13. データ管理の基礎：オンライントランザクションとデータウェアハウスの特徴と活用

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

アウトライン

1. イントロダクション
2. 日時の扱い
3. データ管理とデータウエアハウス
4. 演習

SQLFiddle のサイトにアクセス

Webブラウザを使用

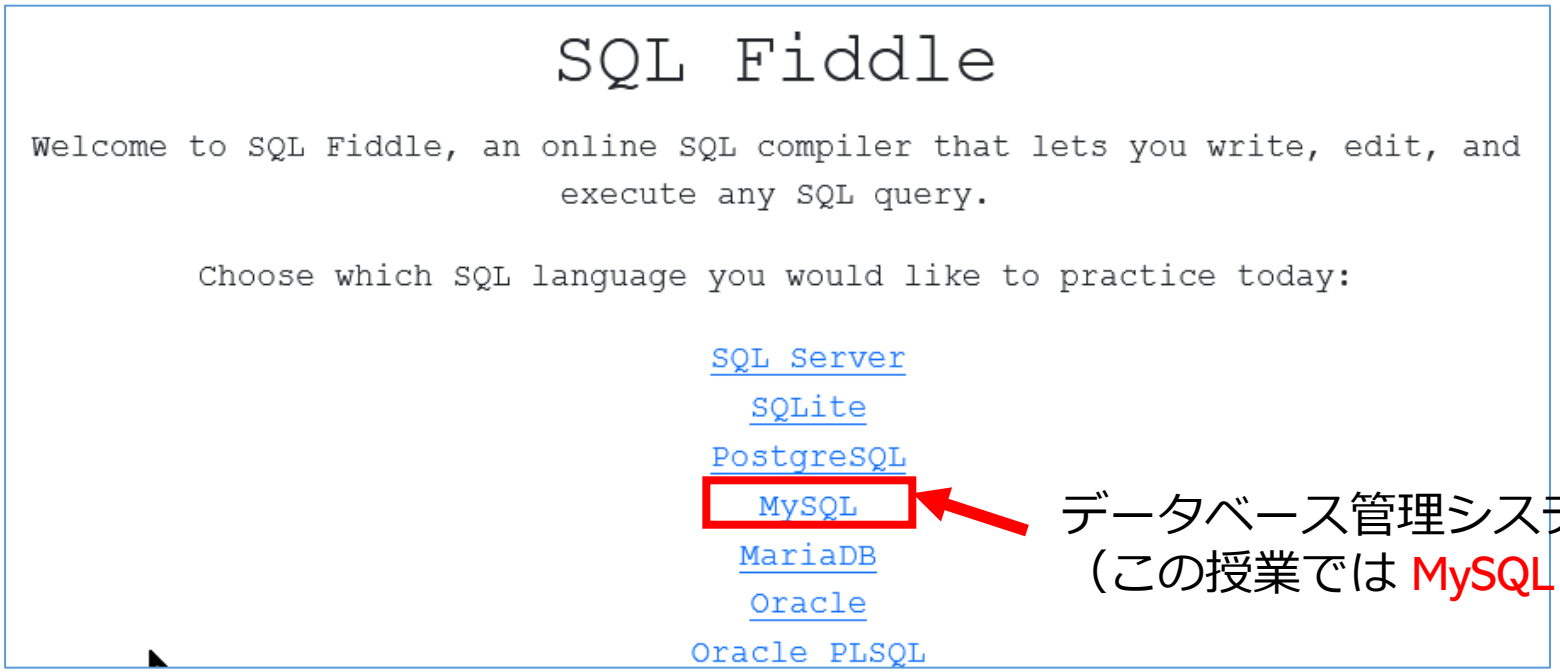
1. ウェブブラウザを開く
2. アドレスバーにSQLFiddleのURLを入力

http://sqlfiddle.com/

3. **MySQL を選ぶ**

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 でのデータベース管理システムの選択



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）

SQLFiddle の画面

上のパネル: SQLの入力（複数可能）

- **テーブル定義** CREATE TABLE
- **データの追加** INSERT INTO
- **SQL問い合わせ**。SELECT, FROM, WHERE など

13-1. 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

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。

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**
- SELECT, FROM, WHERE など、**多様**なコマンドが存在。
- **結合、集計、ソート、副問い合わせ**など、高度な操作も可能

SQL によるテーブル定義

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

```
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));
```

データ追加のSQL

商品

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

```
INSERT INTO 商品 VALUES(1, 'みかん', 50);
INSERT INTO 商品 VALUES(2, 'りんご', 100);
INSERT INTO 商品 VALUES(3, 'メロン', 500);
```

