11. データベース操作とトランザクション管理：データ整合性と永続性

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. データ操作
3. INSERT, DELETE, UPDATE
4. トランザクション
5. ロールバック

SQLFiddle のサイトにアクセス

Webブラウザを使用

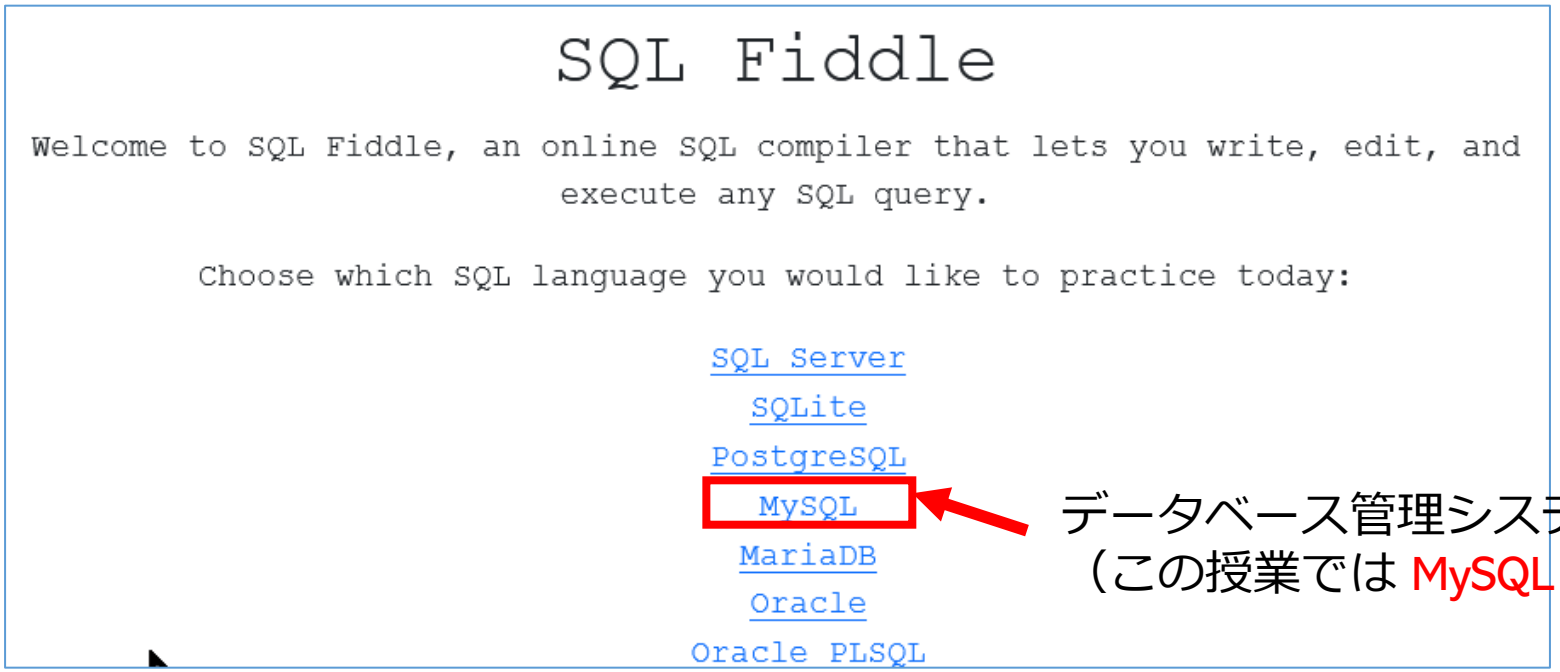
1. ウェブブラウザを開く
2. アドレスバーにSQLFiddleのURLを入力

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

11-1. 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。

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);
```

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);
```

