

データベース

《学修項目》

- ◎テーブル定義、ER図
- ◎主キーと外部キー
- ◎リレーショナルデータベース（RDB）
- データ操作言語（DML）、SQL

《キーワード》

テーブル定義, 実体-関連モデル, 関係モデル, スキーマ, 関係従属性とキー, リレーショナルデータベース (RDB), SQL, RDBに対する演算

《参考文献, 参考書籍》

- [1] [東京大学MIセンター公開教材「2-4 データベース」](#) 《利用条件CC BY-NC-SA》
- [2] [データサイエンスのためのデータベース（データサイエンス入門シリーズ）](#)（講談社）
- [3] [数理・データサイエンス・AI公開講座](#)（放送大学）

1. データベースとテーブル定義

あらまし

- データベースの重要性
- テーブル定義
- 実体-関連モデル (ER図)

1.1 データベースの重要性

データベース (database, DB) とは

- 組織化されたデータの集まりを意味します
 - この「組織化」とは特定の操作のために準備されていることを意味します
 - 操作の典型例：検索，追加，更新，削除
- データベースの例
 - ネットショッピング
 - 銀行口座
 - ルート検索
 - 学務管理
 - デジタルライブラリ
- オンラインシステムは基本的に何らかのデータベースを伴います

データ管理基盤としてのDB

例えば、[Twitterは1分間に 57万メッセージものトラフィックがあり膨大である](#)。Webサービス・インフラのバックエンドとして、データベース基盤は根幹をなしている。

(引用：[Data Never Sleeps 9.0:](#))

大規模データの管理・運用例

- 大学の学生，職員情報の管理システム
- 病院の患者のカルテ管理システム
- 遺伝子データの管理システム
- 判例を管理するシステム
- 論文を管理するシステム
- 商品の注文を行うシステム

1.2 テーブル定義

どうやってデータベースを設計するのか

- 既存のデータベースを操作することと新しくデータベースを作成することは根本的に別の行為です
 - 既存のデータベースの構造を知っていれば、それへの検索・追加・更新・削除のために、どのようなSQLクエリを書けばよいかは分かります
 - 新しくデータベースを作る際には、どのような構造にするかを事前に決める必要があります
- 対象（何らかの概念や状況）をデータベースの構造で表現することをデータモデリングと呼びます
 - データモデリング自体はデータベースに限定されません
 - プログラミングで所望の計算を関数で表現することと似ています
- RDB向けのデータモデリングとしてERモデリングがあります
 - ERモデルの作成を通してRDBを設計することです

設計例) 旅行予約サイトにおけるDBの果たす役割 (列車, 飛行機, 宿泊先の予約, ユーザ情報の登録)

- [OREND | ホテル・宿泊予約管理システム比較](#)

関係データベースの例 (各地区ごとの気象観測データ)

- 都道府県番号, 年月日, 天気, 気温, 湿度, 風向きなどのフィールドが用意
- [気象庁 | 過去の気象データ・ダウンロード](#)
- [長野県 > 長野市 > 気温, 降水, 積雪, 日射, 風, 雲の量, 天気 > 2021年1月1日から12月31日まで](#) を条件として問い合わせ (クエリ) し, CSVデータとしてダウンロードした後, 必要なカラムだけ残してUTF-8へ文字コード変換したもの (on GitHub)

```
In [ ]: # CSVデータを カレントディレクトリ直下のフォルダ(一時作業領域)へダウンロードする.
!wget -nc https://raw.githubusercontent.com/MDASH-shinshu/MDASH-T-DE/main/4/

# wgetしなくても,Google colab.の左メニュー [ファイル] アイコンをクリックして,ブラウザへファイルをアップロード

# ファイル (JMA_Nagano_2021.csv) がダウンロード・配置できたことを確認する
!ls -al ./
```

```
In [2]: # CSVファイルをpandasで読み込んでデータフレーム df を表示(簡単データベース)
import pandas as pd
from IPython.display import display




df = pd.read_csv('JMA_Nagano_2021.csv')
display(df)
```