演習１．テーブル定義とデータの追加、主キー制約

【トピックス】

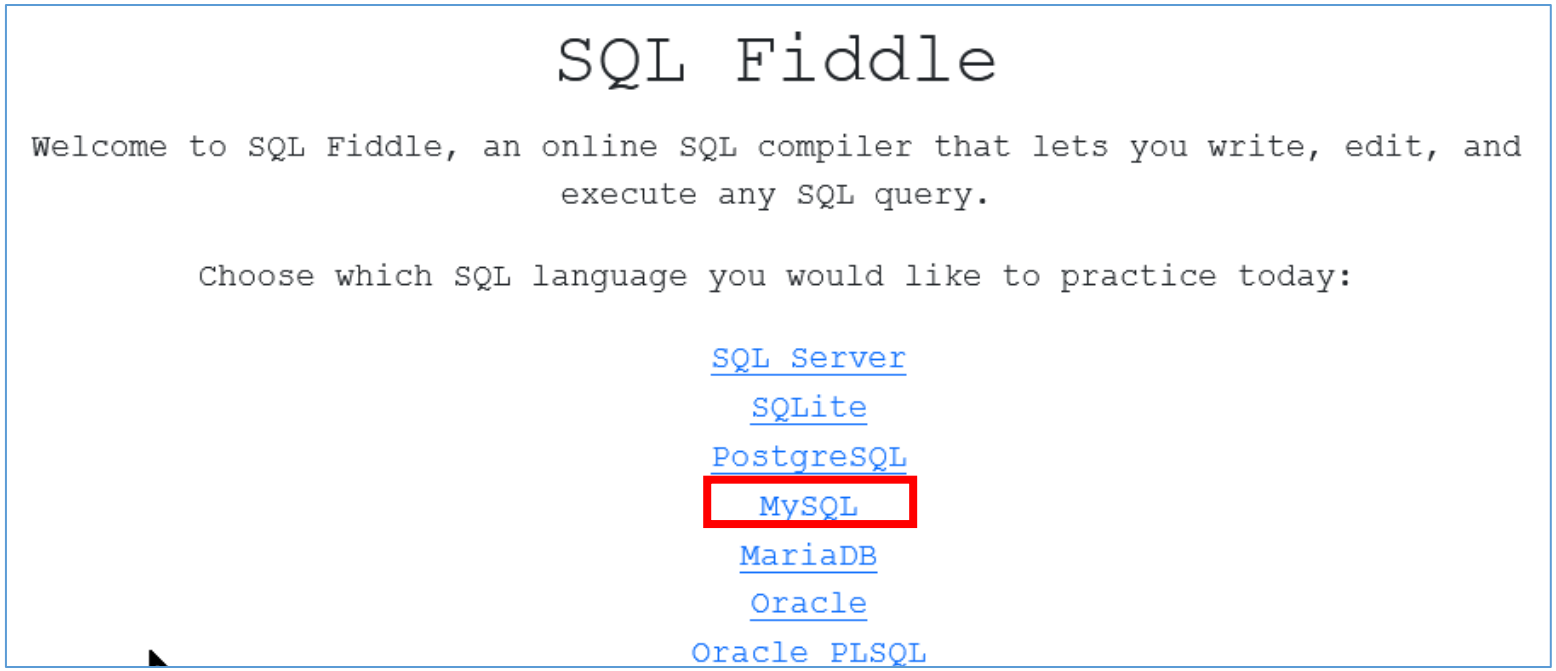
1. **SQL によるテーブル定義**
2. **主キー制約 PRIMARY KEY**
3. **SQL によるデータの追加**
4. **問い合わせ（クエリ）による確認**

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/

② 「**MySQL**」を選択



③　**上のパネル**に、**テーブル定義**と**データの追加**と**問い合わせ**を行う SQL を入れ実行。（**以前の SQL は不要なので消す**）

```
CREATE TABLE 商品 (
     ID INTEGER PRIMARY KEY,
     商品名 TEXT,
     単価 INTEGER);
INSERT INTO 商品 VALUES(1, 'みかん', 50);
INSERT INTO 商品 VALUES(2, 'りんご', 100);
INSERT INTO 商品 VALUES(3, 'メロン', 500);
select * FROM 商品;
```

④ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑤ 下側のウインドウで、**結果を確認**。

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

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**

SQL によるテーブル定義

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

```
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));
```

演習２．外部キー，参照整合性制約

【トピックス】

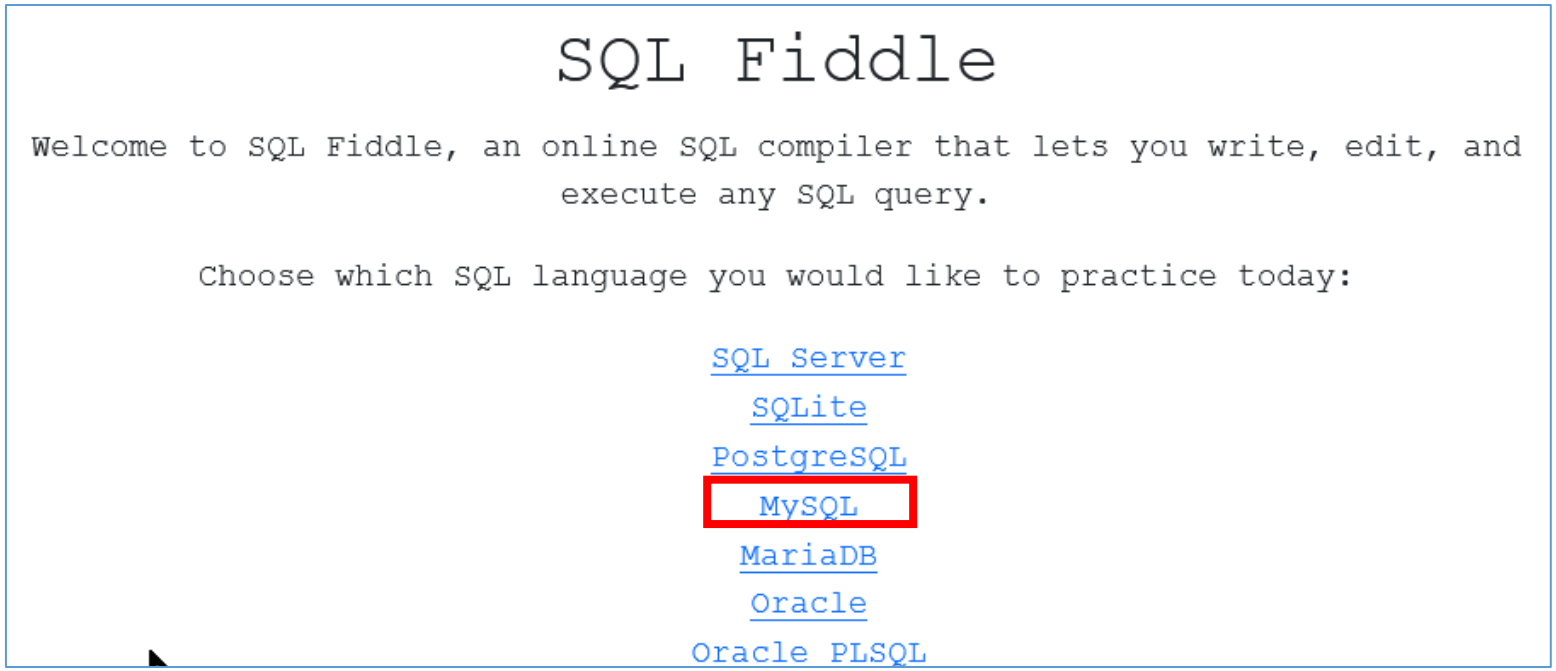
1. **主キー**
2. **外部キー**
3. **参照整合性制約**
4. **PRIMARY KEY**
5. **FOREIGN KEY … REFERENCES**

