django migration で学ぶデータベース設計

Atsushi Odagiri

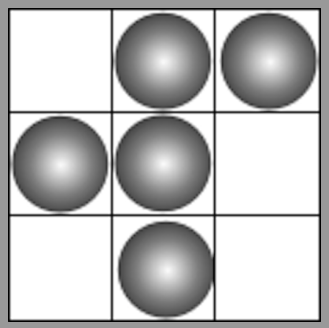
2023-10-07

Outline

1. django migration で学ぶデータベース設計
2. django migration
3. データベース設計
4. オブジェクト指向
5. オブジェクト指向からデータベース設計へ
6. まとめ

お前誰よ

- Atsushi Odagiri
- Open Collector
- Python は 1.5 くらいのころから



Open Collector

ケンオール

アプリケーション開発主導でスキーマを管理する

- django で開発するとモデル定義してそれに合わせたスキーマ設計することになる
- マイクロサービスの流れもデータベースはアプリケーションの持ち物という意識
- 昔は企業のデータベースが DBA に管理されててそこにテーブル設計を出してお伺いするなどしていましたね

はじめに

django には migration というデータベーススキーマの変更を管理するツールがあります。非常にありがたい存在の django migration がやってくれることを今一度確認してみましょう。データベース設計やオブジェクト指向設計などの面から安全で影響の少ないテーブル変更を考えてみましょう。

django migration がやること

- データベーススキーマの変更管理
- モデルの差分からスキーマ変更スクリプトを作成する
- スキーマ変更スクリプトの世代を進めたり戻したりする

スキーマの構成要素

- テーブル
- カラム
- インデックス
- 制約

スキーマ変更の実行コスト

- メタデータの変更のみ
- データ更新
- テーブル構造の再構築

メタデータの変更のみ

- 外部キー制約の変更
- インデックスの削除
- カラムのデフォルト値設定
- カラム名の変更
- テーブル名の変更

データ更新

- インデックスの追加
- カラムの（末尾への）追加

テーブル再構築

- 主キーの変更
- カラムの削除
- カラムのデータ型の変更
- NOT NULL, NULL 制約の変更
- カラムサイズの大幅な (サイズの表現に 1 バイト以内から 2 バイト必要になるなどの) 変更

スキーマ変更の下位互換性

- テーブルの追加
- カラムの追加
- 制約の緩和
- インデックスの有無

スキーマ変更履歴の管理

- django app ごとに世代管理
- django apps をまたぐ依存関係

さてどうしよう

- テーブルを小さく保つ
  - スキーマ変更時の実行コスト
  - スキーマ変更影響範囲
- 依存関係を簡潔なものにする
  - そのデータを触る django apps を減らす
- 小さく恥ずかしがりなデータ

正規化

- 正規化とは
- データを効率的に保存する
- 変更に強い

第一正規化

- 繰り返しの排除
- ARRAY とか使える DB もありますが…

第一正規化の例

元のテーブル

書籍番号	タイトル	著者	管理番号 1	管理番号 2
XXXXX0001	スーパープログラミング	aodag	100001	100002
XXXXX0002	紅茶の挿れ方	dag	200001	NULL

管理番号の繰り返しを排除

書籍番号	タイトル	著者	管理番号 1
XXXXX0001	スーパープログラミング	aodag	100001
XXXXX0001	スーパープログラミング	aodag	100002
XXXXX0002	紅茶の挿れ方	dag	200001

第二正規化

- 部分関数従属性の排除
- 候補キーの一部で確定可能なデータ

第二正規化の例

元のテーブル

書籍番号	タイトル	著者	管理番号	利用者
XXXXX0001	スーパープログラミング	aodag	100001	小田切篤
XXXXX0001	スーパープログラミング	aodag	100001	かしゅー
XXXXX0001	スーパープログラミング	aodag	100001	けーわい
XXXXX0001	スーパープログラミング	aodag	100001	ときびと

