Capitolo 3: Il Catalogo Film e l'Ottimizzazione MongoDB

La gestione del catalogo cinematografico in CineMatch è stata progettata seguendo le best practices del **Document-Oriented Design**, privilegiando la velocità di lettura rispetto alla normalizzazione dei dati tipica dei database relazionali (RDBMS).

In questo capitolo analizziamo la struttura della collezione `movies_catalog` e le motivazioni architettoniche dietro la scelta di un modello dati denormalizzato.

3.1 Struttura della Collezione `movies_catalog`

Il catalogo non è distribuito su tabelle multiple (es. `Movies`, `Genres`, `Actors`), ma è consolidato in un'unica collezione dove ogni documento rappresenta un film completo.

Schema JSON Esemplificativo

```
{
  "_id": ObjectId("..."),
  "title": "Inception",
  "year": 2010,
  "genres": ["Action", "Sci-Fi", "Thriller"],  // Array incorporato (One-to-Many)
  "actors": "Leonardo DiCaprio, Joseph Gordon-Levitt, Elliot Page", // Stringa denormalizzata
  "director": "Christopher Nolan",
  "avg_vote": 8.3,
  "description": "Dom Cobb is a skilled thief...",
  "poster_url": "https://image.tmdb.org/...",
  "normalized_title": "inception"  // Campo calcolato per ricerche veloci
}
```

3.2 La Scelta "One-to-Many" Incorporata (Embedding)

La caratteristica distintiva di questo schema è l'uso del pattern **Embedded Data** per gestire le relazioni "Uno-a-Molti" (One-to-Many).

Il Problema nei Database Relazionali

In un DB SQL (es. PostgreSQL), per mostrare la scheda di un film dovremmo eseguire JOIN costose:

1. `SELECT * FROM movies WHERE id=1`
2. `JOIN movies_genres ON ... JOIN genres ON ...`
3. `JOIN movies_actors ON ... JOIN actors ON ...`

Questo approccio aumenta la latenza, specialmente sotto carico elevato, poiché il database deve incrociare indici di tabelle diverse.

La Soluzione MongoDB: Embedding

In CineMatch, abbiamo scelto di **incorporare** i dati relazionali direttamente nel documento padre.

- **Relazione Film-Generi**: Gestita con un semplice array di stringhe `["Action", "Sci-Fi"]` all'interno del film.
- **Vantaggio Prestazionale**: Per recuperare il film E i suoi generi, MongoDB esegue **una singola operazione di lettura** su disco (Single IOPS). Non c'è alcun bisogno di `$lookup` (la "JOIN" di Mongo), rendendo le query estremamente rapide ed efficienti per la visualizzazione frontend.

3.3 Gestione delle Query Complesse

Nonostante la denormalizzazione, il sistema mantiene una capacità di query sofisticata grazie a indici mirati:

1. **Multikey Index sui Generi**:
   Definendo un indice su `genres`, MongoDB "srotola" l'array nell'indice. Una query come `db.movies_catalog.find({ "genres": "Sci-Fi" })` è istantanea perché punta direttamente ai documenti che contengono quel valore nell'array.
2. **Text Index per la Ricerca**:
   Abbiamo creato un indice testuale composto sui campi `title`, `original_title` e `director`. Questo permette la "Full Text Search" (FTS) su tutto il catalogo con una sola condizione di ricerca.

3.4 Eccezioni: Quando Usare le Referenze?

Non tutto è incorporato. Per entità che crescono indefinitamente (Unbounded Arrays), manteniamo collezioni separate:

- **Commenti/Eventi Streaming**: I commenti di YouTube o gli eventi di tracking NON sono nel documento del film, poiché potrebbero essere migliaia. Sono gestiti nelle collezioni `CommentiLive` o `user_stats` e collegati via ID o logica applicativa.

3.5 Riepilogo Vantaggi

Caratteristica	Approccio Relazionale	Approccio CineMatch (Embedded)	Vantaggio
Recupero Dati	Multiple Query / JOIN	1 Query	Bassa Latenza (Microsecondi)
Aggiornamento	Atomico su singola tabella	Richiede update del documento	Ottimizzato per Read-Heavy Workload
Semplicità	Schema rigido	Schema flessibile JSON	Mapping 1:1 con oggetti applicativi