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/

② 「**MySQL**」を選択



③　**上のパネル**に、**テーブル定義**と**データの追加**と**問い合わせ**を行う SQL を入れ実行。（**以前の SQL は不要なので消す**）.

```
CREATE TABLE 商品 (
     ID INTEGER PRIMARY KEY,
     商品名 TEXT,
     単価 INTEGER);
INSERT INTO 商品 VALUES(1, 'みかん', 50);
INSERT INTO 商品 VALUES(2, 'りんご', 100);
INSERT INTO 商品 VALUES(3, 'メロン', 500);
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));
INSERT INTO 購入 VALUES(1, 'X', 1, 10);
INSERT INTO 購入 VALUES(2, 'Y', 2, 5);
select * FROM 商品;
select * FROM 購入;
```

④ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑤ 下側のウインドウで、**結果を確認**。

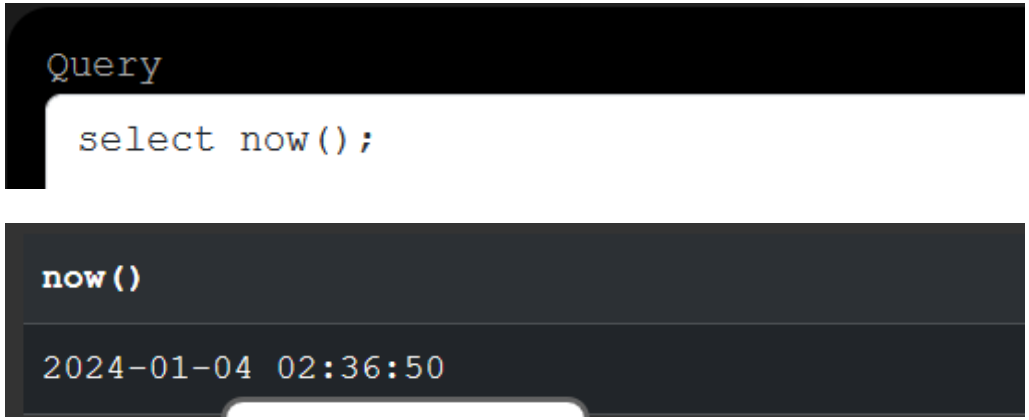
```
+----+-----------+--------+
| ID | 商品名 | 単価 |
+----+-----------+--------+
| 1 | みかん | 50 |
| 2 | りんご | 100 |
| 3 | メロン | 500 |
+----+-----------+--------+

+----+-----------+----------+--------+
| ID | 購入者 | 商品ID | 数量 |
+----+-----------+----------+--------+
| 1 | X | 1 | 10 |
| 2 | Y | 2 | 5 |
+----+-----------+----------+--------+
```