書籍番号で確定可能な部分を分割

書籍番号	タイトル	著者
XXXXX0001	スーパープログラミング	aodag

書籍番号	管理番号	利用者	貸出日	返却日
XXXXX0001	100001	小田切篤	1970-01-01	1970-01-02
XXXXX0001	100001	かしゅー	1970-01-02	1970-01-03
XXXXX0001	100001	けーわい	1970-01-03	1970-01-04
XXXXX0001	100001	ときびと	1970-01-04	1970-01-05

第三正規化

- 推移的関数従属性
- 候補キー以外の項目で確定可能なデータ

第三正規化の例

元のテーブル

書籍番号	タイトル	著者	管理番号	貸出
XXXXX0001	スーパープログラミング	aodag	100001	小田切篤
XXXXX0001	スーパープログラミング	aodag	100002	NULL
XXXXX0002	紅茶の挿れ方	dag	200001	NULL

管理番号で分割

書籍番号	タイトル	著者	管理番号
XXXXX0001	スーパープログラミング	aodag	100001
XXXXX0001	スーパープログラミング	aodag	100002
XXXXX0002	紅茶の挿れ方	dag	200001

管理番号	貸出
100001	小田切篤

ユースケース駆動の設計

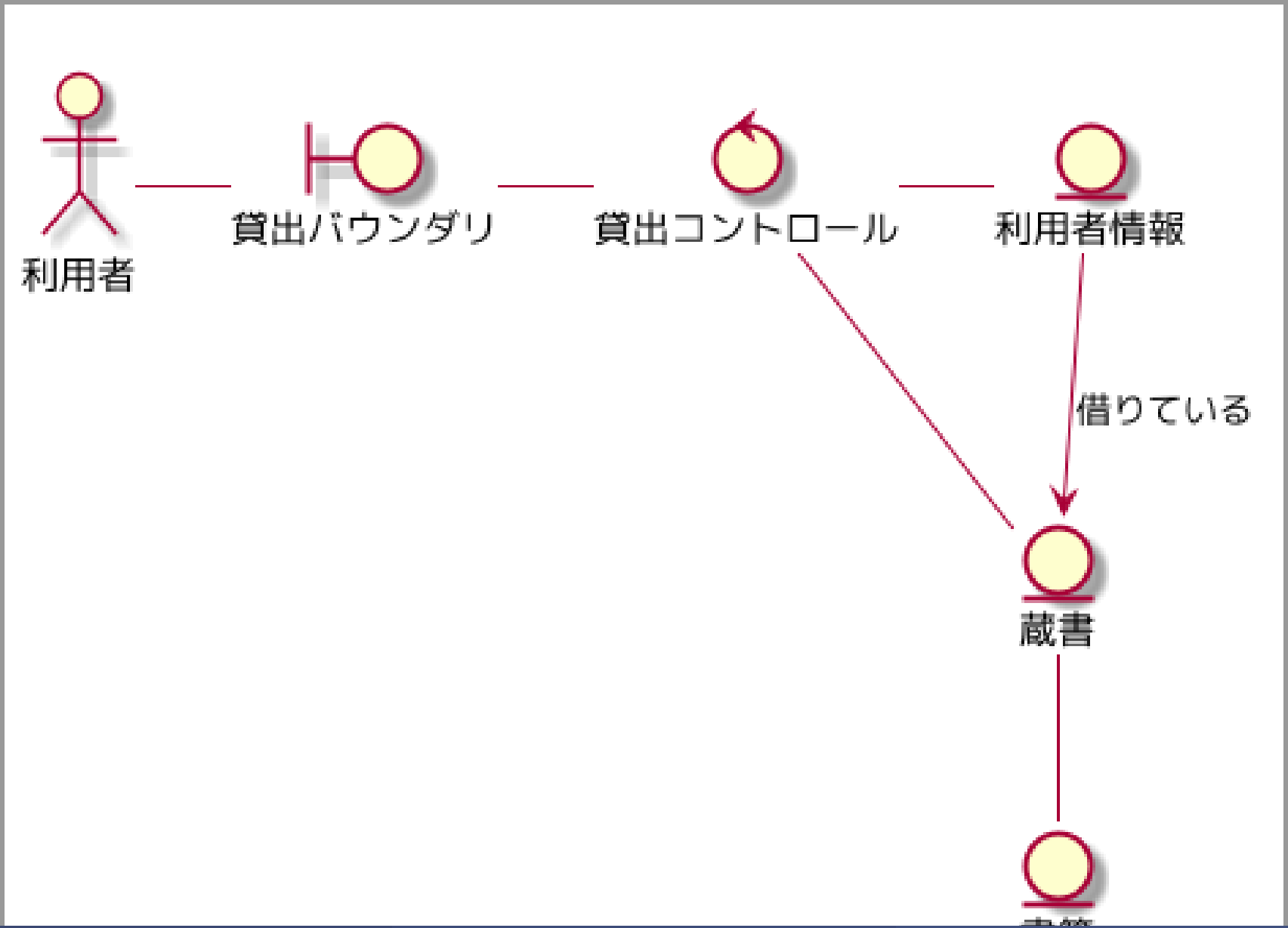
- ユースケース: アクターが複数の機能を利用してなんらかの目的を達成する
- アクター: システム外のなにか (人とか時間とか別システムや現実世界のイベントとか)
- 機能: 入力を処理して出力を返すもの

