データベースシステム全体まとめ

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

金子邦彦

CC BY NC SA

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

15-1. データベースシステム序論

データベースとは

データベースは、**特定のテーマや目的**に従って収集された**大量のデータ**

例：銀行、商店、交通機関、電話会社などさまざま

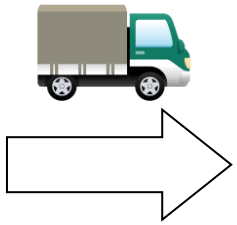
取引

記入

計測　撮影

データ保存

データ収集



データベース
（データの集まり）

データベースの必要性

日常生活での情報管理に不可欠

- 銀行の銀行口座
- ホテルの予約情報
- 交通機関の座席予約情報
- 大学の履修登録や出欠の情報
- 企業の製品情報
- 電話会社の通話量情報

データベースがなければ、
現代の生活が成り立たない

サイバーフィジカル

物理的な現実世界（フィジカル）と、デジタルな情報世界（サイバー）が融合したシステム

農林畜産業、
医療、
ヘルスケア、
製造業、
都市交通、
電力　など

サービス
提供

センサー
データ

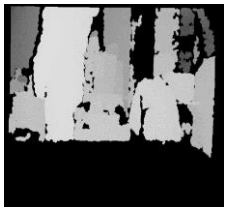
育成法の予測
遠隔医療
故障予測
交通渋滞の予測
需要予測

実世界

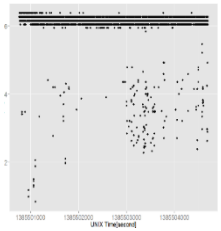
サイバー世界

センサー

センサーの設置



センサーで計
測された距離
画像



人間の通過記録

効率化、品質の向上、新サービスの創生を可能に

情報セキュリティ

データベースは情報セキュリティにも関連

- **データの保護**

データベースは重要な情報を格納。

コンピュータシステムの故障、電力不足、災害からデータを守る必要がある。

- **セキュリティ上の脅威**

不正侵入や情報漏洩から会員情報やデータベースのデータを保護する必要がある。セキュリティ上の脅威に対処するための対策が重要。

ここまでのまとめ

- **データベース**は、**特定のテーマや目的**に従って収集された**大量のデータ**

- **データベース**は**情報セキュリティ**、**サイバーフィジカル**、**クラウドコンピューティング**などの**新たなテクノロジー**と結びついて、**現代社会に多大な影響**を与えている。

情報とデータ

- **データ**は**数字や符号の集まり**

- **情報**は**データに意味を持たせたもの**

データベースシステムとは

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）

【主要目的】

大量のデータを**安全かつ効率的に保存、管理、検索、共有すること**で、迅速な業務実行と正確な意思決定が可能になる。

データベースシステムの役割

- **データの構造化**

データを整理。データは分かりやすくなり、**関連性のあるデータ同士を結び付ける**ことも**簡単**に。

- **データの整合性**

データの整合性を保つための仕組みを提供。**データが正確で矛盾しないように、制約を設定できる**。データの品質が向上し、誤った情報が排除される。

- **データの永続性**

データを永続的に保存する。**データは電源を切っても消えず、長期間にわたって安全に保管される**。データを失う心配がなくなる。

15-2. リレーショナルデータベース

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）（データベース操作言語）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する．

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と他のデータベースの違い

リレーショナルデータベース

- データを**テーブル**と呼ばれる**表形式で保存**
  （エクセルのワークシートに似ている）
- **SQL** 言語を使って、**データの管理と検索**が行える
- **テーブル形式でデータを整理**することで、**データの構造化、整合性、永続性が保証される。**

