

DBモデリング基礎講座

- 2021年6月23日(水)
- GMOペパボ株式会社
- 五十嵐邦明 / igaiga
 - <https://twitter.com/igaiga555>

モデリング、DBモデリングとは

- モデリング
 - 現実世界をコンピュータの世界で扱えるようにするための設計
- DBモデリング
 - 現実世界をDBの世界で扱えるようにするための設計
 - モデリングのとある一部分とも考えられます
 - 永続化するデータ(=DBに保存するデータ)を設計するためにつかいります
 - 日常では「テーブル設計」と呼ばれたりもします
- 今日はDBモデリングについて話します
 - また、DBはRDB(Relational DB)とします

DBモデリングとコード実装の流れ

- 現実世界 → テーブル設計・ER図 → Rails migrationファイル
 - 今日はこの流れで考えていきます
- テーブル設計・ER図の替わりにクラス設計・クラス図から考えてもいいかもしれません
- 私は最初にテーブル設計から考えてます
 - 理由は「慣れているから」な気がします
 - 「永続化するもの」だけ先に考えると登場人物が少ないから私にとって考えやすいのかもしれません

第1問: レストランのメニューをテーブル設計をする

- 実践編
 - 実際にやってやり方をお伝えします
- 一般化編
 - 実践編でやったことを一般化して技術としてつかえるようにします
- 第1問の解説後に練習問題として第2問、第3問を出題します

テーブル設計のざっくりした流れ

- テーブル名になるものを書き出す
 - 主語になる「名詞」を抜き出してテーブル名にする
 - メインとなるテーブル名を出す
 - 「誰が・何が」を考えてテーブル名を出す
 - 「誰を・何を」を考えてテーブル名を出す
- 各テーブルのカラムを出す
- 残ってる、表現できていない情報があれば、テーブルになるか、カラムに追加するかを考える
- 関連を入れる
 - 1:1, 1:N, N:N
- 制約とデフォルト値とINDEXを考える

テーブル設計 実践編

- レストランのメニューを題材にテーブル設計を実際にやっていきましょう
- Railsで実装していく前提で書いています
 - 一部のカラム名や設計が他のフレームワークだと変わることもあるかもです



テーブル名を出す

- メインとなるテーブル名を出す
 - メニューは「注文するための道具」なので「注文」をメインとなるテーブルにします
 - テーブル名はordersとします
- 「誰が・何が」を考える
 - 「誰が注文するか」と考えて「顧客」テーブルを作ります
 - テーブル名はcustomersとします
- 「誰を・何を」を考える
 - 「何を注文するか」と考えて「商品」テーブルを作ります
 - テーブル名はitemsとします
 - 「ほうれん草のソテー」や「マルゲリータピザ」などです

各テーブルのカラムを入れる

- 各テーブルのカラムを入れる
 - かんたんなものからやっていきます

- items(商品)テーブル
 - code(商品番号) varchar型
 - 「11 ほうれん草のソテー」の"11"を商品番号とします
 - integer型でも良いかも
 - "01" といった表記にも対応できるので文字列型にしました
 - name(名前) varchar型
 - price(金額) integer型

- customers(顧客)テーブル
 - name(名前) varchar型
 - telephone_number(電話番号) varchar型
- 電話注文をお勧めしているので、名前と電話番号があれば良いだろうと考えています

- orders(注文)テーブル
 - count(注文個数) integer型
- シンプルにするために、まずは1注文1商品のケースで設計しています
 - 1注文複数商品への拡張もあとで考えてみましょう
- 注文商品名とその金額は商品テーブルにあるのでこのテーブルにはなくて良いとします

残ってる情報はあるか？

- 表現できていない情報があれば、テーブルになるか、カラムに追加するかを考えます
- 「ベジタブル前菜」や「ピザ」もテーブル設計できないか考えます
- 「商品カテゴリー」と考えます
- テーブル名はitem_categoriesとします
- item_categories(商品カテゴリー)テーブル
 - name(名前) varchar型
 - どの商品を含むかは関連で表現することにします

