pd-7. データベース, JSON, NoSQL データベース

（Python による ICT システム）

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

金子邦彦

アウトライン

1. データベース
2. SQLプログラムの例
3. JSON、JSON とデータベースの連携
4. データベースシステムの種類とNoSQL データベース

7-1 データベース

データベース

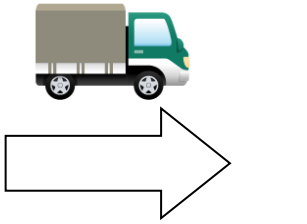
データベースは、特定の主題について**整理，保存，管理**されたデータの集合体

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

データベースの利用分野
①オンラインコミュニケーション

オンラインコミュニケーションでの
データ管理．機能がより便利に．

- **SNS（ソーシャルネットワーク）**

投稿、ユーザプロフィール、「いいね」、コメントの管理

- **電子メール**

本文、添付ファイル、送信者、受信者の管理

- **オンラインのチャット**

ユーザ間のメッセージの管理

データベースの利用分野
②オンラインの取引

リアルタイムで安全、便利なサービスの提供.

- **オンラインの取引**

注文，支払い，配送状況問い合わせ

- **オンラインの銀行**

送金，残高照会，融資申請

- **オンラインの予約**

列車や飛行機などの座席予約

データベースの利用分野
③人工知能

人工知能での学習による上達：データを使用し，**学習**を通じて**知的能力を向上.**

- **ChatGPT などの対話型AI**

  （対話，自由なアイデア出し，

  要約，翻訳など）

- **医用画像や自動運転での画像理解**

  （画像診断、物体認識など）

- **オンラインショッピングでの情報推薦**

  （過去の履歴からの商品の順位付けなど）

データベースの利用分野
④データによる分析，予測

正確な予測，効果的な意思決定.

- **気象予報**
  気温，風速，風向き，湿度、降水などの過去データから天候，台風の進路，気温の変化などを予測
- **市場調査**
  販売，顧客からの問い合わせなどの過去データから，製品の需要などを予測
- **ヘルスケア**
  データに基づき病気の傾向，副作用の可能性などを推定

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

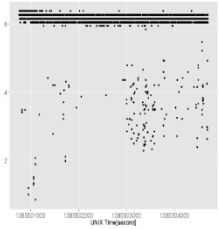
実世界

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

ここまでのまとめ

- **データベース**は、特定の主題について**整理，保存，管理**された**データ**の集合体

- **データベース**は、日常生活の情報管理に不可欠

銀行口座、ホテルの予約、大学の登録情報など、さまざまな情報がデータベース化

- **データベース**により、我々の生活はより豊かで便利に

オンラインコミュニケーション、リアルタイムのサービス提供、人工知能の学習と予測の向上、サイバーフィジカルシステム（現実世界とデジタル情報世界の融合）による新サービスの創出など、多方面に

7-2 SQL プログラムの例

Python と SQL の組み合わせ

- SQL:

リレーショナルデータベース管理システムに特化。

データベースの問い合わせや更新などが容易になる。

- **Python**

データ処理や**解析**、**可視化**など様々な機能。

処理全体の自動化も容易になる。

他のシステムとの連携も容易になる。

Python からの SQLite 3 の利用

SQLite 3 は**リレーショナルデータベース管理システム**。一部の機能に制限（ユーザ登録の機能がないなど）。その分、利用開始が容易。

<u>Python から SQLite3 を利用する方法</u>

[データベースへの接続を開く]

[カーソルの作成]

[SQLの実行（カーソルを利用）]

[SQLで取得されたデータをPythonで処理]

[データベースへの接続を閉じる]

この流れに沿ってプログラムを作成することで、
Python から SQLite 3 を利用することが可能

Python からの SQLite 3 の利用（Google Colaboratory）

```
import sqlite3

conn = sqlite3.connect('hoge.db')
```

[データベースへの接続を開く]

```
cursor = conn.cursor()
```

[カーソルの作成]

```
cursor.execute('CREATE TABLE students (id INTEGER PRIMARY KEY, name TEXT, age INTEGER)')
cursor.execute('INSERT INTO students VALUES (1, "tokugawa ieyasu", 20)')
cursor.execute('INSERT INTO students VALUES (2, "oda nobunaga", 22)')
cursor.execute('SELECT * FROM students')
```