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

他のデータベース（例：NoSQL）

- **データ形式がより柔軟**
- **データの構造化と整合性に関する能力はリレーショナルデータベースより制限される**

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

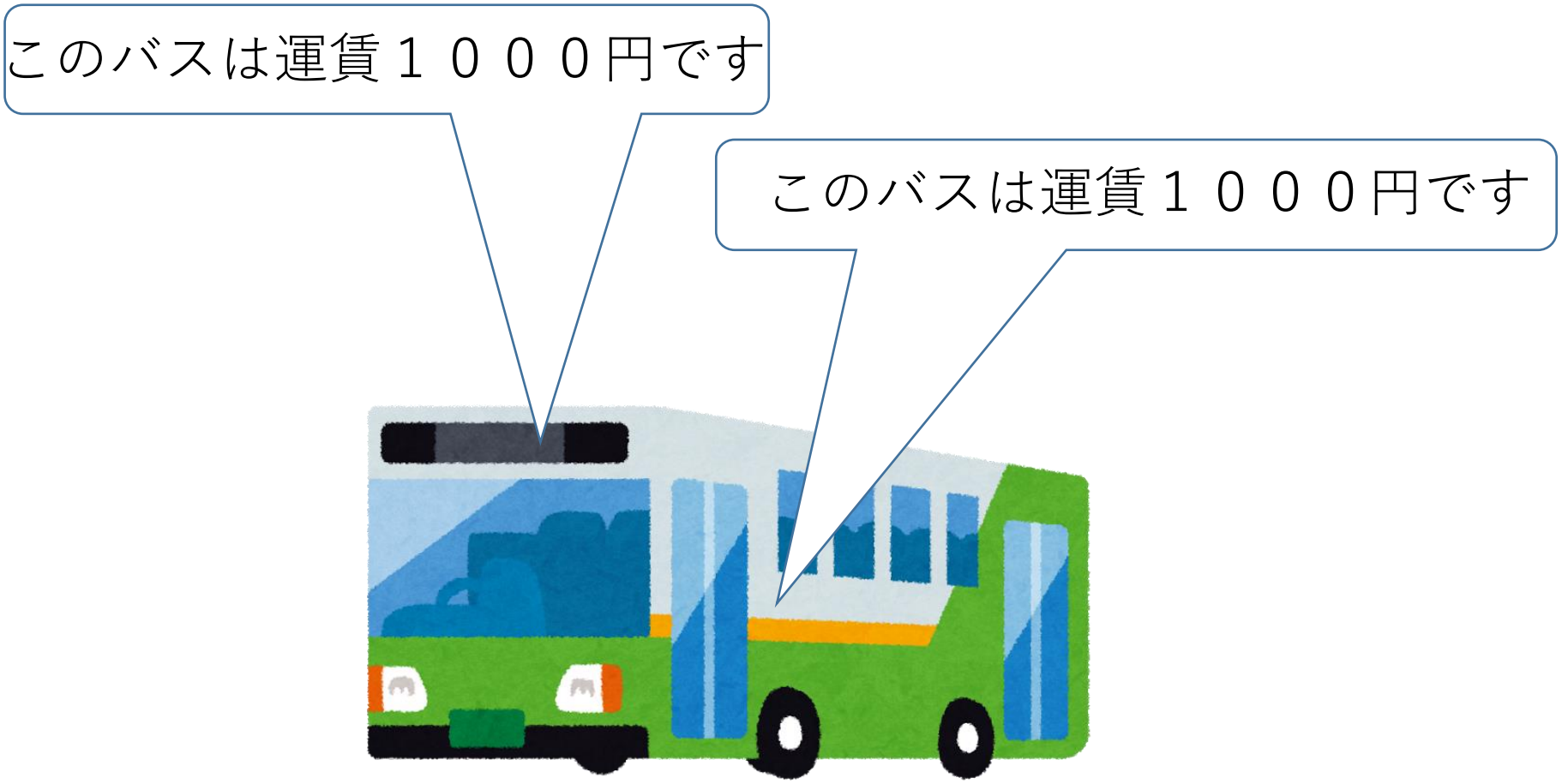
購入者	商品番号
X	1
X	3
Y	2

リレーショナルデータベースが普及している理由

- 使いやすい
- 信頼性が高い
- データを**テーブル**と呼ばれる**表形式で保存**（エクセルに似ていてなじみやすい）
- **データの管理と検索**が、**SQL** 言語を使って、簡単に行える

15-3. データベース設計と正規化

冗長なデータ



冗長なデータ　＝　データベースとしてはNG

「更新の際、すべての個所を更新しないといけない」
などの問題がある

冗長なデータ

冗長なデータは、データベース内で**不必要に重複して保存されるデータ**を指す。

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

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

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

正規化による問題解消

更新

カレーライスが
400円から 350円に値下げ

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~~
うどん	250

350

「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。

正規化のまとめ

- **正規化**：リレーショナルデータベースのテーブル再構成
- **目的**：データの冗長性排除、整合性向上
- **結果**：データの不整合防止
- **位置付け**：データベース設計の必須ステップ