13-2. 日時の扱い

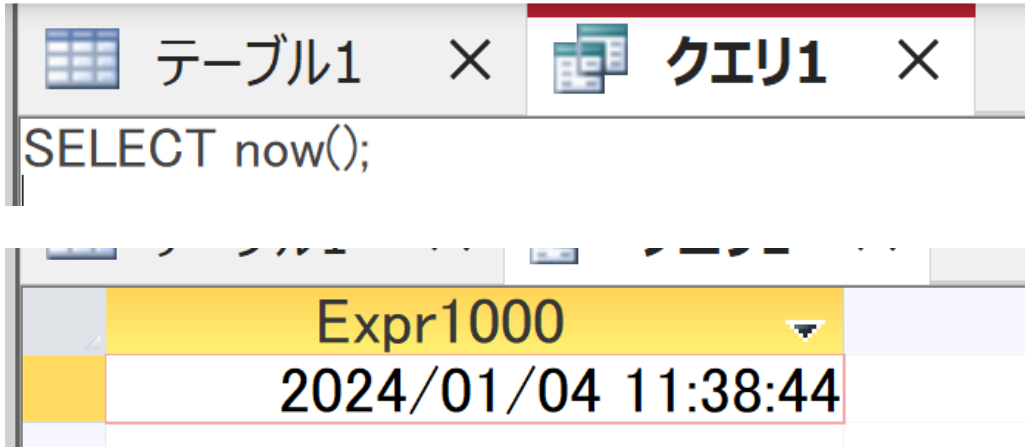
現在日時の取得方法

現在日時の取得に now() を使用する

- **SQL Fiddle の MySQL での実行結果**



- **Access での実行結果**



SQL によるテーブル定義

- **テーブル名**：**購入**
- **属性名**：**ID、購入者、商品ID、数量、購入日時**
- **属性のデータ型**：**数値、テキスト、数値、数値、日時**
- データの整合性を保つための**制約**：**主キー制約、参照整合性制約**

```
CREATE TABLE 購入 (
  ID INTEGER PRIMARY KEY,
  購入者 TEXT,
  商品ID INTEGER,
  数量 INTEGER,
  注文日時 DATETIME,
  FOREIGN KEY (商品ID) REFERENCES 商品(ID));
```

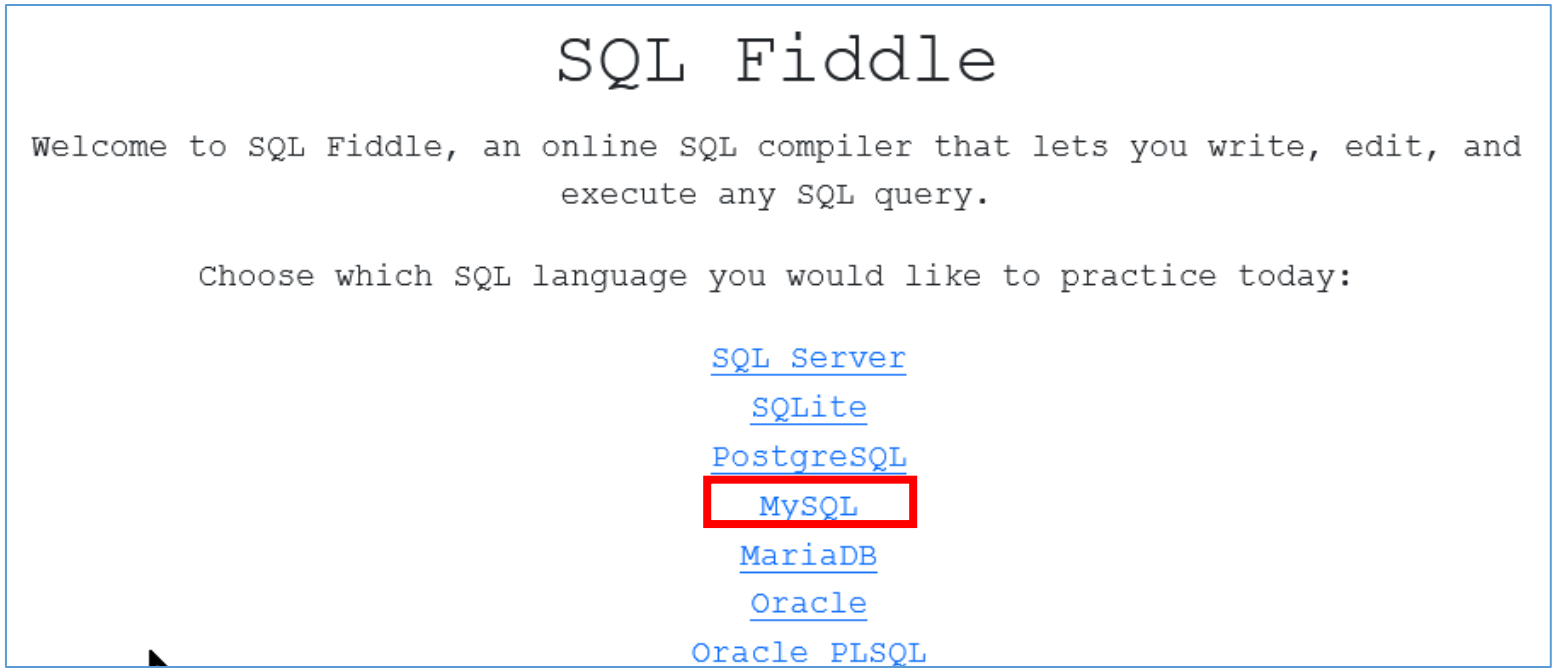
演習３．now() による現在日時の取得

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/

② 「**MySQL**」を選択



③　**上のパネル**に、**テーブル定義**と**データの追加**と**問い合わせ**を行う SQL を入れ実行。（**以前の SQL は不要なので消す**）.

```
select now();
```

④ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑤ 下側のウインドウで、**結果を確認**。

now()
2024-05-23 13:19:53

SQLFiddle では、間違った現在日時が得られる場合があるが、続行してください

演習４．日時を扱うテーブル

【トピックス】

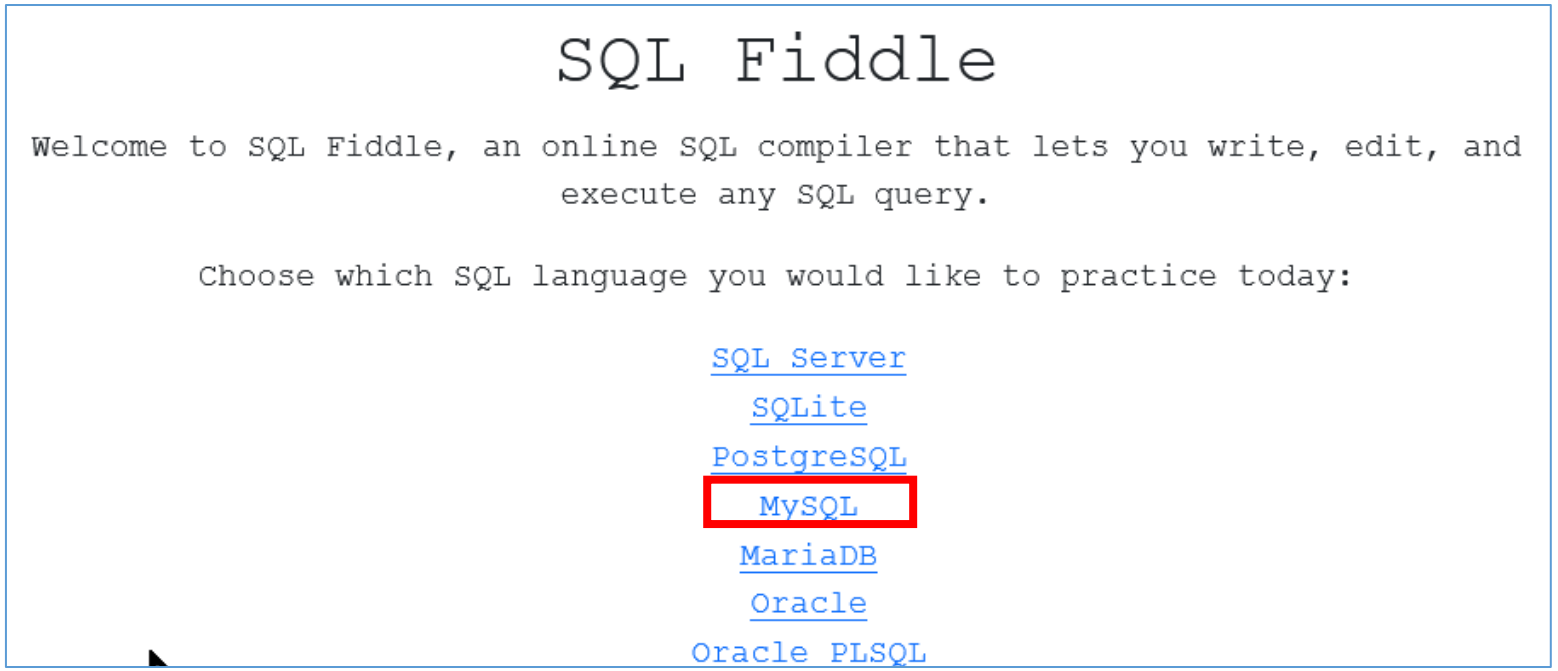
1. **日時**
2. **DATETIME**

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/

② 「**MySQL**」を選択



③　**上のパネル**に、**テーブル定義**と**データの追加**と**問い合わせ**を行う SQL を入れ実行。（**以前の SQL は不要なので消す**）.

```
CREATE TABLE 商品 (
    ID INTEGER PRIMARY KEY,
    商品名 TEXT,
    単価 INTEGER);
INSERT INTO 商品 VALUES(1, 'みかん', 50);
INSERT INTO 商品 VALUES(2, 'りんご', 100);
INSERT INTO 商品 VALUES(3, 'メロン', 500);
CREATE TABLE 購入 (
  ID INTEGER PRIMARY KEY,
  購入者 TEXT,
  商品ID INTEGER,
  数量 INTEGER,
  購入日時 DATETIME,
  FOREIGN KEY (商品ID) REFERENCES 商品(ID));
INSERT INTO 購入 VALUES(1, 'X', 1, 10, `2024-01-04 09:00:00');
INSERT INTO 購入 VALUES(2, 'Y', 2, 5, `2024-01-04 10:00:00');
select * FROM 商品;
select * FROM 購入;
```

④ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑤ 下側のウインドウで、**結果を確認**。

```
+----+----------+--------+
| ID | 商品名 | 単価 |
+----+----------+--------+
| 1 | みかん | 50 |
| 2 | りんご | 100 |
| 3 | メロン | 500 |
+----+----------+--------+
+----+----------+----------+--------+-----
| ID | 購入者 | 商品ID | 数量 | 購入日時 |
+----+----------+----------+--------+-----
| 1 | X | 1 | 10 | 2024-01-04 09:00:00 |
| 2 | Y | 2 | 5 | 2024-01-04 10:00:00 |
+----+----------+----------+--------+-----
```

ここまでのまとめ

- **SQL を用いた現在日時の表示**（MySQL や Access などで動く）

```
select now();
```

- **SQL での日時の書き方**

```
'2023-01-04 09:00:00'
```

- **日時を扱うテーブル定義**

```
CREATE TABLE 購入 (
  ID INTEGER PRIMARY KEY,
  購入者 TEXT,
  商品ID INTEGER,
  数量 INTEGER,
  購入日時 DATETIME,
  FOREIGN KEY (商品ID) REFERENCES 商品(ID));
```