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

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));
```

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

演習１．テーブル定義，主キー，外部キー，参照整合性制約

【トピックス】

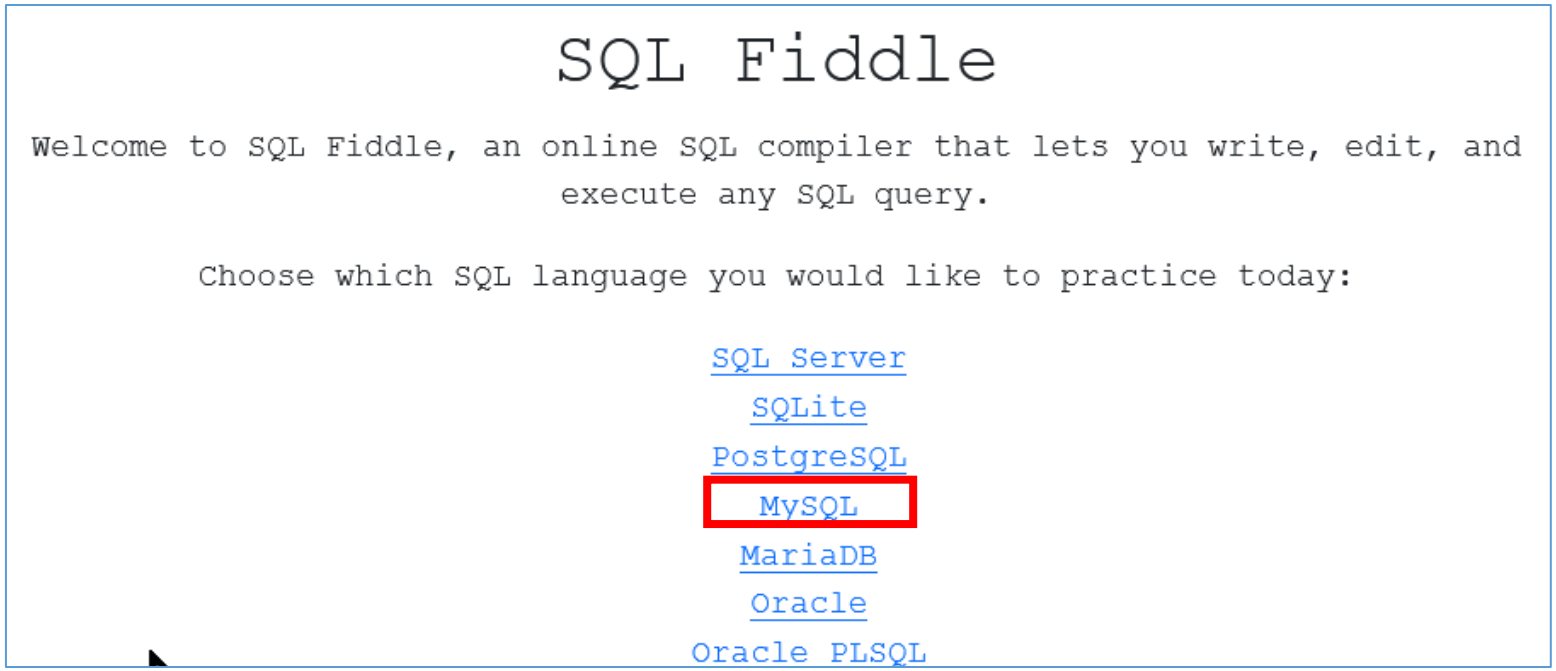
1. **主キー**
2. **外部キー**
3. **参照整合性制約**
4. **PRIMARY KEY**
5. **FOREIGN KEY … REFERENCES**

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);
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 下のパネルで、**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

⑥　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 購入;
```

⑦ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑧ 下のパネルで、**結果を確認。**

ID	購入者	商品ID	数量
1	X	1	10
2	Y	2	5

⑨　**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 購入.購入者, 商品.商品名, 商品.単価 * 購入.数量
FROM 購入
INNER JOIN 商品 ON 購入.商品ID = 商品.ID;
```

⑩「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑪ 下のパネルで、**結果を確認。**

購入者	商品名	商品.単価 * 購入.数量
X	みかん	500
Y	りんご	500

２つのテーブルの利用により，
各購入者の購入商品とその総額が分かる

発展演習１．**みかんの単価を検索する演習**

目的：特定の商品の単価を調べる方法を学ぶ

商品テーブルから「みかん」の単価を見つける SQL 文を作成

ヒント

- SELECT文とWHERE句を使用
- 商品名が「みかん」という条件を指定

発展演習２．**購入者Xの合計金額計算**

目的：複数テーブルの結合と計算機能を使用する

購入者Xが購入した商品の合計金額を計算するSQL文を作成。
合計金額は、各商品の単価と購入数量を掛け合わせて求める。

ヒント

・購入テーブルと商品テーブルの結合

・商品の単価と数量の積の合計を計算： SUM(商品.単価 * 購入.数量)

発展演習３．**「りんご」の購入者を特定**

目的：特定商品を購入者を検索する方法

「りんご」を購入した購入者の名前を検索する SQL 文を作成

ヒント

- 購入テーブルと商品テーブルの結合が必要
- WHERE句で商品名を指定

正解例

発展演習１.