	年月日	平均 気温 (°C)	最高 気温 (°C)	最低 気温 (°C)	降水 量の 合計 (mm)	日 照 時 間 (時 間)	平均 風速 (m/s)	最大 風速 (m/s)	風 向	最大 瞬間 風速 (m/s)	風 向.1	最 多 風 向 (16 方 位)	平均 蒸気 圧 (hPa)	最 小 相 対 湿 度 (%)
0	2021/1/1	-1.9	0.9	-3.5	5.5	1.4	1.2	4.9	北東	6.9	北東	東北東	4.9	77
1	2021/1/2	-2.3	1.3	-4.5	0.0	0.1	1.7	6.8	北	10.6	北	東北東	4.6	73
2	2021/1/3	-2.5	0.7	-5.5	0.0	1.6	1.3	3.6	東	4.9	東北東	東	4.6	79
3	2021/1/4	-1.4	3.5	-5.3	0.0	4.4	1.1	3.2	東北東	4.3	東北東	東	4.9	72
4	2021/1/5	-0.6	2.7	-3.0	0.0	1.5	1.0	2.7	西南西	4.0	西南西	東	5.3	76
...
360	2021/12/27	-3.3	-0.3	-6.2	0.0	1.6	1.1	3.8	東	5.5	東北東	東北東	4.1	61
361	2021/12/28	-2.0	1.7	-5.3	2.0	5.9	2.3	6.0	東北東	8.2	東北東	東北東	4.4	64
362	2021/12/29	-1.3	5.8	-8.0	0.0	8.2	1.3	3.5	南南西	6.0	南南西	南南西	4.2	47
363	2021/12/30	0.0	2.0	-1.9	7.0	0.2	1.4	4.0	東	6.6	東	東	5.9	86
364	2021/12/31	-3.3	-0.5	-5.3	4.5	0.0	2.5	5.1	東	7.6	北東	東北東	4.4	59

365 rows × 17 columns

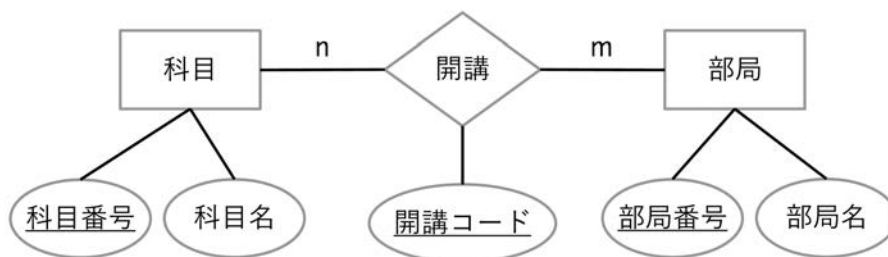
1.3 実体-関連モデル (ER図)

ERモデルの構成要素と構造

- 実体：
- 関連：
- 属性：
- 各要素は、複数のインスタンス（実例）から成る集合です
- 要素を線で繋ぐことでモデルの構造を記述します
- 実体と関連を繋ぐ線には濃度制約を記述します
 - 1を書くとき実体のインスタンスがただ1つしか関連しないことを表します
 - nなどの変数を書くとき、不特定のインスタンスが関連することを表します
 - 2重線で繋ぐとき実体のインスタンス全体が関連することを表します
- 属性は、線で繋がった対象のインスタンス全てが持ちます
 - 主キーには下線を引きます

ERモデル（実体関連モデル）

- 実体（entity）と実体間の関連（relationship）に基づくモデルです
- ERモデルはダイアグラムの形で記述されます
 - ここで紹介する記法は、ERモデルを提案したChenのスタイルに準じます
- 例：「開講」のERモデル
 - 「開講」を「科目」と「部局」という実体間の関連としてモデル化



実体-関連モデル (ER)

- 実体は [四角], 属性は (楕円) で表現する. 実体集合間の関連 は <ひし形> で表現する

[科目]: (科目番号), (科目名)
[部局]: (部局番号), (部局名)
<開講>: (開講コード), [科目], [部局],

ERモデリングの考え方と使い方

- ERモデルは対象を説明する道具です
- 同一対象に対するERモデルは様々考えられます
 - 最も良く説明していると思えるERモデルを作ることが肝要です
- ERモデルに対応するRDBも1つに定まる訳ではありません
 - ERモデルはRDBに繋げるための土台でしかありません
- 例えば、次のようにすれば、ERモデルからRDBを構成できます
 - それぞれの実体を1つの表にする
 - 一対一の関連で繋がる実体は、表を外部キーで参照する
 - 一対多や多対多の関連は、外部キーで双方を参照する表にする
 - インスタンス（外部キーの組）を番号付けして主キーとする
- 実際には、SQLによるデータ操作が効率良くなるように、RDBの構造を工夫することが良くあります
 - 例1：一対一関連で繋がる複数実体を1つの表にまとめる
 - 例2：多対多関連を一対多関連に分解して複数表にする