13-3. データウエアハウス

データの一元管理と保存の利点

- **データの一元管理**

さまざまな種類のデータを一か所で管理することで、データアクセスと利用が容易になる。

- **データの長期保存**

長期間にわたるデータの安全な保存が容易になる

- **データ分析**

さまざまな種類のデータを組み合わせた分析が可能になる

データの一元管理と保存

２つの異なるアプローチ

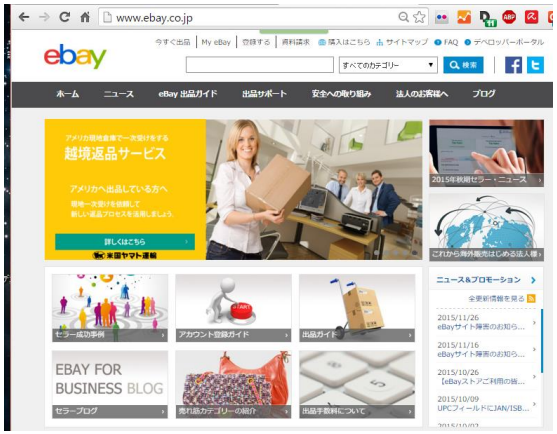
	データウエアハウス	オンライントランザクション
主な目的	データ分析、データからのルール発見	銀行の取引、オンライン予約、販売管理
機能	**長期間のデータを用いたデータ分析**	**トランザクション処理**
使用される技術	SQL、データの挿入、結合、グループ化	SQL、データの挿入と削除と更新、単純な問い合わせ
主な理念	**履歴データ**	**正規化**

データウエアハウスの活用例 ①

◆大量の商品が出品され、購入されるオークションサイトでの利用

- 不正行動の監視
- 出品者、購入者が「どうすればより満足するか」の分析

出品、購入の状況を丸ごと記録



eBay Web ページ
http://www.ebay.co.jp/

出典: Teradata 社 Web ページ
http://jpn.teradata.jp/library/nyumon/ins_1904.html

データウエアハウスの活用例 ②

◆「空席で飛ばすくらいなら、安値でも売りたい」と思っている航空会社

- 満席予測（空席数予測）の実施
- 空席をぴったり埋めるための動的な価格調整

予約、キャンセルの状況を丸ごと記録

出典: Teradata 社 Web ページ
http://jpn.teradata.jp/library/nyumon/ins_1904.html

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

データの正規化

- **データの値の変更や削除**を行う**オンライントランザクションシステム**では、**正規化は欠かせない**
- **正規化**は、**データの冗長性を排除**し、**データの整合性**を向上させる

正規化前

氏名	予約内容	価格
XX	おせちA	10000
YY	おせちB	5000
ZZ	おせちA	10000

冗長なデータがある

正規化後

氏名	予約内容
XX	おせちA
YY	おせちB
ZZ	おせちA

商品	価格
おせちA	10000
おせちB	5000

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.**

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**
- 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

データ管理のまとめ

データウエアハウス

- 「履歴データ」の利用
- データの時間の時間による分析を可能にする
- **一度保存されたデータは恒久的に保持**される
- 削除や変更は原則として行われない

オンライントランザクション

- 最新のデータを保持する
- データの値の変更や削除に伴い異状が発生する可能性がある.
- 正規化によりデータの冗長性を排除し，整合性を確保する

13-4. 演習

演習用データの概要

- 商品情報：ある商店は以下の商品を扱っている

  みかん、りんご、メロン

- 価格履歴管理：商品の単価は変化し，その履歴を以下のテーブルで管理する

商品テーブル

ID	商品名	単価	改訂日時
1	みかん	50	2024-12-01 09:00:00
2	りんご	100	2024-12-01 09:00:00
3	メロン	500	2024-12-01 09:00:00
4	りんご	150	2025-01-01 09:00:00
5	メロン	400	2025-01-01 09:00:00

- 購入を次のようなテーブルで扱う

購入テーブル

ID	購入者	商品ID	数量	購入日時
1	X	1	10	2024-12-10 10:00:00
2	Y	2	5	2024-12-20 12:00:00
3	Y	4	20	2025-01-05 09:00:00
4	Z	5	3	2025-01-05 11:00:00