```
SELECT 単価 FROM 商品 WHERE 商品名 = 'みかん';
```

```
+--------+
| 単価 |
+--------+
| 50 |
+--------+
```

発展演習２.

```
SELECT SUM(商品.単価 * 購入.数量)
FROM 購入
INNER JOIN 商品 ON 購入.商品ID = 商品.ID
WHERE 購入者 = 'X';
```

```
+----------------------------
| SUM(商品.単価 * 購入.数量) |
+----------------------------
| 500 |
+----------------------------
```

発展演習３.

```
SELECT 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 |
+------------+
```

11-2. データベース操作

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

データベース操作の重要性

データベースでのデータ維持

- **データを最新の状態に保つ**ための効率的な方法

データの整合性とセキュリティの確保

- 正しいデータベース操作により，**データの整合性とセキュリティ**を維持

データベース操作で留意点

● データ整合性の確保

正確な条件を指定によるデータ整合性の維持

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

正規化されていない
テーブルでは、
特に注意が必要

● トランザクション管理

複数の操作を一連の単一処理として扱うためのトランザクションを管理する．データの一貫性を確保する

```
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;
```

トランザクションは、「**複数の操作を一連の単一処理として扱うこと**」を意味する

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

データベース操作のまとめ

基本操作の要点

- 追加：INSERT による新規データ追加
- 問い合わせ：SELECT によるデータ検索・取得
- 更新：UPDATE による既存データ変更
- 削除：DELETE によるデータ削除

操作時の留意点

- データ整合性の維持
- トランザクション管理の重要性

データベースの特徴

- データの永続性
- 複数ユーザの同時アクセス対応
- トランザクションと制約のサポート

11-3. INSERT、DELETE、UPDATE

INSERT の基本（データの追加）

INSERT の基本形式

INSERT INTO テーブル名 VALUES (値1, 値2, ...);

名前	昼食	料金

空

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

テーブル T

```
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);
```

DELETE の基本（データの削除）

DELETE の基本形式

DELETE FROM テーブル名 WHERE 条件;

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	300

テーブル T

名前	昼食	料金
B	カレーライス	400
C	カレーライス	400
D	うどん	300

テーブル T

```
delete from T where 名前 = 'A';
```

UPDATE の基本（データの更新）

UPDATE の基本形式

UPDATE テーブル名 SET 列1 = 値1, 列2 = 値2, ... WHERE 条件;

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**

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**	**300**

テーブル T

```
update T set 料金 = 300 where 昼食 = 'うどん';
```

データベース操作の SQL まとめ

操作	SQL の例
テーブルに新しい行を追加	insert into T values('A', 'そば', 250);
テーブルから一つまたは複数の行を削除	delete from T where 名前 = 'A';
既存の値を変更	update T set 料金 = 300 where 昼食 = 'うどん';

データベース操作における注意点

INSERT（追加）での注意点

- **主キー**の列に重複する値を挿入することはできない

DELETE（削除），UPDATE（更新）での注意点

1. データの誤操作防止
    1. 必要なデータの誤削除防止
    2. 正確なデータの保護
2. ロールバック機能の活用
    1. **トランザクション終了前であれば変更取り消し**可能
    2. 誤操作時のデータ復旧手段として利用

SQLと他のプログラミング言語の違い

SQLでの更新例

```
BEGIN TRANSACTION;
UPDATE products SET price = 25 WHERE name = 'banana';
COMMIT;
```

特徴：トランザクション管理が可能

Pythonでの更新例