ERモデルから関係データベースへの変換ルール

- 実体集合：属性をそのまま列にする
- 関連：2つの実体の主キーと関連の属性を列にする

→ 変換後は、複数の実体集合をデータ項目（親テーブル）として、外部参照しながら関連性を表したもの（子テーブル）を作成することとなる。

2. 主キーと外部キー

あらまし

- 関係モデル
- スキーマ
- 関係モデルの定式化
- 関係従属性とキー
- 関数従属性
- 主キー, 外部キー, 参照制約

2.1 関係モデルとしてのリレーショナルデータベース(RDB)

リレーショナルデータベース (RDB)

- 関係 (リレーション) データに基づくデータベースです
 - 1970年にEdgar F. Coddによって理論的基盤が提唱されました
- 関係データ = 表
 - 行 (レコード, タプル) と列 (属性) の構造を持ちます
 - 行が1単位のデータを表し, 列が1種類のデータを表します
- 表の例

氏名	所属	Email
東大太郎	理学部	taro@s.u-tokyo.ac.jp
東大花子	薬学部	hanako@f.u-tokyo.ac.jp
東大史郎	法学部	shiro@j.u-tokyo.ac.jp

- 横方向のひとまとまり (赤) がレコードです
- 縦方向のひとまとまり (青) が属性です
 - 氏名・所属・Emailはデータの種類を表す属性名です

関係モデル

データを関係で表現するモデル. RDBとは関係表の集合と言える

- データテーブル: 実体
- データ項目: 属性集合 (スキーマ)
- フィールド名: 属性 (カラム)
- 各データレコード: タプル (行)

→ 上記全てをあわせて 関係 と呼ぶ

2.2 スキーマ

表の名前と属性の集合 <表の名前> (<属性1>, ...) のように表現する

例) 学生 (学生番号, 名前, 入学年)
科目 (科目番号, 科目名, 中間試験の有無)

2.3 関係モデルの定式化の例

- 属性: 科目 (科目番号, 科目名, 中間試験の有無)
- 属性の定義域: 科目番号={C001,C002,...}, 科目名={データ管理, 情報基礎, ...}, 中間試験の有無={有,無}
- 関係 $R \subseteq \text{科目番号} \times \text{科目名} \times \text{中間試験の有無}$ (直積集合)

→ 各データレコードは, 関係R の 元となる.

2.4 関係従属性とキー

関係内のルールのようなもの

- 関係従属性の例：「学生番号が決まれば、名前も一意に決まる」
- キーの例：「学生番号が決まれば、1人の学生を特定できる」

誰が関係従属性を決めるのか？

- データの性質によって決めていく
- RDB設計者の裁量による

2.5 関数従属性の例

【定義】ある関係Rに、関係従属性 $X \rightarrow Y$ がある

⇔関係Rにおいて、属性Xの値が属性Yの値を一意に決定する

つまり、任意の2つのタプルの属性Xの値が同じであれば、それらのタプルYの値も同じである。

例) 学生番号→名前, 科目番号→科目名が関数従属性を有する

2.6 主キー, 外部キー, 参照制約 【重要】

主キー：データベースの設計者が属性集合の中から1つ選んだキーのこと

- スキーマの属性に下線を引くことで、主キーを表す

例) 学生 (学生番号, 名前, 入学年)

科目 (科目番号, 科目名, 中間試験の有無)

※主キーで、1つのタプル（データレコード）を指定することが多い

外部キー：関係表の表のある列に、別の関係表の特定の列に登録されている項目した入力できないように指定する属性のこと

参照制約：外部キーを設定することで、参照先の関係表の指定された列に、登録された値しか書き込むことができない制約のこと（複数の実体集合をデータ項目とした関係（子テーブル）で用いられる）

主キー (primary key)