演習５．データウエアハウスの構築

【トピックス】

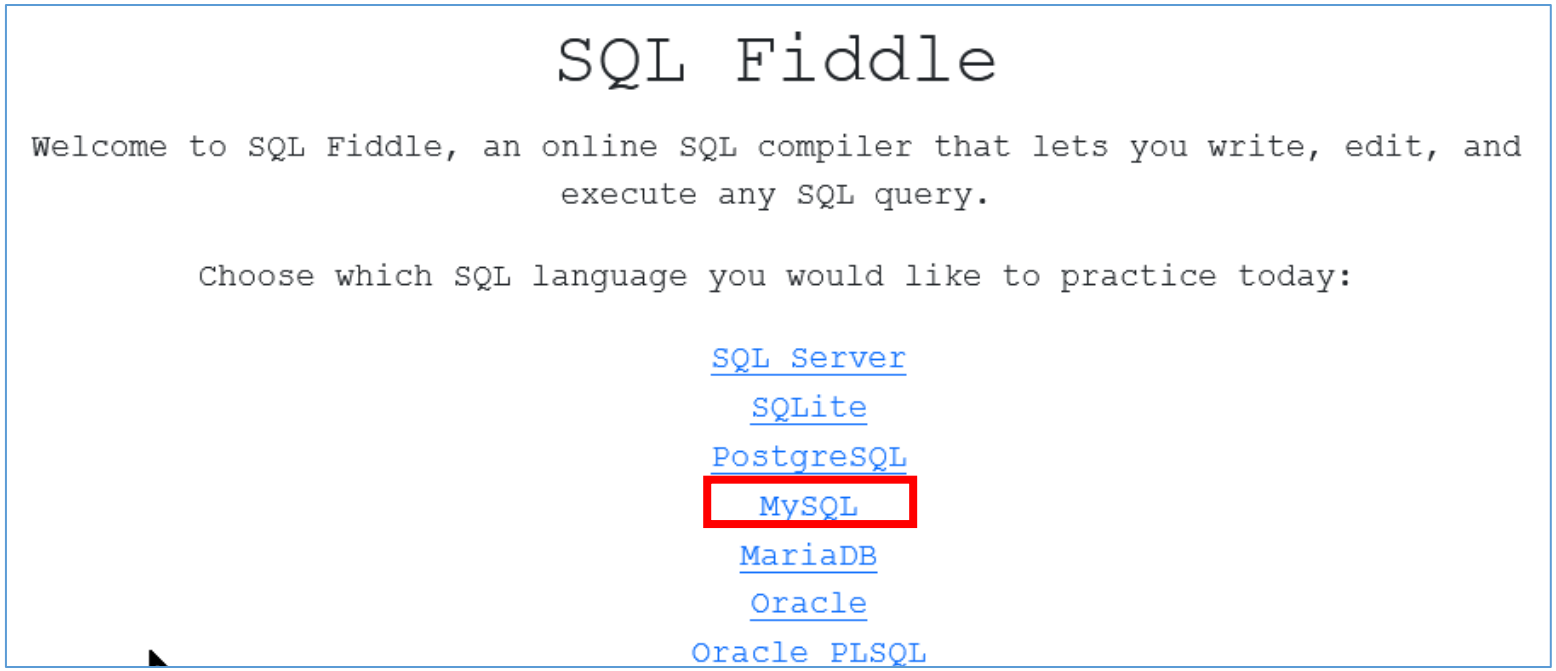
1. **日時**
2. **DATETIME**
3. **履歴データ**

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/

② 「**MySQL**」を選択



③　**上のパネル**に、**テーブル定義**と**データの追加**と**問い合わせ**を行う SQL を入れ実行。（**以前の SQL は不要なので消す**）.

```
CREATE TABLE 商品 (
    ID INTEGER PRIMARY KEY,
    商品名 TEXT,
    単価 INTEGER,
    改訂日時 DATETIME);
INSERT INTO 商品 VALUES(1, 'みかん', 50, `2024-12-01 09:00:00');
INSERT INTO 商品 VALUES(2, 'りんご', 100, `2024-12-01 09:00:00');
INSERT INTO 商品 VALUES(3, 'メロン', 500, `2024-12-01 09:00:00');
INSERT INTO 商品 VALUES(4, 'りんご', 150, `2025-01-01 09:00:00');
INSERT INTO 商品 VALUES(5, 'メロン', 400, `2025-01-01 09:00:00');
CREATE TABLE 購入 (
    ID INTEGER PRIMARY KEY,
    購入者 TEXT,
    商品ID INTEGER,
    数量 INTEGER,
    購入日時 DATETIME,
    FOREIGN KEY (商品ID) REFERENCES 商品(ID));
INSERT INTO 購入 VALUES(1, 'X', 1, 10, `2024-12-10 10:00:00');
INSERT INTO 購入 VALUES(2, 'Y', 2, 5, `2024-12-20 12:00:00');
INSERT INTO 購入 VALUES(3, 'Y', 4, 20, `2025-01-05 09:00:00');
INSERT INTO 購入 VALUES(4, 'Z', 5, 3, `2025-01-05 11:00:00');
select * FROM 商品;
select * FROM 購入;
```

④ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑤ 下側のウインドウで、**結果を確認**。

```
+----+----------+--------+--------------------+
| ID | 商品名 | 単価 | 改訂日時 |
+----+----------+--------+--------------------+
| 1 | みかん | 50 | 2024-12-01 09:00:00 |
| 2 | りんご | 100 | 2024-12-01 09:00:00 |
| 3 | メロン | 500 | 2024-12-01 09:00:00 |
| 4 | りんご | 150 | 2025-01-01 09:00:00 |
| 5 | メロン | 400 | 2025-01-01 09:00:00 |
+----+----------+--------+--------------------+
+----+----------+---------+--------+------------
| ID | 購入者 | 商品ID | 数量 | 購入日時 |
+----+----------+---------+--------+------------
| 1 | X | 1 | 10 | 2024-12-10 10:00:00 |
| 2 | Y | 2 | 5 | 2024-12-20 12:00:00 |
| 3 | Y | 4 | 20 | 2025-01-05 09:00:00 |
| 4 | Z | 5 | 3 | 2025-01-05 11:00:00 |
+----+----------+---------+--------+------------
```

④ **問い合わせ（クエリ）**を行う SQL を**追加**。 「Execute」をクリックして実行し、結果を確認

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

```
+----+----------+-------+---------------------+----+----------+---------+-------+--
| ID | 商品名 | 単価 | 改訂日時 | ID | 購入者 | 商品ID | 数量 | 購入日時 |
+----+----------+-------+---------------------+----+----------+---------+-------+--
| 1 | みかん | 50 | 2024-12-01 09:00:00 | 1 | X | 1 | 10 | 2024-12-10 10:00:00 |
| 2 | りんご | 100 | 2024-12-01 09:00:00 | 2 | Y | 2 | 5 | 2024-12-20 12:00:00 |
| 4 | りんご | 150 | 2025-01-01 09:00:00 | 3 | Y | 4 | 20 | 2025-01-05 09:00:00 |
| 5 | メロン | 400 | 2025-01-01 09:00:00 | 4 | Z | 5 | 3 | 2025-01-05 11:00:00 |
+----+----------+-------+---------------------+----+----------+---------+-------+--
```