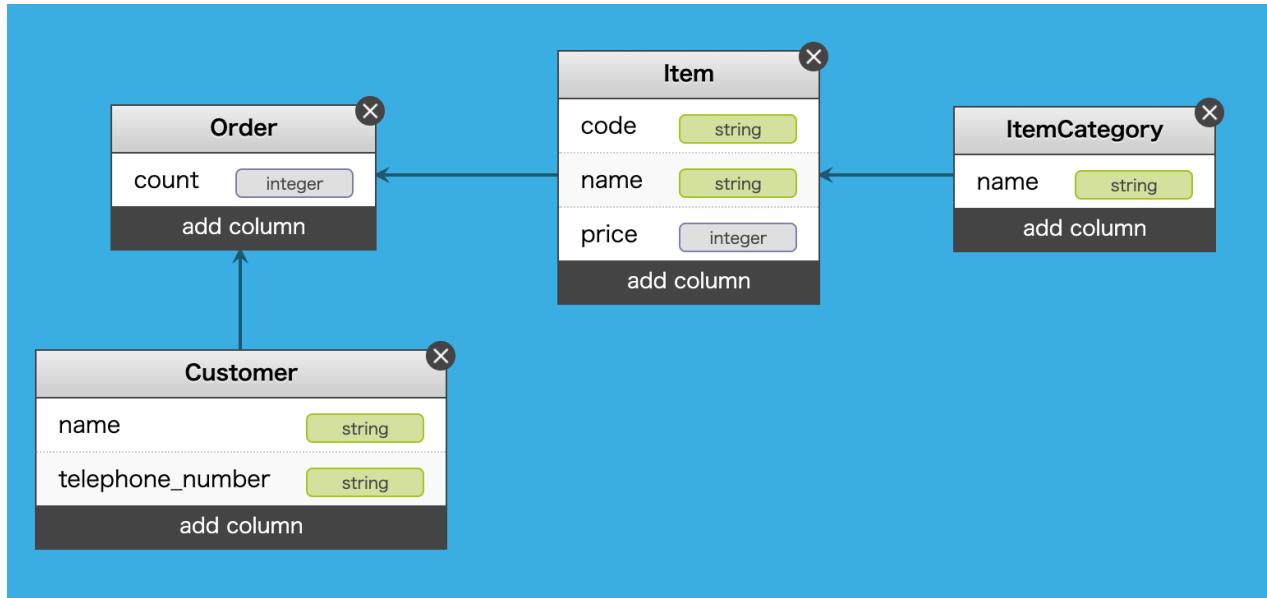
関連を入れる

- テーブル同士の関係を「関連」で考える
- 組になったときのお互いの数によって次のような関連になります
- 1対1(1:1) : 1つのレコードと1つのレコードが組になる
 - どちらかのテーブルに関連先テーブルidを持つ
- 1対多(1:N) : 1つのレコードが多くのレコードと組になる
 - 多側のテーブルに関連先テーブルidを持つ
- 多対多(N:N) : 両テーブルからお互いに1つのレコードが多くのレコードと組になる
 - 中間テーブルを作成して両方の関連先テーブルidを持つ