[SQLの実行（カーソルを利用）]

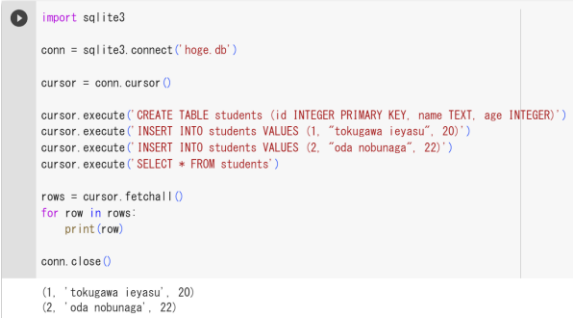
```
rows = cursor.fetchall()
for row in rows:
    print(row)
```

[SQLで取得されたデータをPythonで処理]

```
conn.close()
```

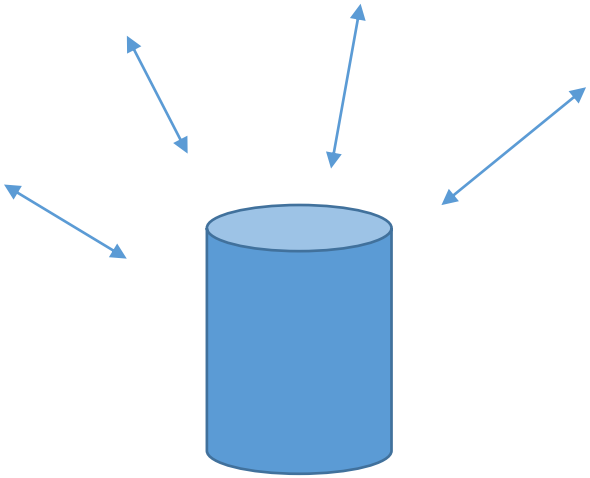
[データベースへの接続を閉じる]

コード



7-3. JSON、JSON とデータベースの連携

データベースとデータ送受信



データベースシステム
・情報の保管、検索

データ送受信
・システムから別のシステムへデータを伝達

JSON

JSON (Java Script Object Notation) は、**キー**と**バリュー**の**ペア**のデータを表現する。**データ送受信**に便利。

- データ：　“キー”, “値“ のように書く
- データ全体を { } で囲む

```
{"name": "apple", "price": 10}
```

{**キー**:　**値**,　　**キー**: **値**}

JSON の例

```
>>> import json
>>> x = {"name": "apple", "price": 10}
>>> print(json.dumps(x))
{"name": "apple", "price": 10}
>>>
```

Pythonで扱うことも簡単

JSON の用途

JSON は、**データの送受信**（データ交換）のざまざまな場面で活用

- Web アプリケーションでの、サーバとのデータ送受信
- 設定ファイルなど、構造化されたデータをファイルとして保存する手段
- 配列、オブジェクトなどのデータを構造化する手段

JSON で構造化されたデータの例

```
{
   "student": "tokugawa",
   "scores": [
      {
         "subject": "Math",
         "score": 90
      },
      {
         "subject": "Science",
         "score": 95
      }
   ]
}
```

JSON での入れ子

JSON では**入れ子**（値が JSON オブジェクトになること）が**可能**。

{"**name**":{"**0**":"apple","**1**":"orange"},"**price**":{"**0**":10,"**1**":20}}

入れ子の例

	name	price
0	apple	10
1	orange	20

```
In[17]: import pandas as pd
In[18]: x = pd.DataFrame({"name": ["apple", "orange"], "price": [10, 20]})
In[19]: print(x.to_json())
{"name":{"0":"apple","1":"orange"},"price":{"0":10,"1":20}}

In[20]:
```

Pythonで扱うことも簡単

文字列の中身が JSON 形式（Google Colaboratory）

```
json_data = '{"name":{"0":"apple", "1":"orange"},"price":{"0":10,"1":20}}'

print(json_data)
```

[json_data は文字列。中身は JSON]

[確認表示]

```
json_data = '{"name":{"0":"apple", "1":"orange"},"price":{"0":10,"1":20}}'
print(json_data)
```

```
{"name":{"0":"apple",                "1":"orange"},"price":{"0":10,"1":20}}
```

