de-14. データベース設計演習，正規化

（データベース演習）

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

金子邦彦

① SQLスキルの向上

② データベース運用スキル

③ 問題解決能力と論理的思考力

14-1. 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

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

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

正規化によるデータの不整合の防止

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

データの不整合はない

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

正規化と情報無損失

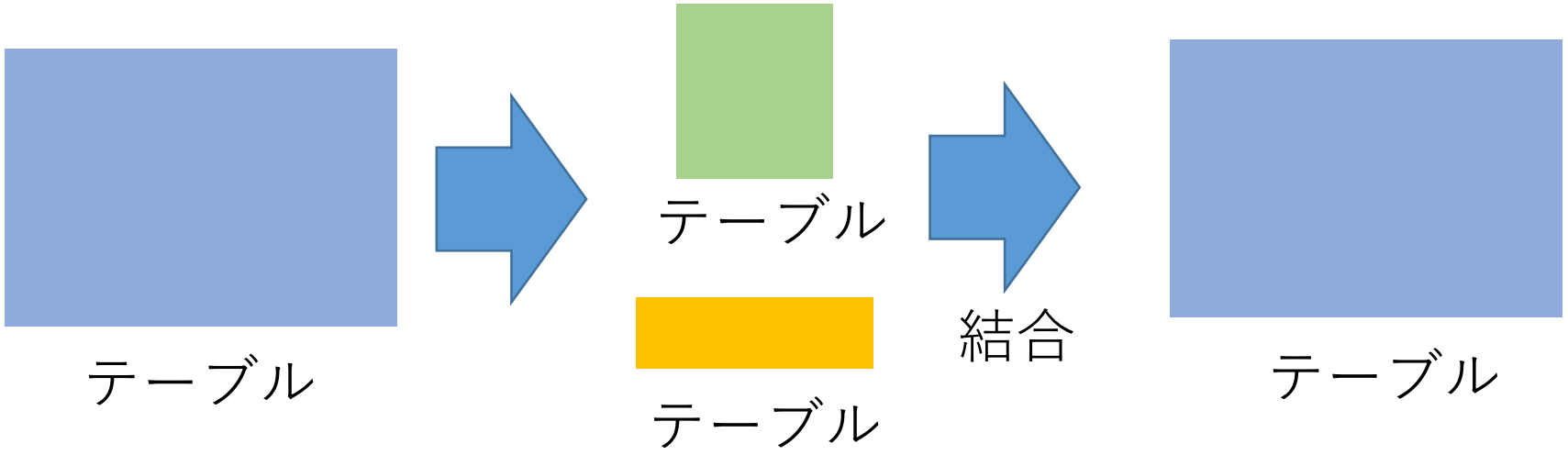
情報無損失の原則

- **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

正規化と情報無損失

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

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 INNER 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**

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
INNER JOIN 購入
ON 商品.ID = 購入.商品番号;
```

SQL による結合の基本

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
INNER 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

結合条件に基づいて，両テーブルのデータが結合される．

テーブル結合の総括

- **結合**は，**異なるテーブルを一つにまとめる**操作である.
- **結合条件**は通常，**テーブルの特定の属性同士の値が等しい**という条件を指定する.
- より複雑な結合条件なども指定できる.

結合条件

結合のためのSQL

```
SELECT * FROM 商品
INNER 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
```

結合結果の絞り込みと Access 固有の SQL 制約

商品テーブルと購入テーブルを結合．特定の商品「X」を購入したものに絞り込み

SQL の世界標準： **INNER JOIN … ON のあとで AND, OR が使える．**

```
SELECT * FROM 商品
INNER JOIN 購入
ON 商品.ID = 購入.商品番号 AND 購入.購入者 = 'X';
```

Access： **ON のあとで AND, OR が使えない．AND の代替で WHERE を使う**