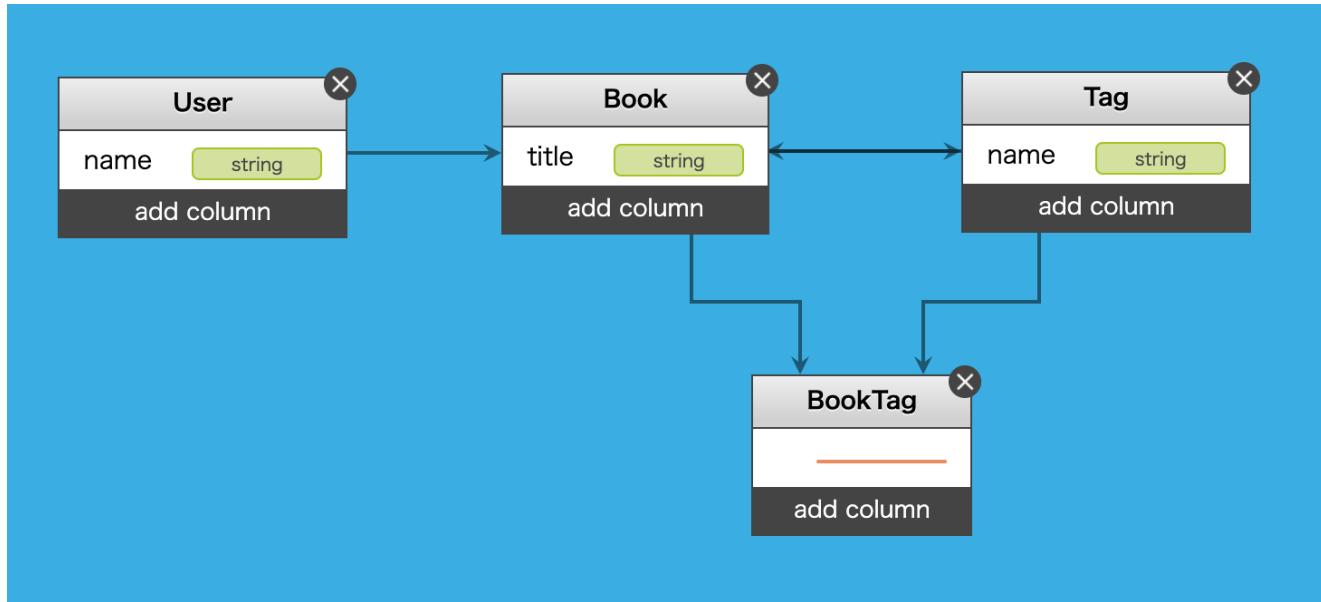
- orders(注文)テーブル
 - count(注文個数) integer型
 - item_id bigint型
 - customer_id bigint型
- customers(顧客)テーブル
 - name(名前) varchar型
 - telephone_number(電話番号) varchar型
- items(商品)テーブル
 - code(商品番号) varchar型
 - name(名前) varchar型
 - price(金額) integer型
 - item_category_id bigint型
- item_categories(商品カテゴリー)テーブル
 - name(名前) varchar型

ER図で表現する

- ER図で表現するとこんな感じになります
 - idとxxx_idは省略しています
- ER図の読み方は次のページで説明します
- RailsアプリでER図を自動で生成する方法もあります(後述)



ER図の読み方



- テーブル名单数形(= Railsではモデル名)をエンティティとしてタイトル部に書く
- テーブルのカラム名とその型を、エンティティの属性(Attribute)として書く
- 関連を書く(いくつかの書き方があり、矢印はその1つ)
 - 1:n(1対多)の関係を示すときに、nの側に矢印を書く
 - n:n(多対多)のときは両側に矢印がつく
 - UserとBookは1対多の関係、BookとTagは多対多の関係(中間テーブルbook_tags)

カラムごとにDB制約を加えるかどうかを考える

- DBの「制約」機能をつかうと、レコードとして許容できる値に制限をつけられる
 - 整合性のとれないデータが存在しないように防御することができる
- 一意性制約（ユニーク制約）
 - テーブル内でそのカラムが一意になるよう制約を課す
- 非NULL制約
 - そのカラムの値としてNULLを許可しない制約を課す
- 外部キー制約
 - 関連先レコードが必ず存在する制約を課す
- 今日は説明しませんが値の範囲を制限する「check制約(検査制約)」もあります
- 次のページから各制約をどのカラムに付けるかを考えていきます

一意性制約（ユニーク制約）

- テーブル内でそのカラムが一意になるよう制約を課す
- 以下に一意性制約をつけます
- customers(顧客)テーブルのtelephone_number(電話番号)
- items(商品)テーブルのcode(商品番号)

非NULL制約

- そのカラムの値としてNULLを許可しない制約を課す
- 以下に非NULL制約をつきます
- orders(注文)テーブル: count(注文個数), item_id, customer_id
- customers(顧客)テーブル: name(名前), order_id
 - 電話番号にもつけてもいいかもしません
- items(商品)テーブル: code(商品番号), name(名前), price(金額), item_category_id
- item_categories(商品カテゴリー)テーブル: name(名前)

