Engineer

ORM

データベースのテーブルをコード内のオブジェクトに対応付けるライブラリー

どのようなものか

ORMは、object-relational mapper、オブジェクト関係マッパーの略で、関係データベースのテーブルをアプリケーションコードのクラスに、行をオブジェクトに対応付けるライブラリーです。著者を求めると `Author` オブジェクトが返り、その属性を変えて保存すると、ORMが必要な `SELECT` 、 `INSERT` 、 `UPDATE` 文を書きます。どちらもPythonで書かれたDjangoのモデルとSQLAlchemyのORMは、よく知られた例で、この記事も、この二つを基にします。

対応付けが必要なのは、両側でデータの整理方法が違うためです。Martin Fowlerは、Data Mapperパターンの説明で、明快に述べます。オブジェクトと関係データベースでは、データを構造化する方法が違い、コレクションや継承など、オブジェクトモデルの一部には、関係データベース側の対応物がありません。両側の間でデータを移すこと自体が問題になり、ORMは、そのために用意された解決策です。

その答えの各部分は、Fowlerの2003年刊『Patterns of Enterprise Application Architecture』で名付けられ、パターンのカタログはオンラインにあります。全体の形を表すのは、二つのパターンです。Active Recordは、テーブルの一行を包み、自身のDBアクセスを扱い、そのデータのドメインロジックを持つオブジェクトです。Djangoの設計思想は、モデルがこのパターンに従うとします。Data Mapperは、オブジェクトとデータベースを互いに独立させたまま、データを移す別の層で、オブジェクトは、DBの存在を知る必要がありません。周りには、より小さなパターンがあります。Identity Mapは、各オブジェクトを一度だけ読み込みます。Unit of Workは、一つの作業が変更したオブジェクトの一覧を保ち、変更をまとめて書き出します。Lazy Loadは、必要なデータの一部だけを持ちながら、残りの取得方法を知るオブジェクトです。SQLAlchemyのドキュメントは、二つを直接挙げます。 `Session` は、読み込んだオブジェクトをアイデンティティーマップに持ち、同じ主キーには一つだけを保ちます。変更を集め、DBへフラッシュする方法が、Unit of Workです。

概念を判断に生かす

「ORMとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 データの保存先、処理のルール、他のシステムとの接続方法など、利用者の目に見えない部分をつくります。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 ETL/ELTパイプライン6本を設計・運用 — 6 3種類のDBエンジンにまたがるETLパイプライン

ETL/ELTパイプライン6本を設計・運用 お問い合わせ →

なぜ重要なのか

一つ目の理由は、減らせるコード量です。2012年の論考『ORM Hate』でFowlerは、オブジェクト関係マッピングを難しい問題とします。同じデータの、大きく異なる二つの表現を揃え、片方の変更を、もう片方にも反映する必要があるためです。ORMが扱えるのは、マッピング問題の約80–90%と見積もり、80%の作業を省くフレームワークなら、そこで止まっても価値があると論じます。手で対応付けないテーブルが一つ増えるごとに、列を間違った属性へ読む場所が一つ減ります。

二つ目は、実行中のプログラム内の一貫性です。アイデンティティーマップがあれば、同じ行を見つける二つのクエリーは、同じオブジェクトを返し、一方の参照を通した変更が、他方からも見えます。SQLAlchemyのドキュメントは、同じ主キーを二度読み込み、同一のオブジェクトが返る例を示します。Unit of Workでは、ORMが保留中の挿入、更新、削除を集め、機能一覧の言葉では、一つのバッチですべてフラッシュします。アプリケーションは、触れたオブジェクトを覚えておく必要がありません。Djangoは、意図して逆を選びます。裏で保存するのでなく、明示的な `save()` を求め、送るSQL文をできる限り少なくする設計です。

三つ目は、境界の位置です。Fowlerは、良いORMは定型作業を扱い、必要なら「SQLへ降りて作業できるマンホールを用意する」と説明します。関係データベースを使ってアプリケーションを作る人は、データベースの仕組みを知るべきだとも明言します。Djangoの設計思想も、フレームワーク側から同じことを言います。データベースAPIは、すべての答えでなく近道であることを前提にし、文全体でも一つの `WHERE` 句でも、独自SQLを書きやすくすべきです。ORMが価値を持つのは、データアクセスの大半を占める定型部分で、残りにチームのSQL技術を使います。

実際の活用例

ORMが最も隠しやすいコストは、送るクエリー数です。SQLAlchemy 2.1の関係は、既定で遅延読み込みです。コードが属性に初めて触れたとき、関連オブジェクトを取得します。ドキュメントは、N個を読み込んだ後、その遅延属性に触れると、N+1の `SELECT` 文が発生すると説明します。以下では、 `Author` と `Book` はすでに対応付け済みで、一対多の `Author.books` 関係があるとし、著者を三通りでたどります。

```
from sqlalchemy import select
from sqlalchemy.orm import raiseload, selectinload

# 既定の動作：著者の取得にSELECTを一回、ループで本にアクセスする
# 著者ごとにさらに一回実行し、合計N+1文になります。
authors = session.scalars(select(Author)).all()
for author in authors:
    print(author.name, len(author.books))

# select IN読み込み：二つ目のSELECTで、全著者の本をまとめて取得します。
# IN句には著者の主キーを指定します。
stmt = select(Author).options(selectinload(Author.books))
for author in session.scalars(stmt).all():
    print(author.name, len(author.books))

# raise読み込み：author.booksにアクセスすると、追加クエリーを
# 黙って送る代わりにエラーを発生させます。
stmt = select(Author).options(raiseload(Author.books))
```