- 主キーとは表中のレコードを一意に特定できる属性を意味します
 - 例えば、前頁の「入学」表における「学籍番号」が主キーです
 - 主キー以外で一意に特定可能な属性（の集合）を代理キーと呼びます
- 主キーは表に必須ではないですが、ないと不便なことが多いです
 - 「入学」表に氏名と入学時期が同じ学生を入れられなくなります
- しばしば機械的に割り振られたID番号が用いられます
 - 例：マイナンバー
- ある表の主キーの値はその表との関連を表現できます
 - 主キーの値さえあれば他の属性の値も参照可能です
- 別の表の主キーを参照する属性を外部キー (foreign key) と呼びます
 - 外部キーの値には参照先の主キーの値しか許容されません

複数の表からなるデータベース

- 複雑な関係データも複数の表に分解すると見通しが良くなります
- 例：「開講」表

開講コード	科目	開講先
0301	1001	03
0302	1002	03
0501	1001	05

複数の科目を複数の部局で開講する多対多の関係

外部キー

科目番号	科目名
1001	統計学
1002	データベース

外部キー

部局番号	部局名
03	理学部
05	薬学部
07	法学部

3. リレーショナルデータベース (RDB) と SQL による操作

あらまし

- SQL (Structured Query Language)
- クエリ (Query: 検索)
- 関係データベースに対する演算

3.1 リレーショナルデータベース (RDB) とは

リレーショナルデータベース (RDB)

- 関係（リレーション）データに基づくデータベースです
 - 1970年にEdgar F. Coddによって理論的基盤が提唱されました
- 関係データ = 表
 - 行（レコード、タプル）と列（属性）の構造を持ちます
 - 行が1単位 of データを表し、列が1種類のデータを表します
- 表の例

氏名	所属	Email
東大太郎	理学部	taro@s.u-tokyo.ac.jp
東大花子	薬学部	hanako@f.u-tokyo.ac.jp
東大史郎	法学部	shiro@j.u-tokyo.ac.jp

- 横方向のひとまとまり（赤）がレコードです
- 縦方向のひとまとまり（青）が属性です
 - 氏名・所属・Emailはデータの種類を表す属性名です

3.2 データベース管理言語 (DML)

データベース言語とは

- データベースの機能を提供するプログラミング言語です
 - つまり、データベースはデータベース言語によって組織化されます
- データベース言語の3つの側面
 - データ操作言語：操作（検索、追加、更新、削除等）を記述する言語
 - データ定義言語：データの構成を記述する言語
 - データ制御言語：データのアクセス制御を記述する言語
- RDB向けのデータベース言語の標準としてSQLがあります
 - Structured Query Languageに由来します
 - 「エスキューエル」もしくは「シーケル」と発音します
 - SQLは前述の3つの側面を全て併せ持つ言語です
- 以降、RDBとSQLを通してデータベースの基本を説明します

3.3 SQL (Structured Query Language)

関係データベース言語

- スキーマ定義やデータ操作などを行うための言語
- ISO (国際標準機構) によって標準化

- 異なる関係データベース管理システム上で動作（プラットフォーム非依存、ミドルウェアと呼ぶ）
- 関係代数の演算（制限、射影、結合、直積）を記述する

クエリ（Query:検索）

- 検索結果を得るための問い合わせ
- リレーショナルデータベースでは、データの登録、検索、削除、更新をSQLによって行うことができる。

3.4 関係データベースに対する演算

- 制限：いくつかのタプル（行）の抽出
- 射影：いくつかの属性（カラム）の抽出
- 結合：ある属性で関係AとBを結びつける
- 直積：関係AとBの組み合わせの「全てのパターン」を得る

3.4.1 制限

ある条件に該当するタプル（行）の取得 表計算ソフトの場合、必要が行だけを残す（選択）することに相当する

例）家賃がX万円以上、Y万円以下のアパートを探したい
駅から徒歩T分以内のホテルを探したい

など

```
# SQLによる操作例
SELECT * FROM purchase_history
WHERE ID = '001001'
```

3.4.2 射影

ある属性（カラム）だけを取得 表計算ソフトの場合、必要が列だけを残す（選択）することに相当する

例）賃貸DBから、家賃と駅からの距離の2カラムだけに絞りたい
カメラの購入時比較のため、名前、値段、画素数だけに絞りたい など

```
# SQLによる操作例
SELECT ID, Name, Item
FROM purchase_history
```

3.4.3 直積

複数の関係（テーブル）を単純に直積集合で求める

例）顧客テーブル、レンタル履歴テーブル、レンタル商品を結合して、ある人が過去に借りた商品を（横断的に）調べたい

※表計算ソフトでは非常に高コストな操作となるが、RDBであればviewという形で実現できる

```
# SQLによる操作例
SELECT ID, faculties.Name, Age, courses.Name
FROM faculties, courses
```

