TanStack DB ~状態管理の新しい考え方~

TanStackファミリー

- **TanStack Query** - Server State Management
- **TanStack Router** - Type-Safe Routing
- **TanStack Table** - Headless Table Building
- **TanStack Form** - Headless Form Building
- **TanStack DB** - Client-Side Database ← **NEW!**

TanStack DBとは？

- フロントエンドに **永続化層(DB)** を設けてコンポーネントからクエリする
- **クエリファーストなAPI**
- React だけでなく **Vue / Svelte / Solid** など多数のUIフレームワークをサポート

設計思想

Client Side Querying

→ データ取得の主導権をフロント側へ

Reactでの使い方の例

```
export const drawCalcCollection = createCollection(
  localStorageCollectionOptions({
    storageKey: "tcg-tool-draw-calculations",
    id: "draw-calculations",
    getKey: (item) ⇒ item.id,
    schema: drawCalcSchema
  }),
);
```

- Collectionという単位でデータの永続化を定義
- ZodやValibotなどでランタイムバリデーションありの型定義可能

```
const { data } = useLiveQuery((q) ⇒
  q
    .from({ calculations: calculationsCollection })
    .where(({ calculations }) ⇒ eq(calculations.id, calculationId)),
);
```

- DrizzleのようなSQLライクなセレクター
- 型安全なクエリビルダー
- クエリで絞り込まれた箇所のみの部分的な**Subscribe**を実現

コンポーネントで使う

```
const CalculationItem = ({ calculationId }: { calculationId: string }) => {
  const { data } = useLiveQuery((q) =>
    q.from({ calculations: calculationsCollection })
      .where(({ calculations }) => eq(calculations.id, calculationId)),
  );
  const firstCalculation = data[0];
  if (!firstCalculation) return <FallbackItem />;
  return (
    <Card.Root>
      <Card.Header>
        {firstCalculation.name}
      </Card.Header>
      <Card.Body>
        {firstCalculation.result}
      </Card.Body>
    </Card.Root>
  );
}
```

派生Collectionも可能

```
// 条件をあらかじめ指定
export const drawCalcHistoryCollection = createLiveQueryCollection((q) ⇒
  q.from({ calculations: drawCalcCollection })
  .orderBy(({ calculations }) ⇒ calculations.updatedAt, "desc"),
);

// 引数ありの動的なCollection
export const createDrawCalcByGameCollection = (gameTemplate: string) ⇒
  createLiveQueryCollection((q) ⇒
    q.from({ calculations: drawCalcCollection })
    .where(({ calculations }) ⇒ eq(calculations.gameTemplate, gameTemplate))
    .orderBy(({ calculations }) ⇒ calculations.updatedAt, "desc"),
  );
```

TanStack DB、なにが美味しい？

永続化層を 自由に差し替え可能

- localStorage
- インメモリ
- REST API
- Electric SQL
- TrailBase

Supabase, IndexedDB, DuckDB WasmなどもAdapterを用意することで利用可能になるかも

バックエンド実装が不要に？

- **フロントエンド中心のデータ設計** が可能
- RLS（Row Level Security）を組み合わせれば都度問い合わせの必要なし
- 特定の データベースSaaS の実装に依存せずに済む

状態管理のベストプラクティスを変える？

- Zustand / Jotai のような状態を
  → **SQLライクなクエリ + スキーマ** に置き換え可能
- 単なる状態管理にとどまらず、データの保管にも利用できる
- **認知負荷を大幅に削減** できるかも！

おまけ

Claude Codeと一緒に既存のアプリケーションをTanStack DBに載せ替えしてみた

できたもの

https://tcg-tool.pages.dev/draw-calc-db

ソースコード見たい方向け

https://github.com/bmthd/tcg-tool/pull/3/files

まとめ

- 「**フロントエンド中心のデータ設計**」を加速させる存在
- 全てのServer Stateの抽象化層になりうる
- フロントエンドの状態管理だけでなくバックエンド設計にも影響を与える可能性大