外部キー制約

- 関連先レコードが必ず存在する制約を課す
- 以下に外部キー制約をつきます
- orders(注文)テーブルのitem_id, customer_id
- items(商品)テーブルのitem_category_id

カラムごとにデフォルト値をつけるかどうかを考える

- カラムのデフォルト値
- カラムの値を指定せずにレコード作成したとき、デフォルト値が与えられる
- 今回の題材では不要そうです

検索インデックスをつけるかどうかを考える

- インデックス: RDBで検索を高速にするための仕組み
 - 複数のカラムを組み合わせて検索するときに利用する「複合インデックス」もある
- 検索機能をつくるときに考えるとして、今はひとまずなしにします
- customers(顧客)テーブルのtelephone_number(電話番号)にはあってもいいかも
- items(商品)テーブルのcode(商品番号)にはあってもいいかも
- 参考資料
 - Railsパフォーマンス・チューニング入門 <https://slides.com/kokuyouwind/rails-performance-tuning>
 - MySQLとインデックスと私
<https://speakerdeck.com/yoku0825/mysqltoindetukusutosi>

できあがったテーブル設計をRailsアプリのmigrationファイルで書く

- サンプルコード: https://github.com/igaiga/rails6034_ruby266_erd_example/

```
class CreateCustomers < ActiveRecord::Migration[6.0]
  def change
    create_table :customers do |t|
      t.string :name, null: false
      t.string :telephone_number, index: { unique: true }

      t.timestamps
    end
  end
end
```

```
class CreateItemCategories < ActiveRecord::Migration[6.0]
  def change
    create_table :item_categories do |t|
      t.string :name, null: false

      t.timestamps
    end
  end
end
```

```
class CreateItems < ActiveRecord::Migration[6.0]
  def change
    create_table :items do |t|
      t.string :code, null: false, index: { unique: true }
      t.string :name, null: false
      t.integer :price, null: false
      t.references :item_category, null: false, foreign_key: true

      t.timestamps
    end
  end
end
```

```
class CreateOrders < ActiveRecord::Migration[6.0]
  def change
    create_table :orders do |t|
      t.integer :count, null: false
      t.references :item, null: false, foreign_key: true
      t.references :customer, null: false, foreign_key: true

      t.timestamps
    end
  end
end
```

1注文1商品から1注文複数商品へ拡張

- ここまで1注文1商品で考えていました
- 1注文で複数商品を注文できるようにしてみましょう
- ordersテーブルで複数の商品(items)を注文するような仕様に対応する
 - ordersとitemsを多対多の関連にする
 - たとえばorder_itemsという名前の中間テーブルを導入する
- order_itemsテーブル
 - order_id
 - item_id (ordersテーブルにあったitem_idをこちらのテーブルへ移動)
 - count (ordersテーブルにあったcountをこちらのテーブルへ移動)

テーブル設計作業 一般化編

- テーブル設計作業の流れは以下のようになります
 - さきほどの具体的な作業を一般化したものです
 - 復習だと思って聞いていただけたら
- メインとなるテーブル名を出す
 - 例: 注文
 - テーブル名は名詞で出す
- 「誰が・何が」を考える
 - 例: 顧客
- 「誰を・何を」を考える
 - 例: 商品

- 各テーブルのカラムを入れる
 - Railsではデフォルトで全テーブルでid, created_at, updated_atが生成されるのでこれらは基本追加で良く、書くときは省略して問題ない
- 残ってる、表現できていない情報があれば、テーブルになるか、属性に追加するかを考える
- 関連(リレーションシップ)を入れる
 - Railsの規約でテーブル名+_id
 - 多対多のときは中間テーブルをつくる
- カラムごとに制約を加えるかどうかを考える
 - postgresqlの制約一覧: <https://www.postgresql.jp/document/9.4/html/ddl-constraints.html>

