B1E Praktischer Nachweis

Ich kann ein Datenmodell für eine NoSQL Datenbank entwerfen.

Fragenstellung und Lernziele

Fragenstellung: Wie entwirft man ein flexibles und performantes Datenmodell für eine NoSQL-Datenbank, exemplarisch an einem E-Commerce-System?

Lernziele:

- Verständnis der Kernkonzepte des NoSQL-Datenmodell-Designs, insbesondere im Kontext von Dokumentendatenbanken.
- Fähigkeit, die Anforderungen einer Anwendung (E-Commerce) in ein geeignetes NoSQL-Datenmodell zu übersetzen.
- Analyse und Begründung von Designentscheidungen wie Denormalisierung, Einbettung (Embedding) vs. Referenzierung (Linking).
- Erstellung eines beispielhaften Datenmodells (im JSON-Format).
- Bewertung der Auswirkungen des Datenmodells auf typische Abfragemuster und Skalierbarkeit.

Umsetzung

Kernkonzepte des NoSQL-Datenmodell-Designs (Dokumentendatenbanken)

1. **Aggregat-Orientierung**: Daten, die zusammengehören und häufig gemeinsam abgerufen werden, sollten in einem einzigen Dokument (Aggregat) gespeichert werden. So werden Joins minimiert.
2. **Schema-Flexibilität**: Dokumente in einer Collection können unterschiedliche Felder haben. Neue Felder werden ohne Migrationsaufwand ergänzt.
3. **Priorisierung von Leseleistung**: Häufige Lesezugriffe werden optimiert – oft durch Denormalisierung, um Joins zu vermeiden. Schreibvorgänge können dadurch komplexer werden.
4. **Skalierbarkeit**: Horizontale Verteilung (Sharding) wird unterstützt. Das Datenmodell sollte hierfür vorbereitet sein.

Anforderungen einer Anwendung in ein NoSQL-Datenmodell übersetzen

1. **Entitäten identifizieren**: Hauptobjekte sind in unserem E-Commerce-System z.B. `User`,

`Product`, `Order`, `Review`.

2. **Beziehungen definieren**:

- Ein `User` kauft viele `Orders`.
- Eine `Order` enthält viele `Products`.
- Ein `User` kann viele `Reviews` schreiben, und ein `Product` hat viele `Reviews`.

3. **Abfragemuster analysieren**:

- Anzeige einer Order mit allen Artikeln und Versandadresse.
- Auflistung aller Orders eines Users.
- Anzeige eines Products mit allen Reviews und Rezensenteninfo.
- Suche nach Produkten in einer Kategorie oder nach Neuerscheinungen.

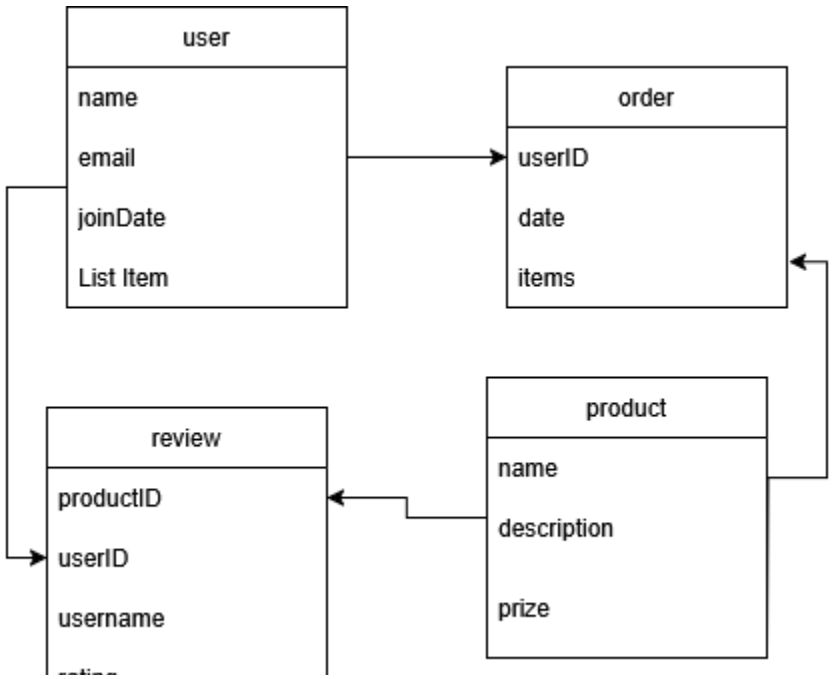
4. **Performance- und Skalierbarkeitsziele**:

- Tausende Lese-/Schreibvorgänge pro Sekunde
- Datenvolumen in den Collections wächst mit Bestellungen und Reviews.

Beispielanwendung: E-Commerce-System

Entitäten:

- `User`
- `Product`
- `Order`
- `Review`



rating

Typische Abfragemuster:

- Bestellung anzeigen (inkl. Artikel und Adresse)
- Bestellungen eines Users auflisten
- Produktdetails mit Reviews anzeigen
- Neueste Produkte anzeigen

Designentscheidungen: Denormalisierung, Einbettung vs. Referenzierung

Denormalisierung

- Redundante Kopien (z.B. Produktnamen in Orders, Benutzername in Reviews), um Lesezugriffe ohne zusätzliche Joins zu ermöglichen.
- **Nachteil**: Änderungen (z.B. Produktname) müssen an mehreren Stellen synchronisiert werden.