15-4. SQL の様々な機能

SQL 理解のための前提知識

○ テーブル

データを**テーブル**と呼ばれる**表形式で保存**

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

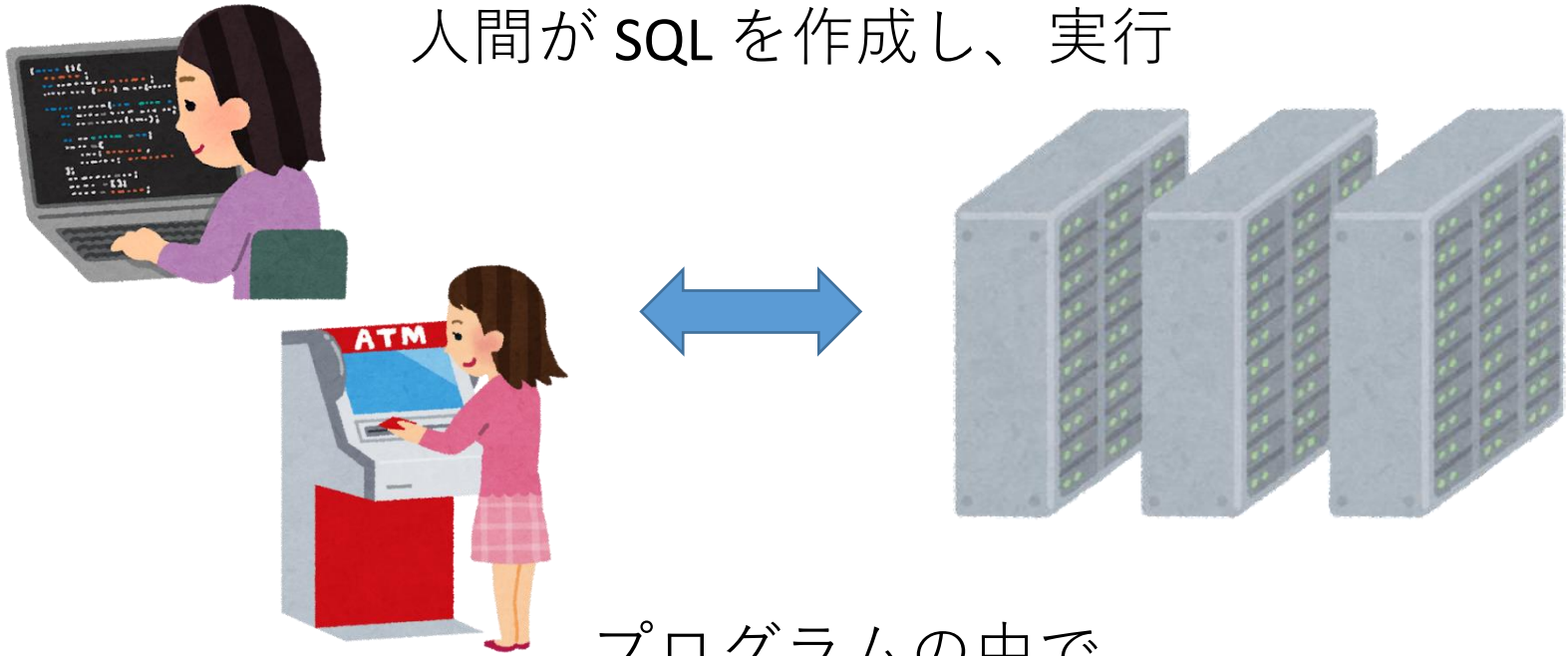
購入者	商品番号
X	1
X	3
Y	2

○ 問い合わせ（クエリ）

- **問い合わせ（クエリ）**は、**データベース**から**必要なデータを検索、加工するための指令**

SQL

SQL は、**リレーショナルデータベースシステム**で、**問い合わせ（クエリ）**や、その他、さまざまな機能を実行できる



問い合わせ（クエリ）の仕組み

1. **発行**：「問い合わせ」をデータベースシステムに送信
2. **解析・取得**：「問い合わせ」が解析され，必要なデータがデータベースから読み込まれる
3. **加工**：計算，集計・集約，並べ替え（ソート），結合などのさまざまな加工が行われる
4. **返却**：結果を，ユーザーやアプリケーションに送る