3.4.4 結合

複数の関係（テーブル）の直積集合のviewから、必要なレコードだけを抽出する 結合 := 直積 >> 制限 とも言える

```
# SQLによる操作例
SELECT customers.ID, Name, Age, Item
FROM customers, purchase_history
WHERE customers.ID = purchase_history.ID
```

3.5 SQL言語によるデータベース操作の実例

3.5.1 気象データベースからの検索例 (Python SQLite3)

```
In [3]: # CSVファイルをpandasで読み込んでデータフレームdfに格納
import sys
import pandas as pd
from IPython.display import display
import sqlite3

df = pd.read_csv('JMA_Nagano_2021.csv')
df.columns = ['date', 'temp_ave', 'temp_high', 'temp_low', 'rain_total', 'sun_shi

##display(df)
```

```
In [4]: # データベースを新規作成して、接続する
dbname = ('weather.db') #データベース名.db拡張子で設定(カレントディレクトリに通常のファイルと
conn = sqlite3.connect(dbname, isolation_level=None) #データベースを作成、自動コミッ
cursor = conn.cursor() #カーソルオブジェクトを作成
```

```
In [5]: # nagano2021テーブル：年月日,平均気温(℃),最高気温(℃),最低気温(℃),降水量の合計(mm),
sql = """CREATE TABLE IF NOT EXISTS nagano2021(date,temp_ave,temp_high,temp_

cursor.execute(sql) #executeコマンドでSQL文を実行
conn.commit() #データベースにコミット
cursor = conn.cursor() #カーソルオブジェクトを作成
```

```
In [20]: # pandas DataFrameから nagano2021テーブルに全てインポートする
df.to_sql('nagano2021',conn,if_exists='append',index=None)
conn.commit() #データベースにコミット
```

```
In [25]: ## クエリ例1) nagano2021テーブルの temp_ave が 29度から35度の間であった日を抽出する
sql = """SELECT * FROM nagano2021 WHERE temp_ave BETWEEN 29 and 35;"""
cursor.execute(sql)
print(cursor.fetchall())#全レコードを取り出す
```

```
[('2021/8/3', 30.1, 35.5, 26.5, 0.0, 8.2, 4.5, 6.7, '西南西', 11.2, '南西', '西南西', 25.3, 41, 60, '晴時々曇', '晴時々曇'), ('2021/8/4', 29.0, 35.2, 24.9, 0.0, 12.0, 2.9, 6.8, '北', 11.5, '北', '北', 26.8, 49, 67, '晴、雷を伴う', '晴時々曇'), ('2021/8/6', 29.8, 37.3, 23.8, 0.0, 12.2, 3.5, 6.7, '東', 9.8, '南西', '西南西', 22.4, 26, 56, '晴', '曇時々晴一時雨'), ('2021/8/7', 29.3, 35.1, 25.8, 0.0, 6.2, 3.5, 6.4, '南西', 11.7, '南西', '西南西', 25.1, 43, 62, '晴時々曇', '曇')]
```

```
In [8]: ## クエリ例2) nagano2021テーブルの 天気概況(日中) forecast_day に"みぞれ" が含まれる日
sql = """SELECT * FROM nagano2021 WHERE forecast_day LIKE '%みぞれ%';"""
cursor.execute(sql)
print(cursor.fetchall())#全レコードを取り出す
```

```
[('2021/1/14', 1.4, 9.0, -3.9, 0.0, 8.2, 2.4, 8.5, '北東', 12.9, '北東', '北東', 5.6, 55, 83, '晴後一時雨、みぞれを伴う', '晴時々曇'), ('2021/1/16', 0.6, 3.8, -1.9, 11.0, 1.3, 2.5, 6.0, '東北東', 9.1, '東北東', '東北東', 5.8, 71, 91, 'みぞれ時々曇一時雪', '曇時々晴一時雪、みぞれを伴う'), ('2021/1/22', 1.4, 6.3, -3.8, 0.0, 3.5, 0.9, 2.2, '南西', 4.0, '西南西', '東南東', 5.9, 56, 87, '曇時々晴後一時雨、みぞれを伴う', '曇時々晴'), ('2021/1/23', 2.7, 4.4, 0.8, 3.5, 0.0, 1.5, 3.8, '東南東', 5.5, '東南東', '東北東', 