```
SELECT * FROM 商品
INNER JOIN 購入
ON 商品.ID = 購入.商品番号 WHERE 購入.購入者 = 'X';
```

正規化のメリットとデメリット

正規化のメリット

- データの**冗長性を排除**
- データの**整合性が向上**
- 更新、削除、挿入時の**異状を減少**

正規化のデメリット

- **過度**に正規化されたデータベースでは、テーブルの数が多くなり、利用が複雑になる場合がある。性能上の問題が発生する可能性もある。

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．

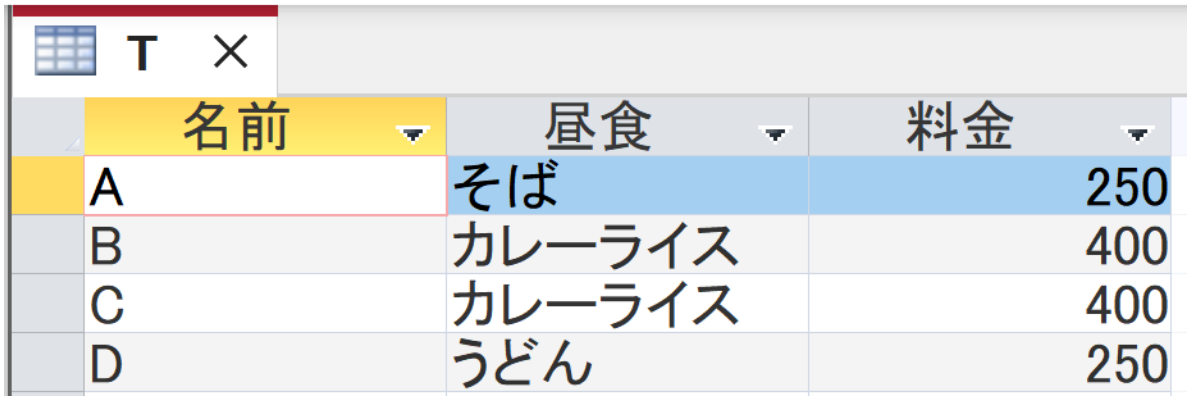
- **第五正規形**

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

8-2. 演習

いまから演習で行うこと、注意点

- 次のテーブルを作成