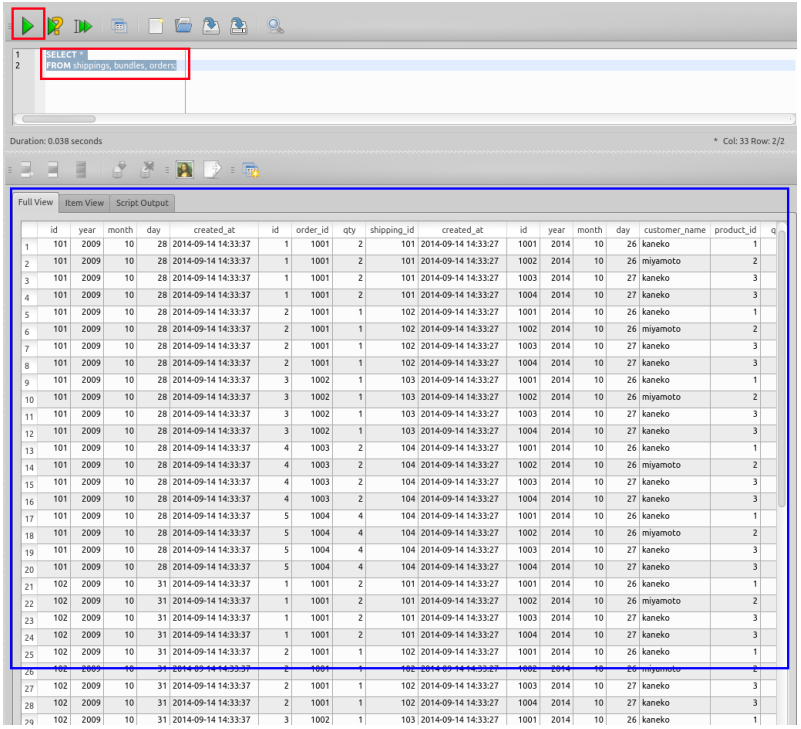
問い合わせ（クエリ）

問い合わせ（クエリ）の最終結果

データベースシステム

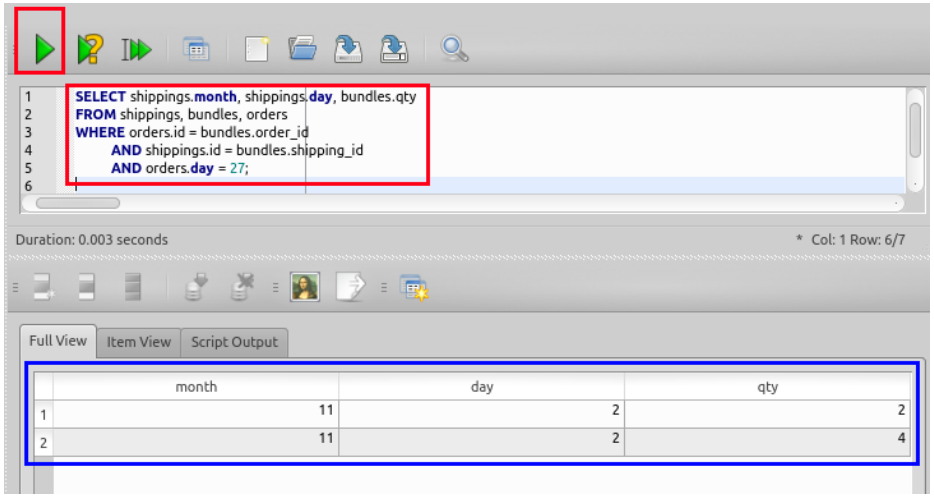
id	name	price	at
1	orange	50	-05-11 13:30:18
2	apple	100	-05-11 13:30:18
3	melon	500	-05-11 13:30:18
	緑 ZZ	貸出	2021-05-11 13:30:18