- 主な制約
 - 一意性制約
 - テーブル中に同じ値のレコードは2つ以上存在しないことを保証する制約です
 - 一意性制約をつけると、既存レコードと同じ値を持つレコードをつくることが禁止されます
 - 非NULL制約
 - NULLを許可するかどうかを定めます
 - 非NULL制約をつけると、NULLを入れることが禁止されます
 - boolean型の場合は、原則NULL FALSEにしてtrue, falseの二値にすべきです
 - 外部キー制約
 - 外部キー(例: user_idなど別テーブルとの関連で使われるカラム)で指定したIDのレコードが、関連テーブルに存在することを保証する制約です
 - 外部キー制約がついていると、関連先のレコードが存在しないレコードをつくることが禁止されます

- カラムごとにデフォルト値をつけるかどうかを考える
 - 該当カラムの値を指定しないときに、デフォルト値が設定されているとその値が利用されます
- 検索INDEX、複合検索INDEXをつけるかどうかを考える
 - 検索時などに利用するINDEXが必要なカラムにINDEXをつけます

- Railsの規約に沿う
 - 規約として以下の場合は定められた規則の名前をカラム名につけます
 - id: 主キー
 - xxx_id: 外部キー
 - xxx_at: 日時型(慣習としてこの名前)
 - xxx_on: 日付型(慣習としてこの名前)

- Railsの予約語を避ける
 - 以下の単語は予約語として特別な意味を持つ
 - 用意された用途以外ではカラム名として使うことができません
 - lock_version: 楽観的ロック
 - type: STI
 - xxx_type: ポリモフィック関連
 - created_at, updated_at: 作成日時、更新日時
- 名前のコツとして以下があります
 - boolean型はis_xxxよりもavailable, activeなどの形容詞にするのがコツです
 - モデルにカラム名+?メソッドで真偽値を返すメソッドが自動で作成されるのでbook.available?といった形になってわかりやすいです
 - is_xxxはRailsでは変なのでやらない方が無難です

DBカラムの型

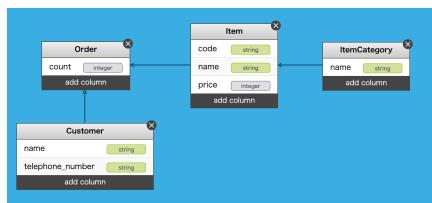
- integer: 整数
- bigint: 整数(integerよりも大きな数値を入れられる。IDのデフォルトで利用される。)
- timestamp: 日付と時刻(created_at, updated_atほかで利用されます)
- varchar(character varying): (可変長)文字列
- boolean: 論理値 (真/偽)
- DBによっても違いがある
 - postgresqlの型一覧:
<https://www.postgresql.jp/document/9.4/html/datatype.html>
 - MySQLの型一覧: <https://dev.mysql.com/doc/refman/8.0/ja/data-types.html>

道具の紹介

- ER図
 - 現実世界をRDB上の設計として図で示したもの
 - DBモデリングの道具
 - 永続化するものを扱う
- クラス図
 - 現実世界をコンピュータ上で扱うために図で示したもの
 - モデリングの道具
 - ER図よりも広い範囲を扱います

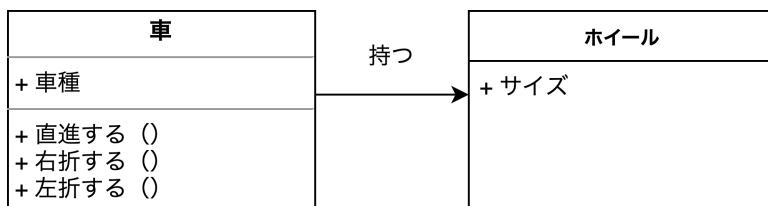
ER図(Entity Relationship Diagram)とは

- DBモデリングの道具
- RDB (Relational Database) の設計を書くための図
- RDBのテーブルをエンティティ (実体) として、テーブル間の関係をリレーションシップ (関連) として書く
- 永続化するものを扱う
- 属性(DBで言うカラム)を扱う
 - 操作(コードで言うメソッド)はER図の範囲外
- 関連を1:1, 1:多, 多対多などRDBで扱える形で図示する