```
for item in products:
    if item['name'] == 'banana':
        item['price'] = 25
```

特徴：トランザクション管理が困難

データベース操作の総括

INSERT（追加）

- テーブルに新しい行を追加
- 形式：INSERT INTO テーブル名 VALUES (値1, 値2, ...);

SELECT（問い合わせ）

- 必要なデータを検索・加工
- 形式：SELECT ... FROM ... WHERE ...

DELETE（削除）

- テーブルから行を削除
- 形式：DELETE FROM テーブル名 WHERE 条件;

UPDATE（更新）

- 既存の値を変更
- 形式：UPDATE テーブル名 SET 列1 = 値1, ... WHERE 条件;

演習２．データベース操作の実践

【トピックス】

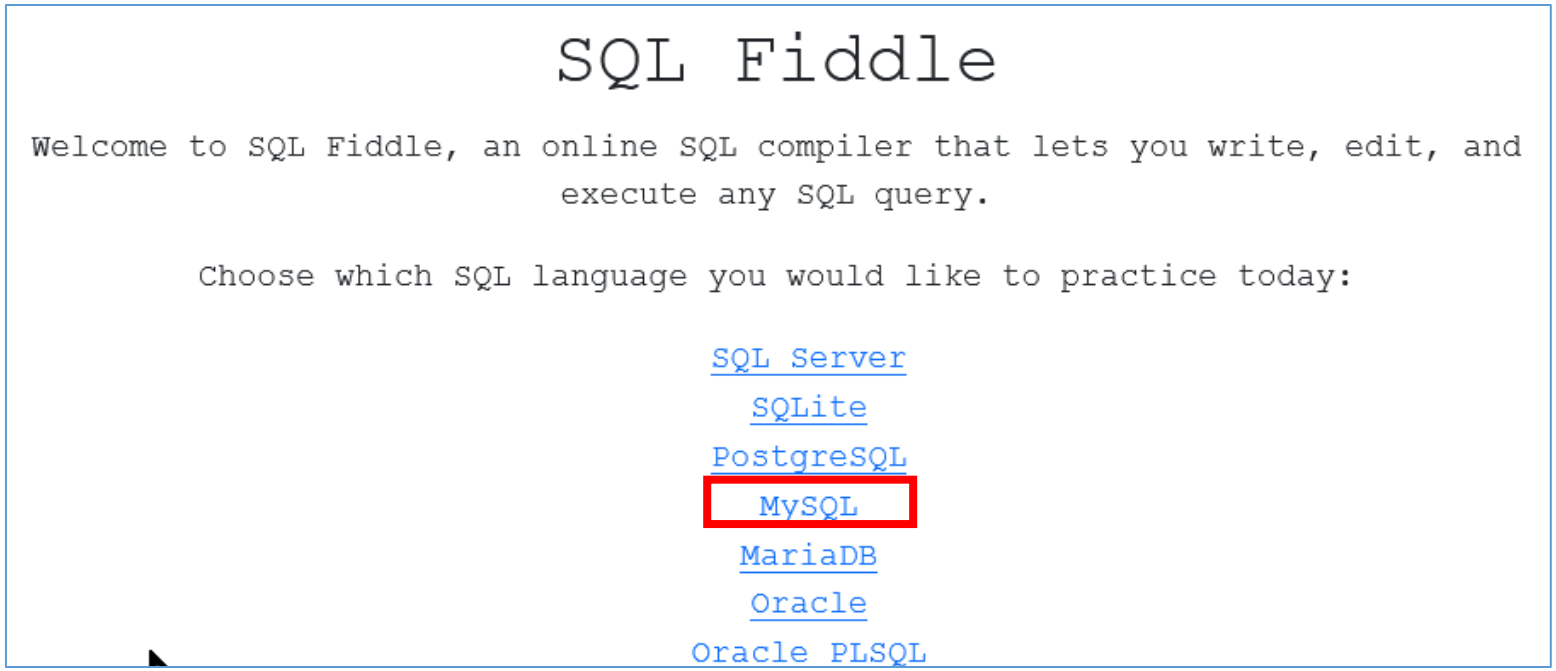
1. SQL によるテーブル定義
2. 追加：INSERT
3. 削除：DELETE
4. 更新：UPDATE
5. 問い合わせ（クエリ）による確認

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 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);
SELECT * FROM T;
```

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。**

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

⑥　**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 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);
UPDATE T SET 料金 = 450 WHERE 昼食 = 'カレーライス';
SELECT * FROM T;
```

カレーライスを 450円に変更

⑦ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑧ 下側のウインドウで、**結果を確認。**

名前	昼食	料金
A	そば	250
B	カレーライス	450
C	カレーライス	450
D	うどん	250

⑨　**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 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);
UPDATE T SET 昼食 = 'ラーメン' WHERE 名前 = 'C';
SELECT * FROM T;
```