ドメインオブジェクト

- 明らかなもの
- ドメイン分析で発見できるもの
- リファクタリングで発見もの
- だいたいエンティティとして DB に保存する

ICONIX 手法のロバストネス図

図書館モデルで貸出するユースケースの例



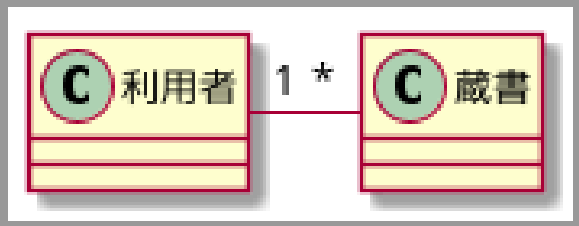
ICONIX 分析のルール

- 1 ユースケースに 1 バウンダリ、1 コントローラーで開始する
- コントローラーがユースケースに関わるエンティティを操作する
- コントローラーが関わるエンティティが 1 つだけならそのコントローラーを削除する
- コントローラーが関わるエンティティが複数ある場合
  - 新たな概念として名前をつけ、エンティティになるか考える
  - そうでなければユースケース内の純粋なロジックとして残す

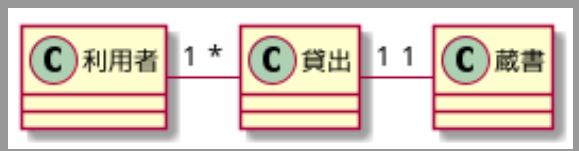
関連オブジェクト

- 利用者が蔵書を借りているという状態の表現
- 利用者と蔵書を関連付ける
- 利用者から蔵書への one-to-many で良いか？
- 貸出日などの付加的な情報
- 貸出という概念を発見