問い合わせ（クエリ）の例 ①
データの検索



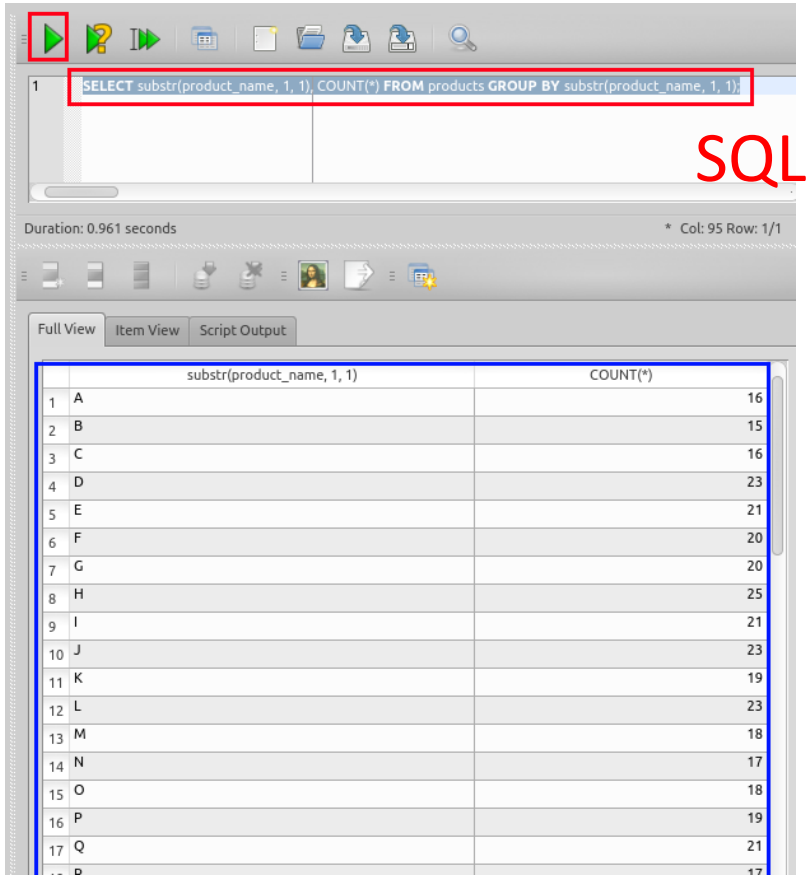
元データ

SQL

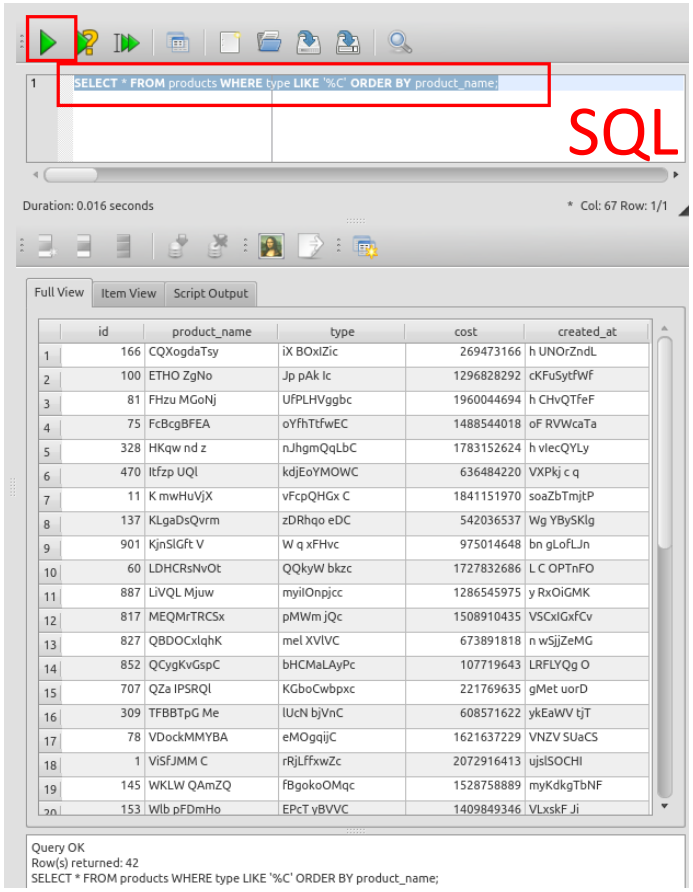


結果

問い合わせ（クエリ）の例 ②
集計・集約，並べ替え（ソート）



	substr(product_name, 1, 1)	COUNT(*)
1	A	16
2	B	15
3	C	16
4	D	23
5	E	21
6	F	20
7	G	20
8	H	25
9	I	21
10	J	23
11	K	19
12	L	23
13	M	18
14	N	17
15	O	18
16	P	19
17	Q	21



	id	product_name	type	cost	created_at
1	166	CQXogdaTsy	iX BOxlZic	269473166	h UNOrZndL
2	100	ETHO ZgNo	Jp pAk Ic	1296828292	cKFuSytfWF
3	81	FHzu MGoNj	UfPLHVggbc	1960044694	h CHvQTfeF
4	75	FcBcgBFEA	oYfhTtfwEC	1488544018	oF RVWcaTa
5	328	HKqw nd z	nJhgmQqLbC	1783152624	h vlecQYLy
6	470	Itfzp UQl	kdjEoYMOWC	636484220	VXPkj c q
7	11	K mwHuVjX	vFcpQHGx C	1841151970	soaZbTmjtP
8	137	KLgaDsQvrm	zDRhqo eDC	542036537	Wg YBySKlg
9	901	KjnSlGft V	W q xFHvc	975014648	bn gLofLJn
10	60	LDHCRsNvOt	QQkyW bkzc	1727832686	L C OPTnFO
11	887	LiVQL Mjuw	myilOnpjcc	1286545975	y RxOiGMK
12	817	MEQMrTRCSx	pMWm jQc	1508910435	VSCxIGxfCv
13	827	QBDOCxlqhK	mel XVlVC	673891818	n wSjjZeMG
14	852	QCygKvGspC	bHCMaLAyPc	107719643	LRFLYQg O
15	707	QZa IPSRQl	KGboCwbpxc	221769635	gMet uorD
16	309	TFBBTpG Me	lUcN bjVnC	608571622	ykEaWV tjT
17	78	VDockMMYBA	eMOgqijC	1621637229	VNZV SUaCS
18	1	ViSfJMM C	rRjLfFxwZc	2072916413	ujslSOCHI
19	145	WKLW QAmZQ	fBgokoOMqc	1528758889	myKdkgTbNF
20	153	Wlb pFDmHo	EPcT yBVVC	1409849346	VLxskF Ji

集計・集約

並べ替え（ソート）

SQLの文法

- **大文字小文字**：

SQLは大文字・小文字を**区別しない**（SELECTとselectは同じ）

- **改行**：

SQL文は途中で改行しても問題ない（改行はコードの可読性を高める）

- **セミコロン (;)**：
  - 一つのSQL文のみの場合，末尾の;は省略可能
  - ２つ以上のSQL文を連ねる場合は，;で文を区切る必要がある

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

SQL 問い合わせの全体像①

- **select**: **データの検索・加工や射影**

  例：**SELECT * FROM employees**
- **from**: 問い合わせ**対象テーブル**の指定

  例：**FROM employees**
- **where**: 条件に一致する行を**選択**

  例：**WHERE age > 30**
- **join, on**: 結合、結合条件

  例：**JOIN B ON A.b_id = B.id**
- **insert into**: 新しい行の**追加**（挿入）
- **update, set:** 条件に一致する行を**更新**
- **delete from**: 条件に一致する行を**削除**

SQL 問い合わせの全体像②

- **distinct:** 重複行の除去

  例：**SELECT DISTINCT age FROM employees**

- **count:** 行数のカウント

  例：**SELECT COUNT(*) FROM employees**

- **avg, max, min, sum**: 平均、最大、最小、合計の計算

  例：**AVG(salary), MAX(salary)**

- **group by**: 属性でグループ化

  例：**GROUP BY department_id**

- **order by:** 並べ替え（ソート）

  例：**ORDER BY age**

- 副問い合わせ: SQL文の中に別のSQL文を埋め込む。

  例：**WHERE salary > (SELECT AVG(salary) FROM employees)**

問い合わせ（クエリ）のバリエーション

① 全データの取得

select * from 商品;

② 特定の属性（列）のみ取得

select 商品名, 単価 **from** 商品;

③ 条件付き検索

select 商品名, 単価 **from** 商品 **where** 単価 > 80;