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

【Access での注意点】

- **SQLビューでは、SQL文を１つずつ実行**

  （複数まとめての一括実行ができない）

- **CREATE TABLE** では、「実行」の後、**画面が変化しない**が実行できている
- **INSERT INTO** では、「実行」の後、**確認表示**が出る。その後、**画面が変化しない**が実行できている

演習１．Access の SQL ビューを用いたテーブル定義とデータの追加

【トピックス】

- **SQLビューを開く**
- **SQL文の編集**
- **create table**
- **insert into**
- **SQL文の実行**

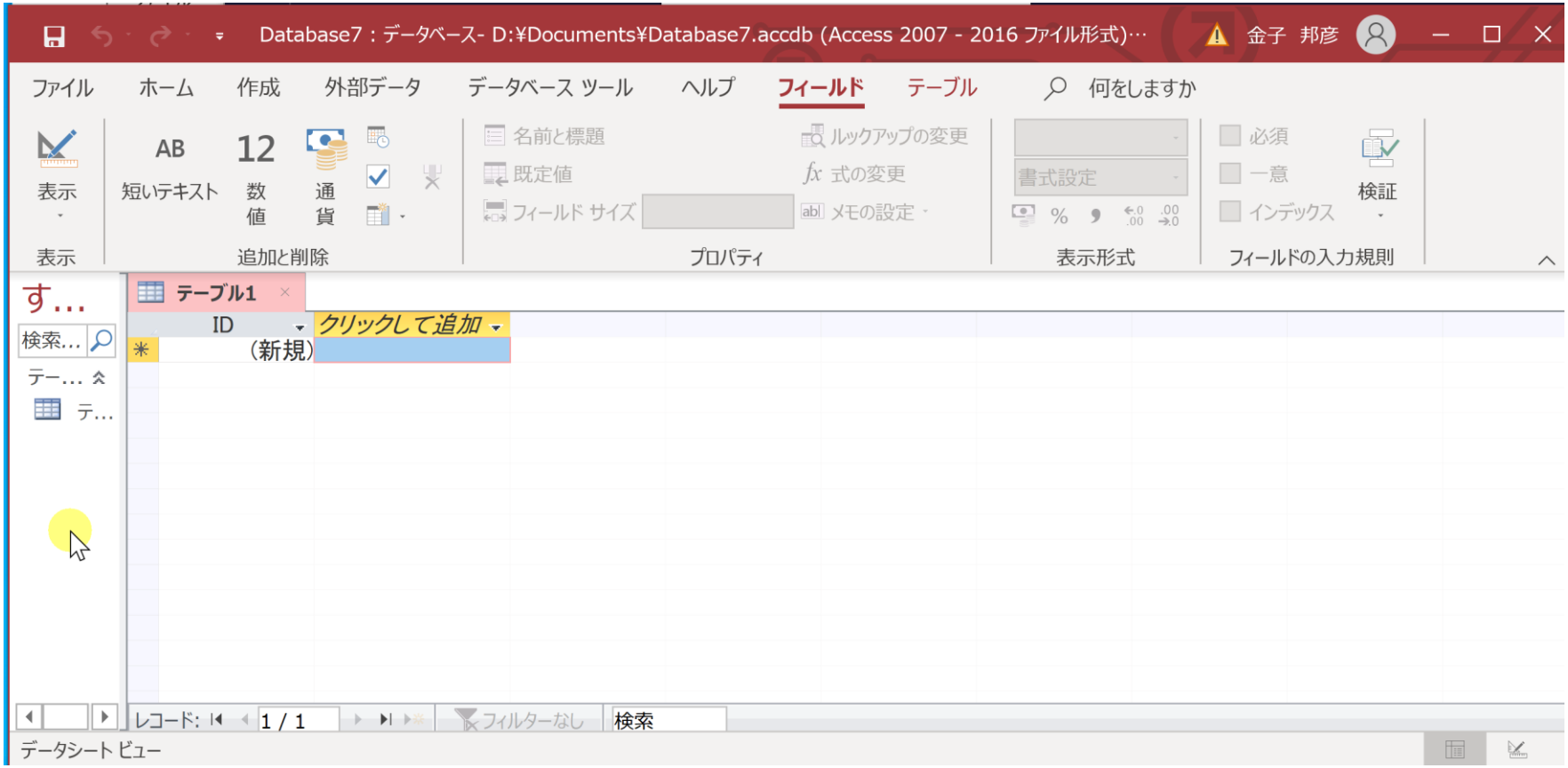
演習

1. パソコンを使用する
   前もって Access をインストールしておくこと

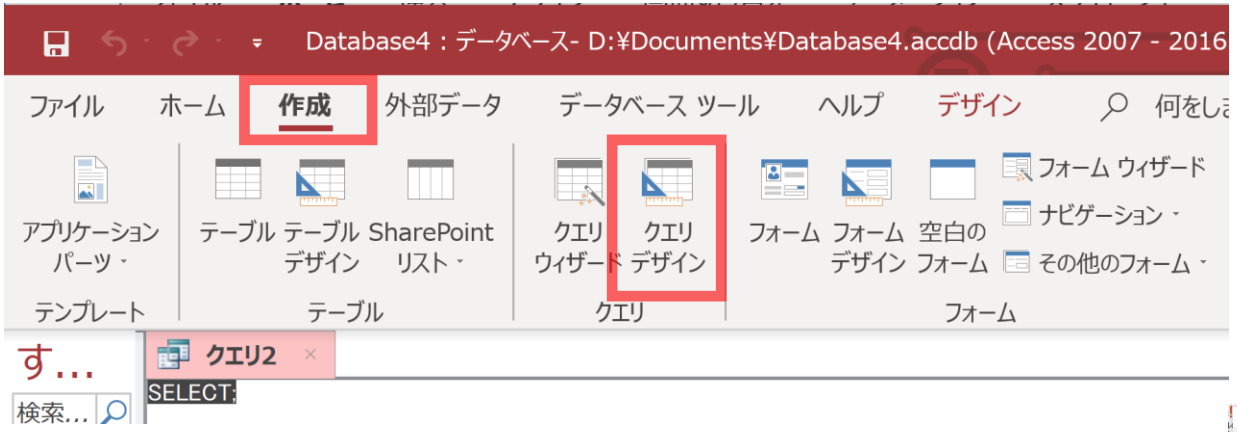