Python の辞書

- **Pythonの辞書**は、キーとバリューのペアを保持するデータ構造

```
d = {}
d["grape"] = 40  # 新たな項目の追加
d["apple"] = 15  # 既存の項目の更新
print(d["apple"])
print(d["grape"])
```

```
15
40
```

- Pythonの辞書と JSON の違い

	Python の辞書	JSON
用途	**Pythonプログラム内でデータを保持**	**データの送受信**
形式	全体で１つのオブジェクト	全体で文字列
バリュー	任意のオブジェクトをバリューにできる	バリューにできるものは限定されている

書き方の細かな違いもある

JSON データを解析してキーとバリューを格納した辞書を得る（Google Colaboratory）

```
import json
json_data = '{"name":{"0":"apple","1":"orange"},"price":{"0":10,"1":20}}'
```

[json_data は文字列。中身は JSON]

```
data = json.loads(json_data)
```

[JSON データの解析]

```
print(data)
```

[確認表示]

```
print(data['name'])
```

[キーが name のバリュー]

```
print(data['price'])
```

[キーが price のバリュー]

```
import json
json_data = '{"name":{"0":"apple","1":"orange"},"price":{"0":10,"1":20}}'

data = json.loads(json_data)
print(data)
print(data['name'])
print(data['price'])
```

```
{'name': {'0': 'apple', '1': 'orange'}, 'price': {'0': 10, '1': 20}}
{'0': 'apple', '1': 'orange'}
{'0': 10, '1': 20}
```

得られたバリューを整形して表示（Google Colaboratory）

```
import json
json_data = '{"name":{"0":"apple","1":"orange"},"price":{"0":10,"1":20}}'
```

[json_data は文字列。中身は JSON]

```
data = json.loads(json_data)
```

[JSON データの解析]

```
names = data['name']
```

[キーが name のバリュー]

```
prices = data['price']
```

[キーが price のバリュー]

```
for key in names:
    print(f"Item {key}:")
    print(f"  Name: {names[key]}")
    print(f"  Price: {prices[key]}")
```

[それぞれのキーについて、キーと name と price を表示

```
Item 0:
  Name: apple
  Price: 10
Item 1:
  Name: orange
  Price: 20
```

JSON のデータを SQLite 3 に格納（Google Colaboratory）

```
import json
json_data = '{"name":{"0":"apple","1":"orange"},"price":{"0":10,"1":20}}'

data = json.loads(json_data)

import sqlite3
conn = sqlite3.connect('hoge.db')
c = conn.cursor()
c.execute("CREATE TABLE T (name TEXT, price REAL)")
for i in range(len(data['name'])):
    c.execute("INSERT INTO T VALUES (?, ?)", (data['name'][str(i)], data['price'][str(i)]))
conn.commit()

c.execute("SELECT * FROM T")
rows = c.fetchall()
for row in rows:
    print(row)
conn.close()
```

[データベースへの接続を開く]

[カーソルの作成]

[SQLの実行（カーソルを利用）]

[SQLの実行（カーソルを利用）]

[SQLで取得されたデータをPythonで処理]

[データベースへの接続を閉じる]

実行結果

```
('apple', 10.0)
('orange', 20.0)
```

JSON のデータ型とその例

- 配列　　例 [1, 2, 3]
- 数値　　例 4.56
- 文字列　　例 "hello"　※　「"」で囲む
- ブール値　　true　false
- null値（存在しないことを示す）

```
{"name": "apple", "price": 10}
```

- **キー**　**必ず文字列である**
- **値**　さまざまな**データ型**を取ることができ、**JSON オブジェクト**を値とすることもできる

JSON とデータベースの連携

- **データベース**は、特定の主題について**整理，保存，管理**された**データ**の集合体
- **JSON** は、**柔軟なデータ構造**をもちつつ、**人間が読みやすい**形式である
- データベースから取り出したデータを JSON 形式で出力したり、JSON形式のデータをデータベースに格納したりすることが可能（**JSONとデータベースの連携**）
- JSONとデータベースの連携は、データ活用の柔軟性と効率性の向上させる