選択

選択は、特定の条件に一致する行を選択

元のテーブル
テーブル名: P

CUST	PRODUCT	PRICE
100	P100	20
101	P100	30
101	X200	1000
102	P300	100

誰が、何を、いくらで買ったか

選択

```
SELECT *
FROM P
WHERE PRICE > 50;
```

CUST	PRODUCT	PRICE
101	X200	1000
102	P300	100

選択と射影の組み合わせ

射影は、必要な属性のみを抽出

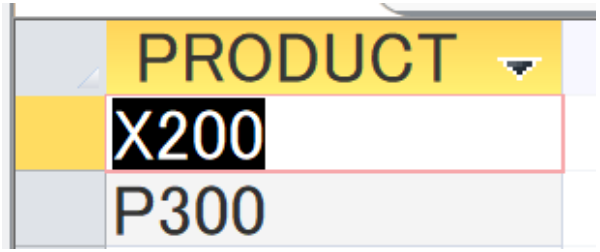
元のテーブル
テーブル名: P

CUST	PRODUCT	PRICE
100	P100	20
101	P100	30
101	X200	1000
102	P300	100

誰が、何を、いくらで買ったか

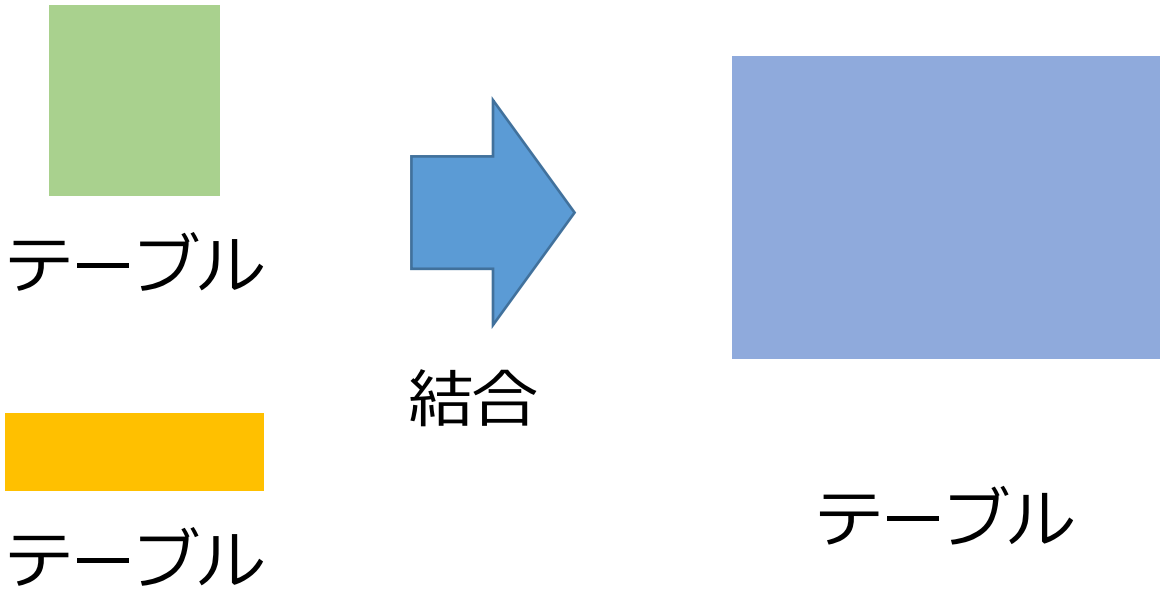
選択と射影の組み合わせ

```
SELECT PRODUCT
FROM P
WHERE PRICE > 50;
```



テーブル結合

結合は，テーブル間の関連に基づいて**複数のテーブル**を**1つにまとめる操作**
例：**従業員テーブル**と**部署テーブル**を結合，従業員の名前と所属部署の名前を**１つのテーブル**に集める．