名前が'C'の人の昼食を「ラーメン」に変更

⑩ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑪ 下側のウインドウで、**結果を確認。**

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

⑫　**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 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);
DELETE FROM T WHERE 名前 = 'B';
SELECT * FROM T;
```

名前が'B'の行を削除

⑬ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑭ 下側のウインドウで、**結果を確認。**

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

発展演習４．**「そば」と「うどん」の料金を変更する**

目的：特定の条件に合致する複数の行のデータを更新する方法を学ぶ。

「そば」と「うどん」の料金を300円に更新してください。

ヒント：WHEREを使って選択し、UPDATEで料金を変更します。

発展演習５．**新しいメニュー項目の追加**

目的：新しいレコードをテーブルに追加する方法を学ぶ。

新しく「E」さんが、「天ぷら」を料金500円を食べたという情報を、Tテーブルに追加してください

ヒント：INSERT INTOを使用

正解例と解説

発展演習４.

UPDATE T SET 料金 = 300 WHERE 昼食 = 'そば' OR 昼食 = 'うどん';

Tテーブル内で昼食が「そば」または「うどん」であるすべての行の料金を300円に更新します。

名前	昼食	料金
A	そば	300
B	カレーライス	400
C	カレーライス	400
D	うどん	300

発展演習５.

INSERT INTO T VALUES ('E', '天ぷら', 500);

このSQL文は新しい行をTテーブルに追加し、名前は「E」、昼食は「天ぷら」、料金は500円となります。

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
E	天ぷら	500

11-4. トランザクション

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.

② データベースの整合性を保つ仕組み

- **主キー制約**
- **参照整合性制約**　など

メニューから
お選びください

枝豆はないんですか？

制約の効果

- すべてのデータベース操作で制約違反がチェックされる
- データの一貫性が保証される

③ データベース制約の動作

- **制約の実行メカニズム**

1.データベース操作時の**制約チェック**

すべての操作で制約違反がないか確認される

2.制約違反時の処理

違反する操作は自動的に拒否される

3.ロールバックの自動実行

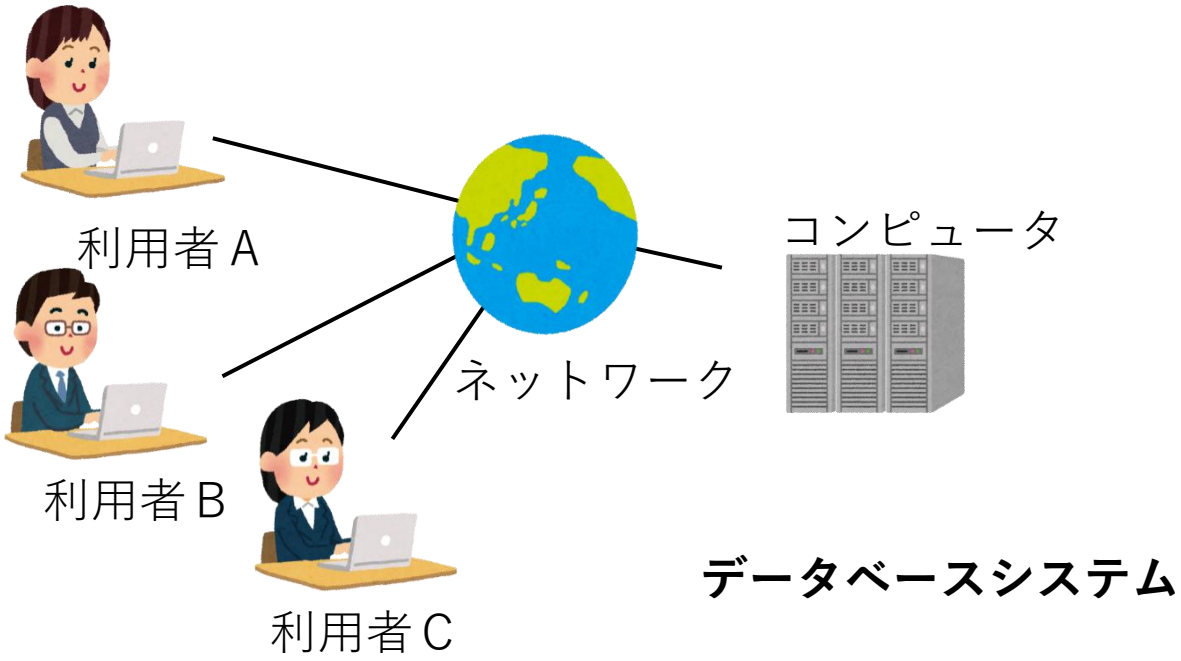
制約違反が発生した場合，**トランザクション開始後のすべての変更が自動的に取り消される**

注：この仕組みにより，データベースの一貫性が常に維持される.