クラス図とは

- モデリングの道具
- 現実世界をコンピュータ上で扱うために図で示したもの
- UMLで規定されている図の一つ
- プログラミング言語における「クラス」を扱う
- クラスの中には可視性や属性、操作を書くことができる
- クラス間の関係には多重度の他に、継承などを表現することができる



https://en.wikipedia.org/wiki/Class_diagram より引用

ER図を書く道具

- <https://www.diagrams.net>
 - Google driveへ保存できる
 - 旧名draw.ioが名前変更したようです
- <https://cacoo.com/ja/>
 - つかいやすそうなオンライン作図ツール
- <https://github.com/amatsuda/erd>
 - Railsでmigrationファイルを実装するとER図を生成してくれるgem
 - 次のページで説明します

RailsアプリでER図を自動生成する

- erd gemを使うとER図を自動生成することができます
 - <https://github.com/amatsuda/erd>
- 使い方
 - Gemfileのdevelopmentグループに `gem 'erd'` を追加します
 - `bundle install`を実行します
 - 描画ライブラリであるGraphvizをインストールします
 - <https://graphviz.org/>
 - macOSではhomebrewをつかって `brew install graphviz` コマンドでインストールできます
 - <http://localhost:3000/erd> へアクセスするとER図が描画されます

練習問題

- 以降の練習問題でDB設計をしてみてください
 - 前述のレストランメニューが第1問で、このあと第2問と第3問があります
 - 正解は1つではないので、自分で考えたテーブル設計をしてみてください
- 以下の2点を提出してください
 - ER図(細かいフォーマットまでは見ないので設計がわかれればOKです)
 - Railsのdb/schema.rbファイル
- メタ切: 6/30(水) 12:00
 - この講義時間中に質問や提出してもらったら一緒に議論できます
 - 採点などはしませんが、メタ切までに提出されたものは目を通してコメント書きます
- 提出方法: このリポジトリに自分の名前フォルダをつくってコミットしてください
 - <https://git.pepabo.com/tech/workshop/tree/master/igaiga-modeling-20210623>

第2問: 図書館の書籍予約申込

- 図書館で貸出中の書籍が戻った時に連絡をもらえる「書籍予約申込」を設計してみてください
- がんばれる人は、市内の別の図書館から取り寄せるようなケースも考えてみてください

第3問: minneの商品詳細ページ

- <https://minne.com/items/xxxxx>
 - 例: <https://minne.com/items/25736725>
- 難しいところはパスしてOKで、できるところまで設計してみてください
- カートと注文履歴は難しいですがぜひ挑戦してみてください

参考資料

- 大名エンジニアカレッジ Rails基礎コース 特別編データベース
 - udzura先生のDBモデリング講義
 - minneの画面からDBを設計する
 - 第3問を自分で設計した後で、うづら先生の解説を聞くと良い学びになります
 - <https://drive.google.com/file/d/1WF7iAYo71-82VhSy4NQPaQy5aEu8UcwY/view?usp=sharing>
 - ペパボさん社内Google Docs版URL

参考資料

- 楽々ERDレッスン
 - <https://www.shoeisha.co.jp/book/detail/9784798110660>
 - ER図をつかったテーブル設計の練習問題がたくさん載っています
 - 今日の講義の設計方法もこの本の内容を踏襲しています
 - 今日の範囲分はWeb公開されています
 - 「イベント系テーブル」という考え方も書いてありますので、興味がある方は読んでみてください
 - 第1回(レストランメニュー): <https://codezine.jp/article/detail/154>
 - 第2回(図書館の予約申込): <https://codezine.jp/article/detail/175>

参考資料

- 思考系UMLモデリング即効エクササイズ
 - <https://www.shoeisha.co.jp/book/detail/9784798107127>
 - クラス図やモデリングについてもっと知りたいときに
 - UML図全般をつかってモデリングする練習問題がたくさん載っています