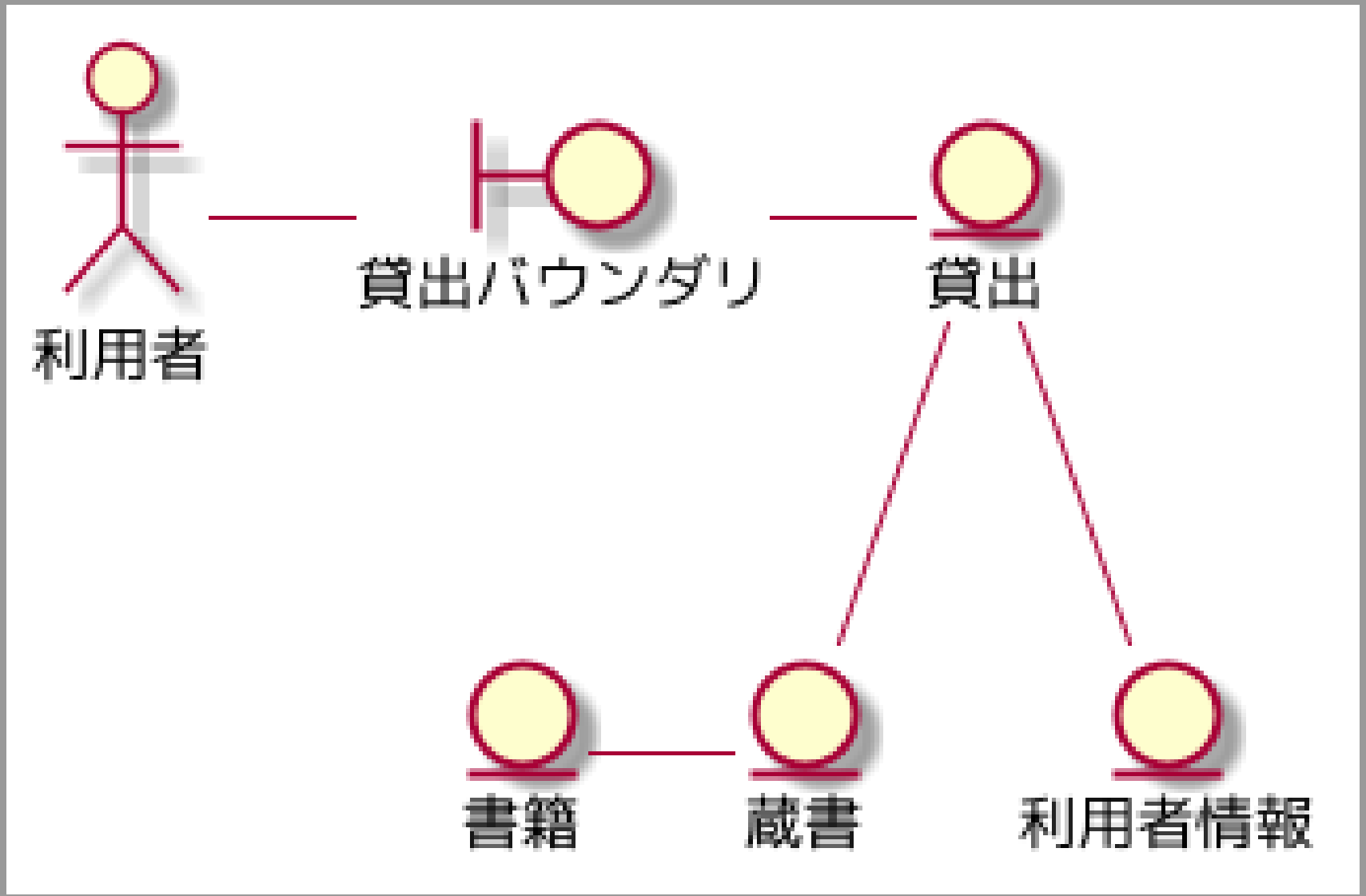
利用者と蔵書を直接関連させる



利用者と蔵書の関連をオブジェクトで表す



コントロールクラスをエンティティに変更



関連オブジェクトに分離した意味

- 利用者と蔵書を直接扱うのではなく貸出オブジェクトの生成に責任を分離
- 貸出という処理で利用者や蔵書の詳細な情報は必要ない
- 貸出ユースケースを取り扱う django app の独立性が高くなる

オブジェクト指向設計原則 (SOLID)

- 単一責任の原則
- 開放閉鎖の原則
- リスコフの置換原則
- インターフェース分離の原則
- 依存性逆転の原則

開放閉鎖の原則

- 他のユースケースに影響を与えない
- 該当ユースケースの機能変更だけで済ませる
- 貸出ユースケースで貸出オブジェクトの変更は他のユースケースに影響しない
- 貸出方法が変わっても蔵書や利用者の管理に影響しない

インターフェイス分離

- 必要のないモデルにアクセスしない
- 貸出オブジェクトを作成するために蔵書や利用者の詳細な情報にアクセスする必要はない
- 蔵書や利用者の構成が変わっても大きな影響を受けない