2. Access を起動する

3. Access で、「**空のデータベース**」を選び、「**作成**」をクリック.

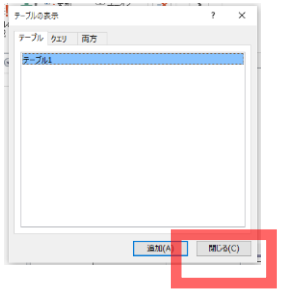
4. **テーブルツール画面**が表示されることを確認



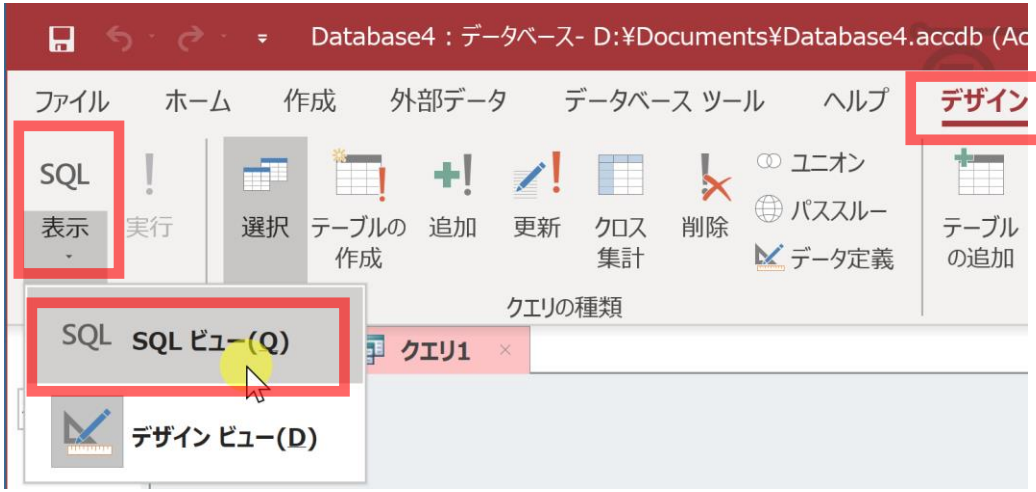
5. 次の手順で、**SQLビュー**を開く.



①「**作成**」タブで、「**クエリデザイン**」をクリック



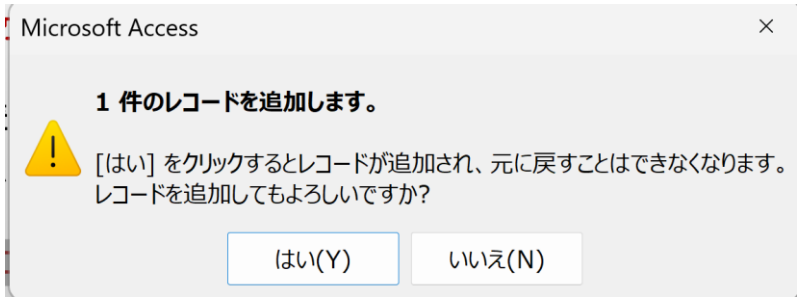
このような表示が出たときは「**閉じる**」をクリック



②「**デザイン**」タブで、「**表示**」を展開し「**SQL ビュー**」を選ぶ

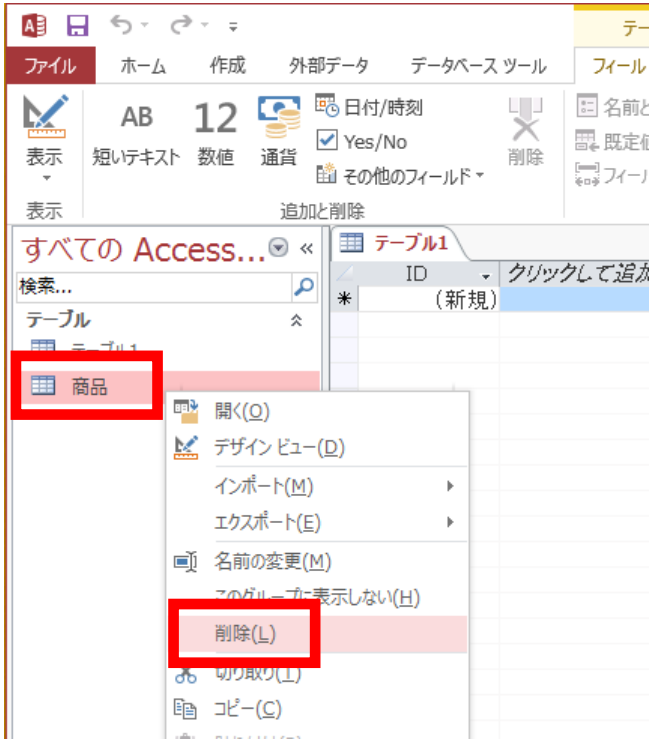
6. **SQL ビュー**に、次の SQL を１つずつ入れ、「**実行**」ボタンで、**SQL文**を実行．結果を確認