一対多、多対多のコレクションでは、ドキュメントは、通常 `selectinload()` が最良の戦略だとします。最も汎用的とする `joinedload()` は、元の文に `JOIN` を加え、既定では `LEFT OUTER JOIN` にして、関連行を同じ結果集合で返します。 `raiseload()` は、意図しない読み込みを防ぎ、ワイルドカードの `raiseload("*")` は、文が実際に読み込むオブジェクトの関係も含め、ほかの関係にも防御を広げます。

Django 6.1では、同じ選択にfetch mode、取得モードという名前があります。最初のクエリーで読み込まなかったフィールド、たとえば外部キー、一対一の関係、 `defer()` か `only()` で除外したフィールドに触れると、既定の `FETCH_ONE` は、そのインスタンスだけの値を取得します。ドキュメント自身が、これをN+1クエリー問題と呼びます。6.1で加わった `FETCH_PEERS` は、同じ `QuerySet` から来た、そのインスタンスとほかのすべてのインスタンスについて、欠けたフィールドを取得します。一行で必要なものは、通常すべての行で必要だという考えです。代わりに `FETCH_RAISE` を使うと、 `FieldFetchBlocked` を発生させます。モード

は、`QuerySet`ごとに、`Book.objects.fetch_mode(models.FETCH_PEERS)`のように設定するか、カスタムマネージャーでモデルの既定値にします。Djangoは、取得した関連オブジェクトにもコピーします。`FETCH_PEERS`が適さないか、十分効率的でなければ、最適化ガイドは、`select_related()`と`prefetch_related()`を案内します。

自社のページは、多くの設計より、境界をデータベース側へ寄せています。技術スタックは、SQLをORMの生成物でなく、第一級の言語として扱います。依頼される性能改善の大半が、そこにあるからです。NextMarinerでの取り組みでは、創業者が、その考えを関数中心のPostgreSQLデータ層で徹底しました。アプリケーションの全操作を、34スキーマに分かれる1,275のストアド関数のいずれかとし、Goからテーブルへの直接アクセスは、一切ありません。APIがログインするロールには、その関数への`EXECUTE`と、そのスキーマへの`USAGE`以外の権限がなく、テーブルを直接扱えません。ページは代償も明かします。新しいフィールドを加えるには、クエリーに一行足すのでなく、テーブルと関数の両方を変えます。

問題が起きるところ

ループが、行ごとにクエリーを送ります。 小さい開発用DBでは、先のN+1は無害に見えても、行数に応じてページを遅くします。重要な各経路で、読み込み方を決めます。SQLAlchemyの`selectinload()`か`joinedload()`、Djangoの`FETCH_PEERS`、`select_related()`、`prefetch_related()`です。想定外の取得は、明確に失敗させます。SQLAlchemyなら`raiseload()`、Djangoなら`FETCH_RAISE`で、後者は、性能が重要なコードで、意図しないクエリーを防ぐ手段と説明されています。

ORMが書くSQLを、誰も読みません。 ORMは、チームの誰も選ばなかったSQLを書き、ページが遅くなるまで見えません。Djangoの最適化ガイドは、ここから始めます。ページが動かすクエリーとコストを調べ、`QuerySet.explain()`で、ある`QuerySet`をDBがどう実行するかを確認します。コードの代わりにDjangoが書くものは、`django.db.connection.queries`で読みます。重要なページやジョブのレビューに、SQLを読むことを含め、遅い文は、DB自身の実行計画で調べます。

DBがすべき処理を、Pythonで行います。 行を読み込んでから、絞り込み、二つのフィールドの比較、合計を行うと、データがある場所でDBができる仕事のために、全行を接続越しに運び、アプリケーション側で処理することになります。Djangoのガイドは、DBで絞る`filter()`と`exclude()`、同じモデル内のフィールドを比べる`F()`式、DBで集約する`annotate()`を挙げます。書き込みも同じで、`bulk_create()`と`bulk_update()`を、オブジェクトごとの`save()`の代わりに使い

ます。ただし、両方のメソッドに関する注意点を、先にドキュメントで確認します。非常に大きな結果集合では、`QuerySet`が読んだものをすべて保持して、大量のメモリーを使う場合があり、`iterator()`で改善することがあります。

最後の部分まで、無理にORMを通します。 Fowlerの見積もりでも、マッピング問題の10–20%は残り、そこには常に慎重な作業が必要だと述べます。特定のSQLを出すようにORMを無理に調整すると、その仕事は、誰も選ばなかった生成SQLの中に隠れます。その文はSQLで書き、ほかのコードと同じようにレビューします。Djangoのガイドは、下へ降りる段階を示します。`RawSQL`式でクエリーに明示的なSQLを加えると、移植性は下がりますが、表現力は増します。その先では、独自SQLでデータを取得したり、モデルにデータを設定したりできます。ORMは、最も手間を減らし、コストも小さい、定型的なレコード処理に残します。

参考資料

- エンタープライズアプリケーションアーキテクチャパターンのカタログ — Martin Fowlerによる、Active Record、Data Mapper、Unit of Work、Identity Map、Lazy Loadの要約です。
- ORMへの反感 — オブジェクト関係マッピングが難しい理由と、フレームワークが担える範囲です。
- 関連の読み込み手法 — SQLAlchemy 2.1の遅延、select IN、結合、エラーによる取得制御と、選び方です。
- 取得モード — Django 6.1で加わった`FETCH_ONE`、`FETCH_PEERS`、`FETCH_RAISE`です。
- データベースアクセスの最適化 — プロファイリング、`QuerySet`の評価、DB内の仕事、一括操作、直接SQLです。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「ORMとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/orm/