オブジェクト指向からデータベース設計へ

- 貸出に関する情報として貸出日を追加
- 利用者と貸出日は同時に更新される
- 候補キーとは別の情報に依存した情報が存在 -> 推移性関数従属
- 第三正規化で排除

書籍番号	タイトル	著者	管理番号	利用者
XXXXX0001	スーパープログラミング	aodag	100001	小田切篤
XXXXX0001	スーパープログラミング	aodag	100002	NULL
XXXXX0002	紅茶の挿れ方	dag	200001	NULL

ライフサイクルから発見するエンティティ

- 利用者と貸出日を分離して貸出テーブルにする
- ロバストネス分析で発見した貸出エンティティと一致するもの

書籍番号	タイトル	著者	管理番号
XXXXX0001	スーパープログラミング	aodag	100001
XXXXX0001	スーパープログラミング	aodag	100002
XXXXX0002	紅茶の挿れ方	dag	200001

管理番号	利用者	貸出日
100001	小田切篤	1970-01-01

オブジェクト指向、データベース設計を行き来しよう

- データベース設計でエンティティを発見すること
- オブジェクト指向設計で正規化対象を発見すること
- どちらからのアプローチでも近い結果が得られる

django apps とモデル設計に活かす

- モデルのライフサイクルに着目
- ある django apps で生成されるモデルに別の django app が情報を追加していく
  - ドメインオブジェクトの発見：その情報に名前をつけ、モデルなのでは？
  - インターフェイス分離：その django app ではそのモデルだけを処理対象にできるのでは？
  - 開放閉鎖の原則: 生成もとの django app に影響を与えずに新たなモデルのみを変更可能か？

モデル分割の実際

A アプリのモデル `ModelA` から, B アプリの `ModelB` に一部分離する

- B アプリ `ModelB` のテーブル追加
- A アプリ `ModelA` のテーブルから該当フィールドに対応するカラムのデータを `ModelB` のテーブルにコピー
  - UPDATE JOIN とか UPDATE SELECT とかデータベースごとに違う！
  - あとこれ A と B のどっちの migrations に入れるべき？
- A アプリから該当カラム削除
  - ALTER TABLE a_amodel DROP COLUMN ...
  - テーブル再構築のコストがかかるやつ
  - しかも一度に複数カラムを DROP できない
- カラム削除があるのでカナリアやブルーグリーン不可能

モデル分割

- 一時的にテーブルの一部分だけを他のテーブルに見せかけることができれば！
- データマッパーの ORM だったらできたのに！

更新可能 VIEW！

- django の view じゃないよ
- RDBMS の機能
- ある条件で作成された VIEW は更新も可能
  - クエリ対象が 1 テーブルのみ
  - 集約を含まない
  - SELECT に主キーを含む
  - SELECT に計算を含まない
  - ...

とはいえ

- レイヤーの違う部分のハックは別の問題を持ち込みやすいので注意
- 問題解決にとてもコストがかかる...

まとめ

- マイグレーションの中でもコストや下位互換性など特性が異なる操作がある
- データベース設計やオブジェクト指向設計などの知見を活かしましょう
  - 他にも活かせるものはたくさんあるはず
- 小さいことはいいことだ
- 原理原則やベストプラクティスに生きていけないから人は悩むのです

参考文献

- プログラマのための SQL 第 4 版 すべてを知り尽くしたいあなたに, Joe Celko, ISBN ISBN-13978-4798128023
- ユースケース駆動開発実践ガイド, ダグ・ローゼンバーグ, ISBN 978-4798114453
- Let's POSTGRES! / PostgreSQL 9.3 の新機能: 更新可能 VIEW, https://lets.postgresql.jp/documents/technical/9.3/updatable_view/1
- MySQL 8.0 リファレンスマニュアル / 15.12.1 オンライン DDL 操作, https://dev.mysql.com/doc/refman/8.0/ja/innodb-online-ddl-operations.html