	Charlie	Math
3	Charlie	History
3	Charlie	Science

自習 4

```sql
CREATE TABLE S (
  StudentID INTEGER,
  StudentName TEXT,
  Course TEXT
);

INSERT INTO S VALUES (1, 'Alice', 'Math');
INSERT INTO S VALUES (1, 'Alice', 'Science');
INSERT INTO S VALUES (2, 'Bob', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Math');
INSERT INTO S VALUES (3, 'Charlie', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Science');
```

Build Schema | Edit Fullscreen | Browser | [;]

```sql
SELECT StudentID, StudentName FROM S;
```

Run SQL | Edit Fullscreen | [;]

StudentID	StudentName
1	Alice
1	Alice
2	Bob
3	Charlie
3	Charlie
3	Charlie

自習 5

```sql
CREATE TABLE S (
  StudentID INTEGER,
  StudentName TEXT,
  Course TEXT
);

INSERT INTO S VALUES (1, 'Alice', 'Math');
INSERT INTO S VALUES (1, 'Alice', 'Science');
INSERT INTO S VALUES (2, 'Bob', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Math');
INSERT INTO S VALUES (3, 'Charlie', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Science');
```

Build Schema | Edit Fullscreen | Browser | [;]

```sql
SELECT DISTINCT StudentID, StudentName FROM S;
```

Run SQL | Edit Fullscreen | [;]

StudentID	StudentName
1	Alice
2	Bob
3	Charlie

自習 6

```sql
CREATE TABLE S (
  StudentID INTEGER,
  StudentName TEXT,
  Course TEXT
);

INSERT INTO S VALUES (1, 'Alice', 'Math');
INSERT INTO S VALUES (1, 'Alice', 'Science');
INSERT INTO S VALUES (2, 'Bob', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Math');
INSERT INTO S VALUES (3, 'Charlie', 'History');
INSERT INTO S VALUES (3, 'Charlie', 'Science');
CREATE TABLE U AS SELECT DISTINCT StudentID, StudentName FROM S;
CREATE TABLE V AS SELECT DISTINCT StudentID, Course FROM S;
```

Build Schema | Edit Fullscreen | Browser | [;]

```sql
SELECT * FROM U;
SELECT * FROM V;
```

Executing SQL... | Edit Fullscreen | [;]

StudentID	StudentName
1	Alice
2	Bob
3	Charlie

Record Count: 3; Execution Time: 16ms View Execution Plan link

StudentID	Course
1	Math
1	Science
2	History
3	Math
3	History
3	Science

Record Count: 6; Execution Time: 9ms View Execution Plan link

9-6. 正規形のバリエーション

正規化と正規形

- **正規形**は、**データベースの正規化**における**さまざまなレベル**
- より高度な正規化レベルへ進むにつれて、データの冗長性をより減少させることを目指す

- 第一正規形
- 第二正規形
- 第三正規形
- ボイスコッド正規形（*3.5* 正規形ともいう）
- 第四正規形
- 第五正規形

ただし、「レベルが高いほど良い」と決まってはいない。レベルが高いほど、データ更新時の性能低下の可能性、制約の記述不可能の可能性

第三正規形への変換例①

第一正規形や第二正規形のテーブルを、第三正規形に変換

名前	昼食	料金
A	そば	250
B	**カレーライス**	400
C	**カレーライス**	400
D	うどん	250

冗長なデータが原因で、第三正規形ではない

名前	昼食
A	そば
B	カレーライス
C	カレーライス
D	うどん

昼食	料金
そば	250
カレーライス	400
うどん	250

両方とも、第三正規形である

第三正規形への変換例②

第一正規形や第二正規形のテーブルを、第三正規形に変換

会員番号	住所	注文した商品
100	福山市野上町4-3-2	りんご
101	福山市曙町1-2-3-4	りんご
100	福山市野上町4-3-2	ばなな

冗長なデータが原因で、第三正規形ではない

会員番号	住所
100	福山市野上町4-3-2
101	福山市曙町1-2-3-4

会員番号	注文した商品
100	りんご
101	りんご
100	ばなな

両方とも、第三正規形である

正規形のレベル

- **第一正規形**

テーブルのセルには，１つの値を入れる．セルの合併はしない.

- **第二正規形**

候補キーに含まれない属性は、すべて候補キーに従属する．そして，候補キーの部分集合には従属しない.

- **第三正規形**

主キー以外の属性は，すべて主キーにのみ直接，従属する

- **ボイスコッド正規形（*3.5* 正規形ともいう）**

すべての従属関係 $X \rightarrow Y$ について、それは自明であるか、Xが超キーである.

- **第四正規形**

すべての多値従属関係 $X \twoheadrightarrow Y$ について、それは自明であるか、Xが候補キーであるか、Xがその超集合である.

- **第五正規形**

すべての結合従属性について、それは自明であるか、候補キーにより含意される

全体まとめ①

データベース設計

- データベース設計は、データベースの構造を定めるプロセス
- テーブル名、属性、データ型、制約、索引、テーブル間の関係性などを定める

データベース設計での考慮事項

- データの冗長性の減少：正規化
- データの整合性の保証：制約
- データの検索や操作の効率化：索引、データベースの構造の簡素化

データベース設計の重要性

- 冗長性の減少と異状の防止
- 整合性と性能の確保
- 拡張性とセキュリティの確保

全体まとめ②

正規化の目的

データベースの構造を**最適化**。効率的なデータベース管理を実現

正規化のメリット

- データの冗長性の減少
- 異状の防止
- 効率の向上
- 信頼性の確保
- 管理の容易化

情報無損失の原則

- 正規化においては、**元のデータベースの情報が失われたり**、**余計な情報が追加されたりしない**ことが重要

SQLを用いた正規化

- **情報無損失**で、**データの冗長性が減少**するように**テーブルを分割**
- CREATE TABLE ... AS コマンドを用いて新しいテーブルを作成

① 理論と実践の両立

正規化の目的は、データベースの構造を最適化し、効果的な管理を実現することです。SQLを用いた正規化では、「CREATE TABLE ... AS」のようなSQLコマンドを使用して、正規化を実行できます。

② データベース設計の重要性の理解

データベース設計は、テーブル名、属性、データ型、制約、索引を定めるプロセスであり、データベース全体の構造を決定します。正規化を行うことで、データの冗長性を減少させ、良いデータベース設計を達成できます。データベース設計は、冗長性の減少、異常の防止、整合性と性能の確保など、さまざまな観点から重要です。

③ データベース運用能力の向上

正規化により、効率的なデータベース設計が可能になり、データベース運用がより簡単になり、性能の向上も期待できます。正規化においては、情報無損失原則に基づくテーブル分割を行います。正規化を理解することは、データベース運用能力の向上に寄与します。