6.5, 68, 88, '曇後時々みぞれ', 'みぞれ後時々雨'), ('2021/1/27', 5.2, 9.6, 2.2, 1.0, 2.6, 3.6, 9.5, '北', 15.4, '北北東', '北北東', 7.2, 65, 81, '曇後一時雨、みぞれを伴う', '曇'), ('2021/1/31', -0.8, 2.6, -4.1, 0.5, 3.0, 3.2, 7.9, '東', 12.4, '東', '東北東', 5.1, 65, 88, '雪時々曇、みぞれを伴う', '晴'), ('2021/2/2', 2.9, 7.7, -1.7, 6.5, 1.0, 3.2, 9.6, '東', 15.0, '東', '東北東', 6.8, 72, 89, '曇時々雨、みぞれを伴う', '晴後曇一時雪'), ('2021/2/4', -0.6, 3.5, -4.4, 0.0, 0.6, 1.8, 6.3, '西南西', 9.6, '南', '北東', 4.8, 47, 82, '曇後時々みぞれ一時雪', '曇一時雪'), ('2021/2/7', 2.2, 10.5, -2.2, 0.0, 4.8, 2.2, 6.5, '北', 10.2, '西', '東北東', 5.9, 49, 84, '晴後時々雨、みぞれを伴う', 'みぞれ後曇時々雪'), ('2021/2/11', 1.5, 6.1, -1.4, 3.5, 1.4, 2.3, 8.4, '北東', 12.6, '北東', '東北東', 5.8, 61, 86, '曇一時雪後時々晴、みぞれを伴う', '曇一時晴'), ('2021/2/16', 2.3, 6.9, -1.1, 2.5, 2.7, 3.0, 9.9, '西', 17.0, '西', '西北西', 5.0, 29, 71, '曇時々晴、みぞれを伴う', '雪時々曇'), ('2021/2/23', 2.5, 8.3, -2.0, 4.5, 4.6, 4.3, 8.6, '北東', 12.7, '北東', '東北東', 5.9, 54, 80, '曇一時雨後時々晴、みぞれを伴う', '晴時々雪'), ('2021/3/2', 8.1, 18.5, 0.2, 21.0, 0.0, 5.2, 8.2, '東北東', 13.4, '東北東', '西南西', 7.8, 46, 74, '曇後時々雨、みぞれを伴う', '雪時々曇一時晴'), ('2021/3/6', 4.7, 9.3, 0.4, 0.0, 0.2, 3.7, 9.3, '北北西', 14.3, '北北西', '北東', 7.3, 64, 84, '曇一時雨、みぞれを伴う', '曇、みぞれを伴う'), ('2021/4/9', 4.1, 10.5, 0.5, 0.0, 7.5, 3.6, 8.9, '東', 15.0, '北', '北北東', 5.5, 36, 68, '晴後一時みぞれ', '晴時々曇、みぞれを伴う'), ('2021/11/27', 2.0, 5.1, -0.1, 12.5, 2.2, 1.9, 6.6, '西', 11.1, '西', '東北東', 6.5, 67, 92, 'みぞれ時々雨一時雪、雷を伴う', 'みぞれ時々曇、雷・霧を伴う'), ('2021/12/4', 4.6, 9.7, 1.0, 0.0, 5.7, 2.2, 5.4, '東北東', 8.4, '東北東', '東北東', 6.5, 57, 77, '晴時々雨一時曇、みぞれを伴う', '曇時々晴一時雪'), ('2021/12/6', 3.3, 6.7, -2.1, 0.0, 0.0, 1.3, 2.9, '東', 5.0, '南南西', '北北東', 7.2, 79, 90, '雨時々曇一時雪、みぞれを伴う', '曇一時晴後雨'), ('2021/12/13', 2.8, 6.6, -0.4, 0.0, 7.6, 3.8, 8.1, '東', 13.1, '東北東', '東北東', 6.0, 66, 80, 'みぞれ一時雨後晴', '晴時々曇'), ('2021/12/22', 1.7, 4.3, -1.1, 0.0, 2.6, 3.0, 6.5, '北東', 10.0, '北', '北東', 5.8, 63, 84, 'みぞれ時々曇一時雪', '晴'), ('2021/12/25', 1.1, 4.2, -3.8, 0.0, 0.2, 3.1, 6.5, '東', 12.6, '北', '北東', 5.8, 68, 86, 'みぞれ一時雨後曇時々雪', '晴時々曇一時雪'), ('2021/12/30', 0.0, 2.0, -1.9, 7.0, 0.2, 1.4, 4.0, '東', 6.6, '東', '東', 5.9, 86, 96, 'みぞれ時々曇一時雪、雷を伴う', '曇時々雪一時みぞれ')]
```

```
In [9]: ## クエリ例3) nagano2021テーブルの 3月中で、天気概況 forecast_day に"雪" が含まれる日
sql = """SELECT * FROM nagano2021 WHERE date BETWEEN '2021/3/1' and '2021/3/31';"""
cursor.execute(sql)
print(cursor.fetchall())#全レコードを取り出す
```

```
[('2021/3/3', -0.1, 3.7, -2.3, 0.0, 5.6, 3.4, 7.9, '北', 13.6, '北', '北東', 4.7, 59, 78, '曇時々晴一時雪', '晴時々曇')]
```