これらの技術を**理解**し、**実践する能力を磨くこと**は、**将来的に大きな利点になる**

7-4 データベースの種類とNoSQL データベース

データベースシステムの種類①

○ **リレーショナルデータベースシステム　SQLを使用**

	路線コード番号	事業者コード番号	路線名称一般	路線名称一般カナ
1	1001	3	中央新幹線	チュウオウシンカンセン
2	1002	3	東海道新幹線	トウカイドウシンカンセン
3	1003	4	山陽新幹線	サンヨウシンカンセン
4	1004	2	東北新幹線	トウホクシンカンセン

テーブル

データベースシステムの種類②

○ SQL を使用しない NoSQL データベース

NoSQL データベースは、伝統的なリレーショナルデータベースとは違い、**スキーマ（テーブル定義）を持たない。大量データを効率よく処理する**ことを目指している。

① **ドキュメント指向**の**データベース**

JSON, XMLなどの**ドキュメント**に特化.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap
0.0.2">
 <bounds minlat="54.0889580"
minlon="12.2487570" maxlat="54.0913900
maxlon="12.2524800"/>
 <node id="298884269" lat="54.0901746"
lon="12.2482632" user="SvenHRO"
uid="46882" visible="true" version="1"
changeset="676636" timestamp="2008-09-
```

② **キー・バリュー形式**の**データベース**

キー（鍵）	バリュー（値）
45343430	金子邦彦

データの形は、
キー　＋　バリュー

③ **グラフ（ノードとエッジ）形式**の**データベース**

④ **カラム指向**の**データベース**

TinyDB

- Tiny DB は、Python で実現された、ドキュメント指向のデータベースシステム．**NoSQL**の一種。

Python からの TinyDB の利用（Google Colaboratory）

```
!pip install tinydb

from tinydb import TinyDB, Query

db = TinyDB('my.json')

db.insert({'type': 'apple', 'count': 7})
db.insert({'type': 'peach', 'count': 3})

Fruit = Query()
db.search(Fruit.type == 'peach')
```

[インストール]

[インポート]

[データベースの初期化]

[データの挿入]

[データの検索]

コード

```
!pip install tinydb

from tinydb import TinyDB, Query

db = TinyDB('my.json')

db.insert({'type': 'apple', 'count': 7})
db.insert({'type': 'peach', 'count': 3})

Fruit = Query()
db.search(Fruit.type == 'peach')
```

```
Looking in indexes: https://pypi.org/simple, https://
Requirement already satisfied: tinydb in /usr/local/l
[{'type': 'peach', 'count': 3}]
```

	リレーショナルデータベース	NoSQLデータベース
データベースの構成物	テーブル	コレクション、オブジェクト、キーとバリューのペア、ノードなど
主キーの機能	有り	有り
二次索引の機能	有り	有り（一部のデータベースでは制限あり）
ロックの単位	データベース全体，テーブル，レコード（行）	オブジェクト、ドキュメント、レコード、キーとバリューのペア
基礎概念	正規形，トランザクション，リレーショナル代数とSQL	非正規形、スキーマレス

まとめ

- **リレーショナルデータベース（RDB）**
  - テーブル構造を持つデータベース
  - 各テーブルは、主キーを使ってデータを一意に識別
  - 二次索引を使用してデータの検索性能を向上
  - ロック機構による並行処理。データ整合性を保証
- **NoSQLデータベース**
  - RDBのスケーラビリティと柔軟性に対する問題の解決を目指す
  - ドキュメント型、キーバリュー型、カラム型、グラフ型などさまざまな種類
  - 主キーは存在することが多いが、必須ではない場合もある
  - 二次索引の利用は可能だが、制限がある場合がある
  - ロックは、行わない場合がある

全体まとめ

1. **データベース**は**特定の主題に関するデータを整理、保存、管理する**もの。
2. **SQL**と**Python**との組み合わせにより、**データの問い合わせや更新**、やデータ処理が柔軟になる。
3. **JSON**は**キーと値のペアでデータを表現**。データの送受信に便利。
4. データベースから取り出したデータをJSON形式で出力したり、JSON形式のデータをデータベースに格納したり（**JSONとデータベースの連携**）することが可能。
5. リレーショナルデータベースとNoSQLデータベースの2種類
6. **リレーショナルデータベース**は**テーブル構造**を持ち、**SQL**を使用。
7. **NoSQLデータベース**は、大量のデータを効率よく処理することを目指している。**ドキュメント指向**、**キー・バリュー形式**、**グラフ形式**、**カラム指向**のデータベースなどがある。