Einbettung (Embedding) vs. Referenzierung (Linking)

Einbettung: Verknüpfte Daten direkt im Elterndokument speichern.

Vorteile	Nachteile
Alles in einem Dokument abrufbar (z.B. Order + Items)	Dokumente können sehr gross werden
Atomare Updates des Aggregats möglich	Eingebettete Objekte nur zusammen bearbeitbar
Ideal für „one-to-few“ Beziehungen (z.B. Adresse eines Users)	

Referenzierung: Separate Collections, nur IDs oder Zähler im Hauptdokument.

Vorteile	Nachteile
Kleine Hauptdokumente, übersichtlicher	Mehrere Abfragen für zusammengesetzte Daten
Unabhängige Bearbeitung der Dokumente	Application-Level-Joins nötig bzw. $lookup in MongoDB

Gut für „one-to-many“ mit vielen Elementen (z.B. Reviews)	

Entscheidungsfindung

- **„One-to-few“**: Einbettung (z.B. Adressen in `User`).
- **„One-to-many/one-to-squillions“**: Referenzierung (z.B. viele Reviews).
- **Häufigkeit des Zugriffs**: Gemeinsamer Zugriff → Einbettung; getrennte Nutzung → Referenzierung.
- **Datenkonsistenz**: Denormalisierte Felder erfordern Transaktionen oder Background Jobs für Synchronisation.

Beispielhaftes Datenmodell (JSON-Format)

users.json

```
[
  {
    "_id": "user001",
    "name": "Alice Muller",
    "email": "alice@example.com",
    "join_date": "2024-01-10T08:00:00Z",
    "addresses": [
      {
        "type": "home",
        "street": "Musterstrasse 1",
        "city": "Zurich",
        "postal_code": "8000",
        "country": "Schweiz"
      }
    ]
  },
  {
    "_id": "user002",
    "name": "Bob Schmid",
    "email": "bob@example.com",
    "join_date": "2024-02-05T11:15:00Z",
    "addresses": []
  }
]
```

products.json

```
[
```

```
  {
    "_id": "prodA1",
    "name": "Wireless Maus",
    "description": "Ergonomische kabellose Maus",
    "price": 29.99,
    "categories": ["Elektronik","Zubehoer"],
    "stock": 150,
    "created_at": "2024-05-01T09:00:00Z",
    "rating_average": 4.5
  },
  {
    "_id": "prodB2",
    "name": "USB-C Kabel",
    "description": "Schnelles Ladekabel, 1.5m",
    "price": 9.99,
    "categories": ["Elektronik","Kabel"],
    "stock": 500,
    "created_at": "2024-04-20T14:30:00Z",
    "rating_average": 4.2
  }
]
```

orders.json

```
[
  {
    "_id": "order1001",
    "user_id": "user001",
    "order_date": "2024-06-10T14:30:00Z",
    "items": [
      {
        "product_id": "prodA1",
        "product_name": "Wireless Maus",
        "quantity": 2,
        "price": 29.99
      },
      {
        "product_id": "prodB2",
        "product_name": "USB-C Kabel",
        "quantity": 1,
        "price": 9.99
      }
    ],
    "total_amount": 69.97,
    "shipping_address": {
      "street": "Examplestrasse 5",
      "city": "Bern",
      "postal_code": "3000",
      "country": "Schweiz"
```

```
    },
    "status": "shipped"
  }
]
```

reviews.json

```
[
  {
    "_id": "rev500",
    "product_id": "prodA1",
    "user_id": "user002",
    "user_name": "Bob Schmid",
    "rating": 5,
    "comment": "Sehr gute Maus, liegt gut in der Hand.",
    "review_date": "2024-06-12T10:15:00Z"
  }
]
```

Bewertung der Auswirkungen des Datenmodells auf typische Abfragemuster und Skalierbarkeit

Abfragemuster

1. **Bestellung anzeigen**

- Lade das Dokument aus `orders` per `_id`.
- Artikeldetails (`product_name`) sind denormalisiert und direkt verfügbar.
- Versandadresse als eingebettetes Objekt.
- **Performance**: Eine Abfrage, keine Joins nötig.

2. **Bestellungen eines Users**

- Suche in `orders` nach `user_id`. (Index auf `orders.user_id`)
- **Performance**: Einfache, schnelle Abfrage.

3. **Produktdetails mit Reviews**

- Lade `product` aus `products` nach `_id`.
- Suche in `reviews` nach `product_id`. (Index auf `reviews.product_id`)
- `user_name` in den Reviews ist bereits denormalisiert.
- **Performance**: Zwei indizierte Abfragen, verhält sich besser als Joins in relationaler DB.

4. **Neueste Produkte anzeigen**

- Suche in `products`, sortiere nach `created_at` absteigend, limitiere auf N.
- **Performance**: Sehr schnell.

5. **Neues Review hinzufügen**

- Insert in `reviews`.
- Optional: Aktualisierung von `rating_average` in `products`.
- **Performance**: Schreiboperation plus minimaler weiterer Aufwand.

Skalierbarkeit

- **Horizontale Skalierung**: Sharding pro Collection.
- **Dokumentgrössen**: Beherrschbar durch begrenzte Item-Anzahl in Orders.
- **Lese-Skalierbarkeit**: Denormalisierung & Embedding optimieren.
- **Schreib-Skalierbarkeit**: Einfach, Änderungen an denormalisierten Feldern erfordern Synchronisation.
- **Trade-Off**: Denormalisierte Daten vs. Synchronisationsstrategien (Transaktionen, Background Jobs).