データベース設計ドキュメント

概要

このプロジェクトでは、SQLiteとChromaDBの2つのデータベースを使用し、それぞれの役割を明確に分離しています。データロックがかからない設計を実現するため、読み取りと書き込みを分離し、書き込みキューを使用しています。

データベースの役割分担

SQLite

役割: 構造化データの永続化

用途:

- 組織情報（organizations）
- 組織メンバー情報（organizationMembers）
- 事業会社情報（companies）
- 組織・会社表示関係（organizationCompanyDisplay）
- 議事録（meetingNotes）
- 注力施策（focusInitiatives）
- エンティティ（entities）- メタデータのみ
- 関係（relations）- メタデータのみ
- トピック（topics）- メタデータのみ
- システム設計ドキュメント（designDocSections, designDocSectionRelations）

特徴:

- ACIDトランザクション保証
- リレーショナルデータの管理
- 構造化クエリ（JOIN、集計など）
- ChromaDB同期状態の管理（chromaSynced, chromaSyncError, lastChromaSyncAttempt）

ChromaDB

役割: ベクトル検索とセマンティック検索

用途:

- エンティティの埋め込みベクトル（entity_embeddings）
- 関係の埋め込みベクトル（relation_embeddings）
- トピックの埋め込みベクトル（topic_embeddings）
- システム設計ドキュメントの埋め込みベクトル（design_doc_embeddings）

特徴:

- 高次元ベクトルの保存と検索

- セマンティック類似度検索
- メタデータとベクトルの組み合わせ検索
- SQLiteとは独立したデータストア

データロック回避の設計

書き込みキューシステム

すべてのデータベース書き込み操作は、単一の書き込みワーカー（`WriteWorker`）を経由します。

アーキテクチャ:

```
フロントエンド/API
    ↓
書き込みキュー（async_channel）
    ↓
WriteWorker（単一スレッド）
    ↓
SQLite（書き込み専用コネクション）
```

利点:

- 書き込み操作の順序保証
- デッドロックの回避
- トランザクションの適切な管理
- エラーハンドリングの一元化

読み取り操作

読み取り操作は、接続プール（`DatabasePool`）から取得したコネクションを使用します。

特徴:

- 複数の読み取り操作を並列実行可能
- 書き込み操作と競合しない
- 読み取り専用トランザクションを使用

ChromaDB操作

ChromaDB操作は、Rustの非同期クライアント（`ChromaClient`）を使用します。

特徴:

- SQLiteとは独立した操作
- 非同期処理
- エラーハンドリングとリトライ機能

同期メカニズム

SQLite → ChromaDB同期

1. SQLiteにデータを書き込み（`chromaSynced = 0`）
2. バックグラウンドでChromaDBに埋め込みベクトルを保存
3. 同期成功時に`chromaSynced = 1`に更新
4. エラー時は`chromaSyncError`にエラーメッセージを記録

同期状態の管理

各テーブルには以下のカラムが追加されています：

- `chromaSynced`: 同期状態（0: 未同期、1: 同期済み）
- `chromaSyncError`: 同期エラーメッセージ（NULL: エラーなし）
- `lastChromaSyncAttempt`: 最後の同期試行日時

ポート設定

開発環境

- Next.js開発サーバー: ポート3010
- Rust APIサーバー: ポート3011（デフォルト、環境変数で変更可能）
- ChromaDB Server: ポート8000（環境変数で変更可能）

本番環境

- Rust APIサーバー: ポート3011（環境変数で変更可能）
- ChromaDB Server: ポート8000（環境変数で変更可能）
- Next.js: 静的エクスポート（Node.js不要）

注意:

- Next.jsとRust APIサーバーは異なるポートを使用（競合回避）
- ポートは環境変数（`API_SERVER_PORT`, `CHROMADB_PORT`）で変更可能
- ポート競合時は自動的に利用可能なポートを検出
- ChromaDB Serverが起動しない場合はSQLite全文検索モードにフォールバック

詳細:
ポート設定、サーバー構成、フォールバック仕様の詳細については、ポート設計とサーバー構成の設計書を参照してください。

ルーティング方式

動的ルーティングではなく、クエリパラメータ方式を使用しています。

例:

- `/companies/detail?id=123` （動的ルーティング `/companies/detail/[id]` の代わり）
- `/organization/detail?id=456&tab=focusInitiatives`

利点:

- 静的エクスポートが可能
- Node.jsサーバーが不要
- Rustアプリケーションで直接配信可能