```
BEGIN TRANSACTION;
UPDATE 口座 SET 残高 = 残高 - 1000 WHERE 口座番号 = 'A';
UPDATE 口座 SET 残高 = 残高 + 1000 WHERE 口座番号 = 'B';
COMMIT;
```

たとえ，COMMIT と書いたとしても，制約違反があれば，自動的にロールバックされ，全体が取り消される

④ 同時アクセスの制御



データベースシステムが同時実行される操作の干渉を防ぎ，
データの整合性を保持する．

⑤ データの永続性保証

ユーザ

データベース管理システム

永続性の仕組み

データベース管理システムの役割

- コミットされたトランザクションの結果を永続的に保存
- システム障害からのデータ保護

ユーザー視点では

- コミット済みデータの確実な保存
- データの信頼性確保

トランザクション制御のSQL文

- **トランザクション開始**
  begin transaction

- **ロールバック実行**
  rollback

- **コミット実行**
  commit

トランザクション処理のまとめ

主要な特徴

トランザクションの管理

- 複数操作を単一処理単位として扱う
- 処理の一貫性を保証
- 途中経過は外部に非公開

制約とトランザクション

- 制約違反時は自動ロールバック
- データの整合性を保持

同時アクセス

- 複数ユーザーの操作を適切に制御
- データの一貫性を維持

11-5. ロールバック

ロールバックの基本概念

特徴と機能

ロールバックとは

- **トランザクション開始**時点への**データ状態の復元**機能

システムの特徴

- リレーショナルデータベース管理システムの**標準機能**
- **他のユーザーへの影響なし**
- 特定のトランザクションのみを対象とした処理

ロールバック (rollback) の動作イメージ

begin transaction
操作１
操作２
操作３
rollback

処理の流れ

•基本的な動作

1.複数の操作を実行

2.必要に応じてrollbackコマンドを実行

3.トランザクション開始時点の状態に戻る

注：ロールバックにより，
すべての操作が取り消される.

トランザクション開始処理

「begin transaction」トランザクション開始コマンド

ユーザ

begin transaction

データベース管理システム

トランザクション実行中の状態

トランザクション実行中は，各操作は一時的な変更として扱われる

ユーザ

begin transaction

操作１

操作２

データベース管理システム

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

ロークバック発生のタイミング

主な発生条件

1.明示的なrollbackコマンドの実行

2.制約違反による自動ロールバック

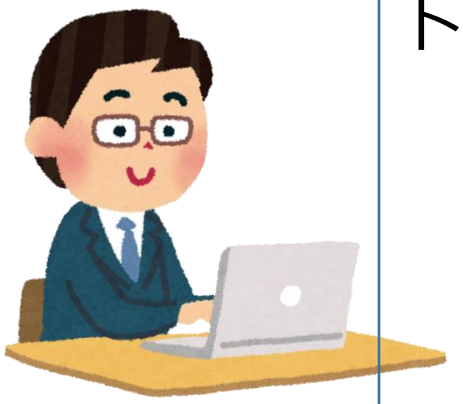
3.システムダウン時の自動復旧

重要性

- データベースの整合性を維持するための重要な機能として機能する.

システムダウン時の処理

データベース管理システムの責任で,
全トランザクションの未完了の操作をすべて取り消す.



トランザクション開始 ← 作業のスタート

操作１

操作２

操作３

作業のやりかけ ★

もしここで,
システムがダウン
→　システムの**再起動時に**
自動でロールバック

全体まとめ①

基本操作のまとめ

INSERT

- 新規データの追加
- 形式：INSERT INTO テーブル名 VALUES (値1, 値2, ...);

UPDATE

- 既存データの変更
- 形式：UPDATE テーブル名 SET 列1 = 値1, ... WHERE 条件;

DELETE

- データの削除
- 形式：DELETE FROM テーブル名 WHERE 条件;

全体まとめ②

トランザクション管理
- 複数操作の一括処理
- BEGIN TRANSACTION開始，COMMIT終了

データ整合性
- 主キー制約と参照整合性制約
- 制約違反時の自動ロールバック

同時アクセス
- 複数ユーザーの同時操作対応，干渉の防止

データ永続性
- コミット後のデータ永続保存，システム障害からの保護

ロールバック機能
- 不要操作の取り消し，システム障害時の自動実行