重複行除去

元のテーブル
テーブル名: P

CUST	PRODUCT	PRICE
100	P100	20
101	P100	30
101	X200	1000
102	P300	100

誰が、何を、いくらで買ったか

重複行除去しない

```
SELECT CUST
FROM P;
```

CUST
100
101
101
102

重複行除去する

```
SELECT DISTINCT CUST
FROM P;
```

CUST
100
101
102

グループ化

GROUP BY句を使用．
指定した属性でデータをグループ化する機能．
各グループに対して集計関数（例：count）を適用

元のテーブル
テーブル名: P

CUST	PRODUCT	PRICE
100	P100	20
101	P100	30
101	X200	1000
102	P300	100

誰が、何を、いくらで買ったか

集計の例

```
SELECT CUST, COUNT(*)
FROM P
GROUP BY CUST;
```

100	1
101	2
102	1

CUST 属性について 100, 101, 102
のグループ化が行われる

並べ替え（ソート）

指定した属性についてソートする

元のテーブル
テーブル名: P

CUST	PRODUCT	PRICE
100	P100	20
101	P100	30
101	X200	1000
102	P300	100

誰が、何を、いくらで買ったか

並べ替え（ソート）

```
SELECT *
FROM P
ORDER BY PRICE;
```

CUST	PRODUCT	PRICE
100	P100	20
101	P100	30
102	P300	100
101	X200	1000

PRICE で並べ替え

副問い合わせ

元のテーブル
テーブル名: P

CUST	PRODUCT	PRICE
100	P100	20
101	P100	30
101	X200	1000
102	P300	100

誰が、何を、いくらで買ったか

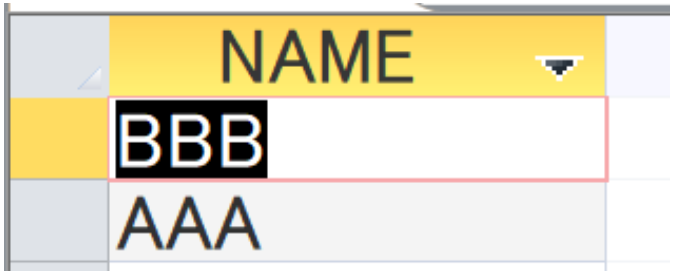
テーブル名: C

ID	NAME	MEMBER
100	BBB	2023/10/02
101	AAA	2023/10/10
102	CCC	2023/10/15

名簿（番号と氏名と入会日）

副問い合わせ

```
SELECT NAME
FROM C
WHERE ID IN
  (SELECT CUST
FROM P
WHERE PRODUCT='P100');
```



P100 を買ったのは誰？

15-5. 主キーと外部キー

主キー

- **主キー**は、**テーブルの各行を識別する**ためのキー

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

ID属性は主キーである

SQL によるテーブル定義

- **テーブル名**：**商品**
- **属性名**：**ID、商品名、単価**
- **属性のデータ型**：**数値、テキスト、数値**
- データの整合性を保つための**制約**：**主キー制約**

```
CREATE TABLE 商品 (
    ID INTEGER PRIMAY KEY,
    商品名 TEXT,
    単価 INTEGER);
```

主キー制約と PRIMARY KEY

PRIMARY KEY はテーブル定義時に使用し，「**主キー制約**」を示す

```
CREATE TABLE テーブル名 (
   列名1 データ型 PRIMARY KEY,
   列名2 データ型,
   …
);
```

SQL の書き方

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

主キー

```
CREATE TABLE 商品 (
   ID INTEGER PRIMAY KEY,
   商品名 TEXT,
   単価 INTEGER);
```