3.5.2 ERモデルから関係データベースへの変換例 (Python SQLite3)

複数の表からなるデータベース

- 複雑な関係データも複数の表に分解すると見通しが良くなります
- 例：「開講」表

開講コード	科目	開講先
0301	1001	03
0302	1002	03
0501	1001	05

複数の科目を複数の部局で開講する多対多の関係

外部キー

科目番号	科目名
1001	統計学
1002	データベース

外部キー

部局番号	部局名
03	理学部
05	薬学部
07	法学部

データベース course, 親テーブル subject, department, 子テーブル registrationを作成して、ERモデルに基づいた関係データベースを作成してみる（外部キー参照による制約付き）

```
In [10]: ## ERモデルから関係データベースへの変換例 (SQLite3)
import sqlite3

# データベースを新規作成して、接続する
dbname = ('course.db') #データベース名.db拡張子で設定(カレントディレクトリに通常のファイルとして保存)
conn = sqlite3.connect(dbname, isolation_level=None) #データベースを作成、自動コミット
cursor = conn.cursor() #カーソルオブジェクトを作成

In [11]: # subject [科目]テーブルを作成する(主キー:id)
sql = """CREATE TABLE IF NOT EXISTS subject(id varchar(4) primary key, name
cursor.execute(sql) #executeコマンドでSQL文を実行
conn.commit() #データベースにコミット

In [12]: # 作成したテーブルにレコードを複数行格納する
sql = """INSERT INTO subject VALUES(?,?)"""

data = [
    ("1001", "統計学"),
    ("1002", "データベース")
]
cursor.executemany(sql, data) #SQL文を実行

Out[12]: <sqlite3.Cursor at 0x12ac41bc0>

In [13]: # department [部局]テーブルを作成する(主キー:id)
sql = """CREATE TABLE IF NOT EXISTS department(id varchar(2) primary key, name
cursor.execute(sql) #SQL文を実行

Out[13]: <sqlite3.Cursor at 0x12ac41bc0>
```



```
In [14]: # 作成したテーブルにレコードを複数行格納する
sql = """INSERT INTO department VALUES(?,?)"""

data = [
    ("03", "理学部"),
    ("05", "薬学部"),
    ("07", "法学部")
]
cursor.executemany(sql, data)#SQL文を実行
```

Out[14]: <sqlite3.Cursor at 0x12ac41bc0>

```
In [15]: sql = """SELECT * FROM subject"""
cursor.execute(sql)
print(cursor.fetchall())#全レコードを取り出す
sql = """SELECT * FROM department"""
cursor.execute(sql)
print(cursor.fetchall())#全レコードを取り出す

[('1001', '統計学'), ('1002', 'データベース')]
[('03', '理学部'), ('05', '薬学部'), ('07', '法学部')]
```

```
In [16]: # registration [開講科目]テーブルを作成する(外部キー:subject_id, department_id)
sql = """CREATE TABLE IF NOT EXISTS registration(id varchar(4) primary key,
cursor.execute(sql) #SQL文を実行

sql = """PRAGMA foreign_keys=true; """ # SQLite3 このテーブルについて外部キー参照を
cursor.execute(sql) #SQL文を実行
```