```
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);
```



INSERT INTOでは、「実行」の後、確認表示が出る。その後、**画面が変化しない**が実行できている

間違ってしまったときは、テーブルの削除を行ってからやり直した方が早い場合がある



テーブルビューで、削除したいテーブルを**右クリック**して、「**削除**」

テーブルを削除するときは、
間違って必要な**テーブル**を削除しないように、十分に注意する！
（元に戻せない）

演習２．種々のSQL問い合わせ. Access の SQL ビューを使用.

【トピックス】

1. **単純な表示**
2. **特定の属性のみ表示（射影）**
3. **重複行の除去 DISTINCT**

Access の SQL ビューを用いた問い合わせ

① Access の **SQLビュー**開く

② **SQL 文**の**編集。select, from, where** を使用

例: select * from テーブル名 where 列1 = 値1;

③ **SQL 文**の**実行**

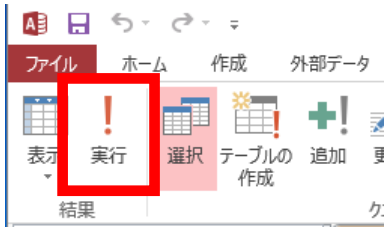
実行の結果、**データシートビュー**に画面が変わり、そこに**問い合わせの結果**が表示される

④ さらにSQL 文の編集、実行を続ける場合には、**画面を SQL ビューに切り替える**

SQL 問い合わせ（クエリ）で使用する２つのビュー

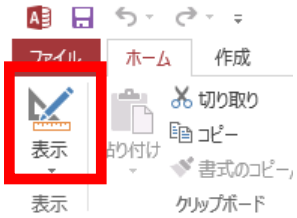
```
SELECT * from 商品;
```

SQL ビュー
SQL 文の**作成**、**編集**



実行

ID	名前	単価
1	みかん	50
2	りんご	100
3	りんご	150
＊	（新規）	0

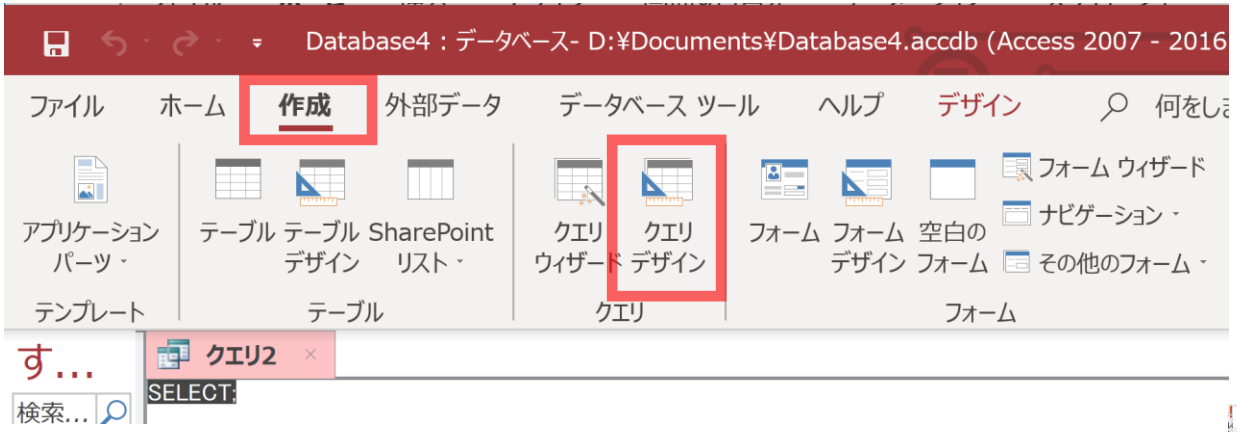