外部キー

外部キーは、**他のテーブルの主キーを参照する**キー

購入テーブル

外部キー

ID	購入者	商品ID	数量
1	X	1	10
2	Y	2	5

商品テーブル

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

主キー

購入テーブルの**外部キー**「商品ID」は、**購入**テーブルの**主キー**「ID」を**参照**

参照整合性制約のイメージ

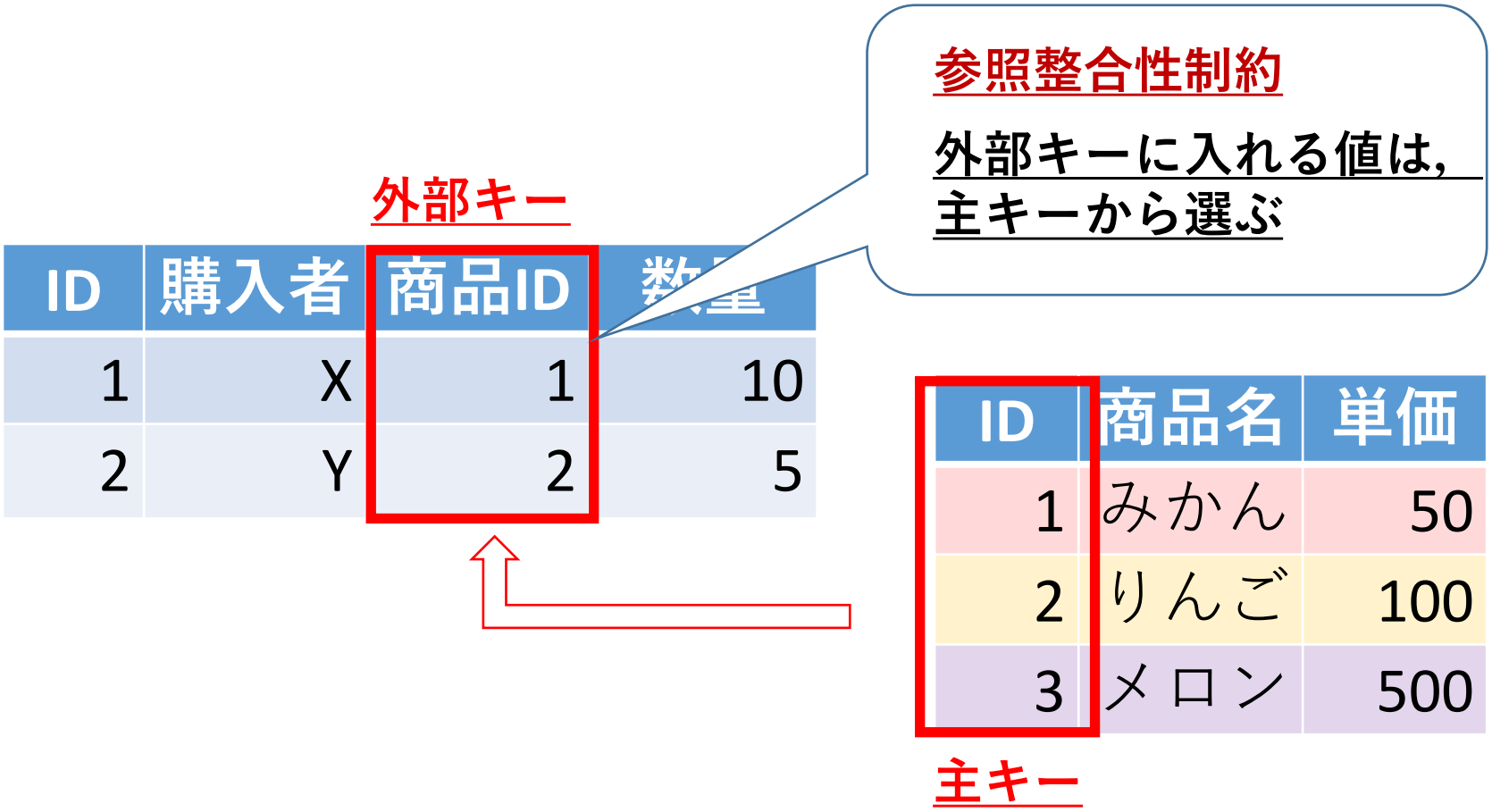
商品から
お選びください

枝豆はないんですか？

テーブルのデータを別のテーブルから参照するときの
制約として、**参照整合性制約**がある

参照整合性制約の例

商品テーブルに ID が存在しない場合、購入テーブルに、そのIDを持つ行を挿入することはできない



FOREIGN KEY … REFERENCES

PRIMARY KEY … REFERENCES はテーブル定義時に使用し、あるテーブルの**外部キー**が別のテーブルの**主キー**を**参照**する「**参照整合性制約**」を示す

```
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 商品(ID));
```

き方

外部キー

ID	購入者	商品ID	数量
1	X	1	10
2	Y	2	5

主キー

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

主キー

15-6. データベース操作とトランザクション

データベース操作の種類

INSERT（追加）

- テーブルに新しい行を追加する操作

SELECT（問い合わせ）

- 必要なデータを検索・加工する操作

DELETE（削除）

- テーブルから条件に合致する行をすべて削除する操作

UPDATE（更新）

- 条件に合致する部分について，値を変更する操作

普通のプログラミングでのデータ更新とデータベース操作の違い

	データベース操作	普通のプログラミング
①トランザクション管理	**可能**	困難
②データベースの整合性を保つ	**制約機能あり**	機能なし
③複数ユーザからの同時アクセス	**システム側で同時アクセス対応**	原則，ユーザのプログラミングで制御
④データの永続性	**あり**	実行中のみデータ保持

① トランザクション管理の基本

トランザクションの概念

- トランザクションは，複数のデータベース操作を一連の単一処理として扱う機能である．

例：口座間送金

```
BEGIN TRANSACTION;
UPDATE 口座 SET 残高 = 残高 - 1000 WHERE 口座番号 = 'A';  口座Aから1000円引き出す
UPDATE 口座 SET 残高 = 残高 + 1000 WHERE 口座番号 = 'B';  口座Bに1000円預け入れる
COMMIT;
```

すべての操作を**一つのトランザクション**として処理

②トランザクションの特徴

重要な性質

1.処理の独立性

1. トランザクション途中のデータは外部からは見えない

2.終了時の選択

1. **コミット (commit)**：全操作を反映
2. **ロールバック(rollback)**：全変更を取り消し

例：

```
BEGIN TRANSACTION;
UPDATE 口座 SET 残高 = 残高 - 1000 WHERE 口座番号 = 'A';   口座Aから1000円引き出す
UPDATE 口座 SET 残高 = 残高 + 1000 WHERE 口座番号 = 'B';   口座Bに1000円預け入れる
COMMIT;
```

注：トランザクションがコミットするまで,
中間状態は外部から見えない.

ロールバック (rollback) 実行の詳細

ロールバックの特徴

- rollbackコマンドの実行で全変更を取り消し
- データはトランザクション開始時の状態に復元

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

データウエアハウスとオンライントランザクションの比較

- **オンライントランザクション：最新情報のみを使用**

予約テーブル

氏名	予約内容
XX	おせちA
YY	おせちB
ZZ	おせちA

価格テーブル

商品	価格
おせちA	10000
おせちB	5000

- **データウエアハウス：「履歴データ」を重視する**

予約テーブル

氏名	予約内容	予約日	キャンセル日
XX	おせちA	2023-12-01	
YY	おせちB	2023-12-04	
ZZ	おせちB	2023-12-05	2023-12-06
ZZ	おせちA	2023-12-06	

価格テーブル

商品	価格	価格改定日
おせちA	12000	2023-11-10
おせちA	10000	2023-11-20
おせちB	5000	2023-11-10

履歴データの概念

- **データの各行に日時情報**を付加
- データ変化があるたびに新しい行を**挿入**
- **一度保存されたデータは削除しない（データベース操作は基本「挿入」のみ）**
- 過去のデータの変化の履歴を保持

予約テーブル

氏名	予約内容	予約日	キャンセル日
XX	おせちA	2023-12-01	
YY	おせちB	2023-12-04	
ZZ	おせちB	2023-12-05	2023-12-06
ZZ	おせちA	2023-12-06	

価格テーブル

商品	価格	価格改定日
おせちA	12000	2023-11-10
おせちA	10000	2023-11-20
おせちB	5000	2023-11-10

「商品の価格は１つに決まっている」という考え方ではなく、
商品の価格の履歴データを保持