⑤ **問い合わせ（クエリ）**を行う SQL を**追加**。 「Execute」をクリックして実行し、結果を確認

```
SELECT 購入.購入日時, 購入.購入者, 購入.数量 * 商品.単価
FROM 商品
INNER JOIN 購入 ON 商品.ID = 購入.商品ID;
```

購入日時	購入者	購入.数量 * 商品.単価
2024-12-10 10:00:00	X	500
2024-12-20 12:00:00	Y	500
2025-01-05 09:00:00	Y	3000
2025-01-05 11:00:00	Z	1200

⑥ **問い合わせ（クエリ）**を行う SQL を**追加**。 「Execute」をクリックして実行し、結果を確認

```
SELECT 購入.購入者, SUM(購入.数量 * 商品.単価)
FROM 商品
INNER JOIN 購入 ON 商品.ID = 購入.商品ID
GROUP BY 購入.購入者;
```

２つのテーブルを使い、購入者ごとに申し込みの合計金額を求める

購入者	SUM(購入.数量 * 商品.単価)
X	500
Y	3500
Z	1200

⑦ **問い合わせ（クエリ）**を行う SQL を**追加**。 「Execute」をクリックして実行し、結果を確認

```
SELECT 商品名, MAX(改訂日時)
FROM 商品
GROUP BY 商品名;
```

各商品の最新の
改訂日時を得ている。

商品名	MAX(改訂日時)
みかん	2024-12-01 09:00:00
りんご	2025-01-01 09:00:00
メロン	2025-01-01 09:00:00

⑧ **問い合わせ（クエリ）**を行う SQL を**追加**。 「Execute」をクリックして実行し、結果を確認

```
SELECT 商品名, 単価, 改訂日時
FROM 商品
WHERE (商品名, 改訂日時) IN (
   SELECT 商品名, MAX(改訂日時)
   FROM 商品
   GROUP BY 商品名);
```

副問い合わせで、
最新の改訂日時である
行を得る。その結果
を用いて最新価格を得る

```
+----------+--------+---------------+
| 商品名 | 単価 | 改訂日時 |
+----------+--------+---------------+
| みかん | 50 | 2024-12-01 09:00:00 |
| りんご | 150 | 2025-01-01 09:00:00 |
| メロン | 400 | 2025-01-01 09:00:00 |
+----------+--------+---------------+
```

発展演習１．Yによる購入

購入テーブルを用いて、購入者が 「Y」 のすべての購入情報を得るSQLを作成しなさい

ID	購入者	商品ID	数量	購入日時
2	Y	2	5	2024-12-20 12:00:00
3	Y	4	20	2025-01-05 09:00:00

ヒント：SELECT を使用

発展演習２．商品「りんご」を購入した人の取得

商品名が 「**りんご**」である商品を購入したすべての購入者を得るSQLを作成しなさい。 DISTINCT による重複行の除去も行うこと。

購入者
Y

ヒント：SELECT、DISTINCT、INNER JOIN、WHERE を使用

発展演習３．購入者別の申し込み数の計算

目的：購入者ごとに、申し込みの回数を得る

購入テーブルを使用して、購入者ごとに、購入の回数を得る SQL を作成しなさい。

購入者	COUNT(*)
X	1
Y	2
Z	1

ヒント：COUNT と GROUP BY を使用

解答例

発展演習１.

```
SELECT *
FROM 購入
WHERE 購入者 = 'Y';
```

```
+----+----------+---------+--------+---
| ID | 購入者 | 商品ID | 数量 | 購入日時 |
+----+----------+---------+--------+---
| 2 | Y | 2 | 5 | 2024-12-20 12:00:00 |
| 3 | Y | 4 | 20 | 2025-01-05 09:00:00 |
+----+----------+---------+--------+---
```

発展演習２.

```
SELECT DISTINCT(購入.購入者)
FROM 購入
INNER JOIN 商品 ON 購入.商品ID = 商品.ID
WHERE 商品.商品名 = 'りんご';
```

```
+----------+
| 購入者 |
+----------+
| Y |
+----------+
```

解答例

発展演習３.

```
SELECT 購入者, COUNT(*)
FROM 購入
GROUP BY 購入者;
```

```
+----------+----------+
| 購入者 | COUNT(*) |
+----------+----------+
| X | 1 |
| Y | 2 |
| Z | 1 |
+----------+----------+
```

全体まとめ

- **オンライントランザクション**では、**リアルタイムのデータ処理**により、**オンラインでの情報共有**を行う
- **データウェアハウス**は**「履歴データ」を重視**し、**データは一度格納されると削除、変更されることなく保存**される
- **日時**の属性を用いて**履歴を管理**できるようになる
- **SQL**では、「**DATETIME**」を使って**日時**を扱うことができる
- **日時のデータ**は SQL では、「**'2025-01-05 09:00:00'**」のような形式で表される
- MySQL や Access では、「**select now();**」により、現在の日時を取得できる