表示＋SQL ビュー

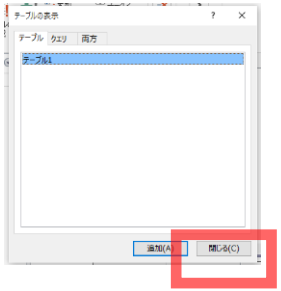
データシートビュー
問い合わせ（クエリ）の結果

マウス操作でビューを切り替え

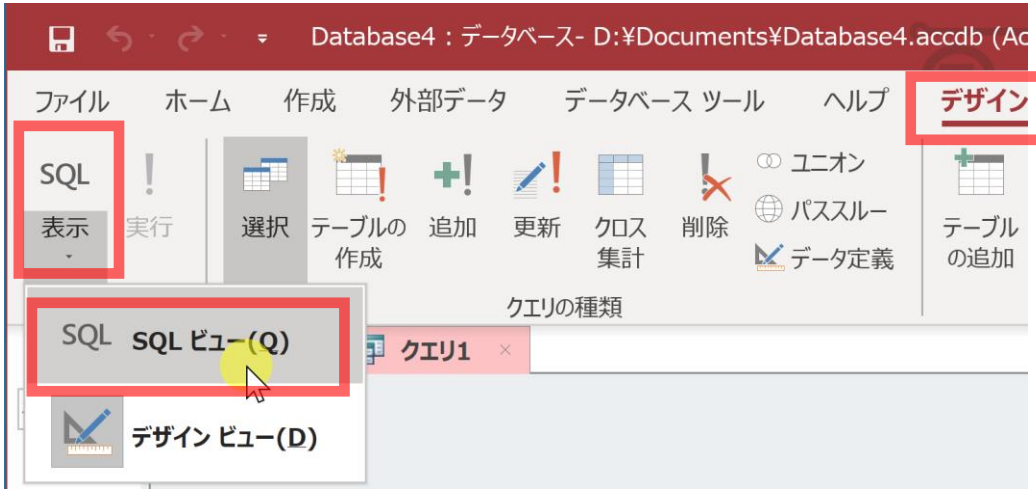
1. 次の手順で、**SQLビュー**を開く.



①「**作成**」タブで、「**クエリデザイン**」をクリック



このような表示が出たときは「**閉じる**」をクリック



②「**デザイン**」タブで、「**表示**」を展開し「**SQL ビュー**」を選ぶ

2. **SQL ビュー**に、次の SQL を１つずつ入れ、「**実行**」ボタンで、**SQL文**を実行．結果を確認

1. 単純な表示

SELECT * FROM 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
✳			

2. 昼食の列のみ

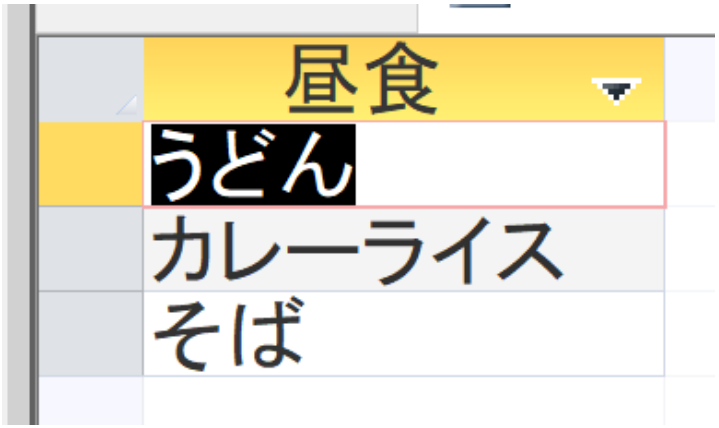
SELECT 昼食 FROM T;

	昼食
	そば
	カレーライス
	カレーライス
	うどん
✳	

(続き)

3.重複行の除去 DISTINCT

SELECT DISTINCT 昼食 FROM T;



演習３で行うこと

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

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

X と Y の結合により元に戻る

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

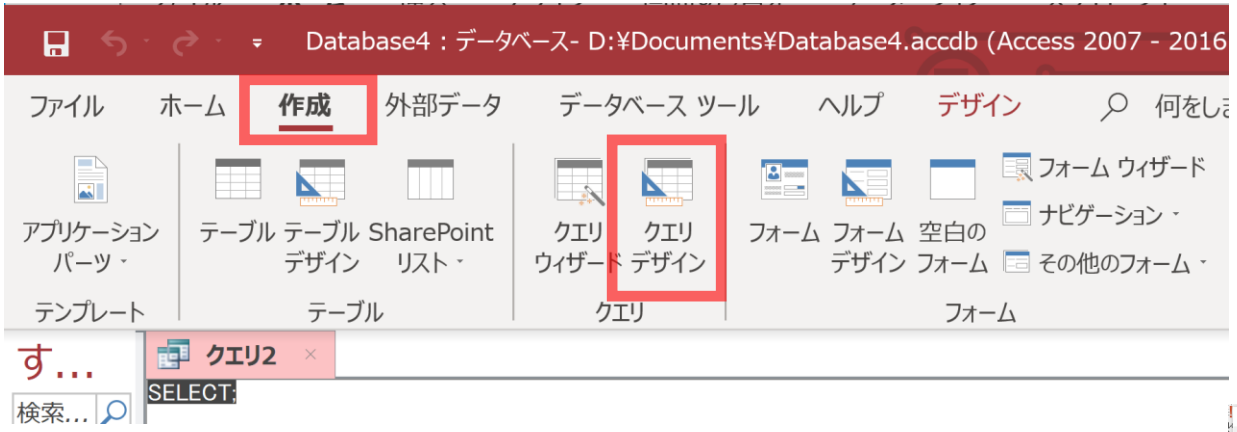
- **情報無損失である：OK**
- **データの冗長性が減少している：OK**