Out[16]: <sqlite3.Cursor at 0x12ac41bc0>

```
In [17]: # 作成したテーブルにレコードを複数行格納する
sql = """INSERT INTO registration VALUES(?,?,?)"""

data = [
    ###("0201", "1999", "03"), # 外部キー設定してあるので,存在しない科目番号を指定すると
    ("0301", "1001", "03"),
    ("0302", "1002", "03"),
    ("0501", "1001", "05")
]
cursor.executemany(sql, data)#SQL文を実行
```

Out[17]: <sqlite3.Cursor at 0x12ac41bc0>

```
In [18]: sql = """SELECT * FROM registration"""
cursor.execute(sql)
print(cursor.fetchall())#全レコードを取り出す

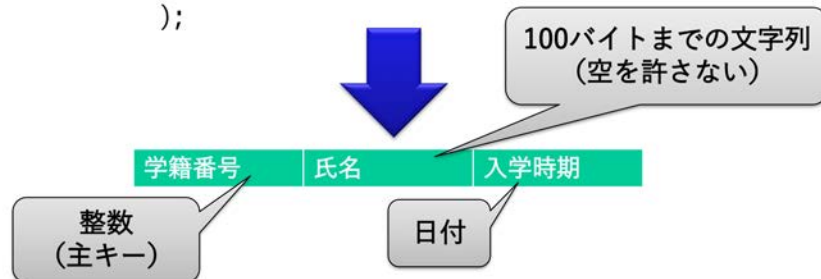
[('0301', '1001', '03'), ('0302', '1002', '03'), ('0501', '1001', '05')]
```

SQLの代表的なコマンド (リファレンス)

SQLによるデータ定義：CREATE

- 表を作成する操作です
- 例：空の「入学」表の作成

```
CREATE TABLE 入学 (  
  学籍番号 INTEGER PRIMARY KEY,  
  氏名 CHAR(100) not null,  
  入学時期 DATE  
);
```



SQLによるデータ操作：INSERT

- 表にデータを追加する操作です
- 例：「所在地」表

キャンパス	市区町村
本郷	文京

```
INSERT INTO 所在地  
VALUES ('駒場', '目黒');
```

キャンパス	市区町村
本郷	文京
駒場	目黒

SQLによるデータ操作：UPDATE

- 表の一部データを更新する操作です
- 例：「預金」表

預金者	預金額
東大太郎	100,000
東大史郎	1,000,000

```
UPDATE 預金  
SET 預金額 = 0  
WHERE 預金者 == '東大史郎';
```



預金者	預金額
東大太郎	100,000
東大史郎	0

SQLによるデータ操作：DELETE

- 条件を満たすレコードを削除する操作です
- 例：「預金」表

預金者	預金額
東大太郎	100,000
東大史郎	1,000,000

```
DELETE 預金  
WHERE 預金額 < 200000;
```



預金者	預金額
東大史郎	1,000,000

SQLによるデータ操作：SELECT

- 表から一部のデータを取り出す操作です
 - 条件を満たすレコードから指定された属性だけを返す
- 例：「名簿」表

氏名	所属	Email
東大太郎	理学部	taro@s.u-tokyo.ac.jp
東大花子	薬学部	hanako@f.u-tokyo.ac.jp
東大史郎	法学部	shiro@j.u-tokyo.ac.jp

```
SELECT 氏名,Email  
FROM 名簿  
WHERE 所属 == '理学部';
```



氏名	Email
東大太郎	taro@s.u-tokyo.ac.jp

3.6 RDB以外のデータベース

その他のデータベース

- RDBとは異なるデータモデルを採用するデータベースもあります
 - ドキュメント指向DB：ドキュメントの集合を保持するDB
 - 列指向DB：RDBと同様に表データだが、列を丸ごと操作するDB
 - XML DB：XMLの木構造を保持するDB
 - グラフDB：一般のグラフ構造を保持するDB
- RDBでないデータベースにはSQLでない言語を使うのが一般的です
 - データベースシステム毎に固有の言語であることも多いです
- 「RDB以外」を指す言葉としてNoSQLというものがあります
 - 字面からはSQLに対応する言語を指しているように見えますが、言語を指している訳ではありません
 - データベースシステムの分類として用いられます
- 複数の異なるデータベースを統合したものをデータウェアハウスと呼ぶことがあります
 - データウェアハウスもデータベースの一種と見做せます
 - 企業データを管理蓄積する文脈で使われる言葉です

memo