演習３．正規化、正規化における情報無損失

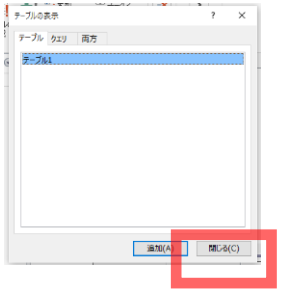
【トピックス】

1. **問い合わせ結果によるテーブル生成 INTO**
2. **結合**
3. **正規化**
4. **正規化における情報無損失**

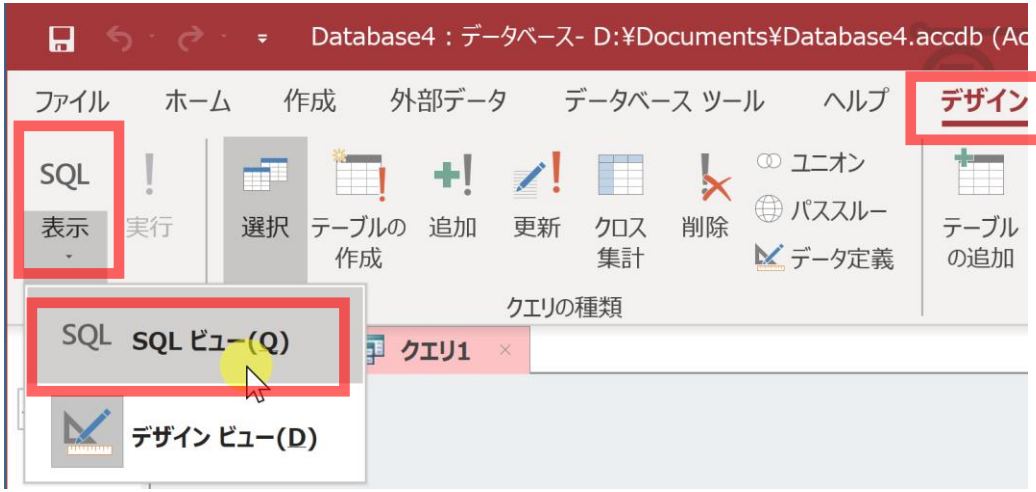
1. 次の手順で、**SQLビュー**を開く.



①「**作成**」タブで、「**クエリデザイン**」をクリック



このような表示が出たときは「**閉じる**」をクリック

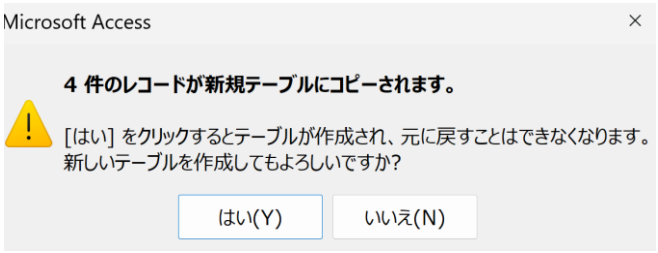


②「**デザイン**」タブで、「**表示**」を展開し「**SQL ビュー**」を選ぶ

2. **SQL ビュー**に、次の SQL を１つずつ入れ、「**実行**」ボタンで、**SQL文**を実行．結果を確認

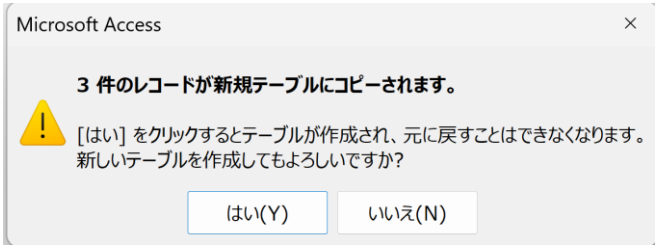
1. テーブル X の生成

SELECT DISTINCT 名前, 昼食 INTO X FROM T;



2. テーブル Y の生成

SELECT DISTINCT 昼食, 料金 INTO Y FROM T;



(続き)

3. テーブル X の確認

SELECT * FROM X;

	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
✳		

4. テーブル Y の確認

SELECT * FROM Y;

	昼食	料金
	うどん	250
	カレーライス	400
	そば	250
✳		

(続き)

5.テーブルの結合により元に戻ることを確認

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

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

リレーショナルデータベースのテーブルでは、行の順序は気にしないことになっている

発展演習１．SQLを用いた正規化

目的：次のテーブル S を SQL を用いて正規化する

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

Access で次ページの SQL を１つずつ実行し、結果を確認

```
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 DISTINCT StudentID, StudentName INTO U FROM S;

SELECT DISTINCT StudentID, Course INTO V FROM S;

SELECT * FROM S;

SELECT * FROM U;

SELECT * FROM V;

SELECT U.StudentID, U.StudentName, V.Course
FROM U INNER JOIN V ON U.StudentID = V.StudentID;
```

SELECT * FROM S; の結果

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

SELECT * FROM U; の結果

StudentID	StudentNam
1	Alice
2	Bob
3	Charlie

SELECT * FROM V; の結果

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

SELECT U.StudentID, U.StudentName, V.Course
FROM U INNER JOIN V ON U.StudentID = V.StudentID;

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