SoccerDBDocumentation

Alessandro Clericuzio

June 2025

1. Introduzione

In questo progetto ho voluto realizzare una soluzione per la gestione e l'analisi di dati relativi ai campionati di calcio, utilizzando un database NoSQL, nello specifico MongoDB. L'obiettivo è stato esplorare concretamente come applicare i principi dei database NoSQL, in particolare le proprietà BASE e il teorema CAP, e implementare un'applicazione web con backend e frontend integrati che consenta operazioni CRUD e query più complesse, come le join tra collection.

2. Descrizione del Miniworld

Il miniworld su cui mi sono concentrato è costituito da dati sportivi reali, estratti da un dataset pubblico presente su Kaggle. Ho organizzato le informazioni in entità principali, facilmente riconoscibili, ognuna rappresentata da una collection in MongoDB:

Countries

- **id**: Identificatore univoco del paese.
- **name**: Nome completo del paese.

Leagues

- **id**: Identificatore univoco della lega.
- **country_id**: Collegamento al paese in cui si svolge la lega.
- **name**: Nome ufficiale della lega (es. Premier League).

Teams

- **id**: Identificatore interno della squadra (chiave primaria).
- **team_api_id**: ID della squadra usato nelle altre tabelle per collegamenti.

- **team_fifa_api_id**: ID usato per collegamenti con altri dataset FIFA.
- **team_long_name**: Nome completo della squadra (es. Manchester United).
- **team_short_name**: Nome abbreviato o sigla della squadra.

Matches

- **id**: Identificatore univoco della partita.
- **match_api_id**: ID usato per API o riferimenti incrociati.
- **country_id**: Collegamento al paese in cui si è giocata la partita.
- **league_id**: Collegamento al campionato a cui appartiene la partita.
- **season**: Stagione di riferimento (es. 2015/2016).
- **stage**: Fase del campionato (es. giornata 1, giornata 2, ecc.).
- **date**: Data in cui si è giocata la partita.
- **home_team_api_id**: ID della squadra di casa.
- **away_team_api_id**: ID della squadra ospite.
- **home_team_goal**: Numero di gol segnati dalla squadra di casa.
- **away_team_goal**: Numero di gol segnati dalla squadra ospite.
- **B365H**: Quota di vittoria della squadra di casa (Bet365).
- **B365D**: Quota per il pareggio (Bet365).
- **B365A**: Quota di vittoria della squadra ospite (Bet365).
- **BWH**: Quota di vittoria della squadra di casa (BWin).
- **BWD**: Quota per il pareggio (BWin).
- **BWA**: Quota di vittoria della squadra ospite (BWin).
- **IWH**: Quota di vittoria della squadra di casa (Interwetten).
- **IWD**: Quota per il pareggio (Interwetten).
- **IWA**: Quota di vittoria della squadra ospite (Interwetten).

Queste entità sono strettamente correlate: ad esempio, ogni partita è associata a due squadre tramite ID, mentre le squadre sono collegate ai paesi di appartenenza. Questo rappresenta un mini-mondo realistico e abbastanza complesso, perfetto per testare modelli NoSQL.

3. Contesto Applicativo

L'applicazione si inserisce nel contesto della gestione e analisi dati sportivi, con potenziali utilizzi per statistica, monitoraggio performance e visualizzazione storica.

4. Soluzione Proposta

4.1 Scelta Tecnologica

Ho optato per MongoDB come database NoSQL, per la sua efficacia nella gestione di dati gerarchici e la facile integrazione con tecnologie moderne come Node.js e React.

4.2 Struttura delle Collection e Operazioni CRUD

Nel database ho creato quattro collection principali, ciascuna derivata da una delle entità del dataset. Per ognuna ho incluso i campi essenziali per le funzionalità dell'applicazione:

- **teams:** contiene informazioni sulle squadre di calcio.
  - `id`: Identificatore interno della squadra (chiave primaria).
  - `team_api_id`: ID usato per join con altre tabelle.
  - `team_fifa_api_id`: ID alternativo per collegamenti FIFA.
  - `team_long_name`: Nome esteso della squadra.
  - `team_short_name`: Nome breve o sigla della squadra.
- **matches:** rappresenta le partite giocate tra le squadre.
  - `id`, `match_api_id`: Identificatori univoci della partita.
  - `country_id`, `league_id`: Riferimenti a paese e campionato.
  - `season`, `stage`, `date`: Dati temporali della partita.
  - `home_team_api_id`, `away_team_api_id`: Squadre partecipanti.
  - `home_team_goal`, `away_team_goal`: Gol segnati.
  - `B365H`, `B365D`, `B365A`: Quote scommesse (Bet365).
  - `BWH`, `BWD`, `BWA`: Quote scommesse (BWin).
  - `IWH`, `IWD`, `IWA`: Quote scommesse (Interwetten).
- **leagues:** raccoglie i dettagli delle competizioni ufficiali.
  - `id`: Identificatore della lega.
  - `country_id`: Collegamento al paese in cui si gioca.
  - `name`: Nome del campionato.

- **countries:** contiene i dati geografici.
  - `id`: Identificatore del paese.
  - `name`: Nome completo del paese.

Per la collection `leagues`, in particolare, ho implementato tutte le operazioni CRUD: creazione, lettura, aggiornamento e cancellazione, permettendo così una gestione completa delle leghe.

4.3 Implementazione delle JOIN

MongoDB non supporta le join relazionali come i database SQL, ma grazie alla pipeline di aggregazione e all'operatore `$lookup` ho simulato le join tra collection. Ad esempio, per mostrare una partita con i dettagli delle squadre coinvolte, ho scritto una query di aggregazione che combina i dati di `matches` con quelli di `teams` per le squadre di casa e ospiti.

4.4 Proprietà BASE e Teorema CAP: come le ho applicate nel progetto

Nel progettare il sistema, ho voluto rispettare le proprietà BASE (Basically Available, Soft state, Eventual consistency) e tenere in considerazione il teorema CAP (Consistency, Availability, Partition tolerance), facendo scelte ben bilanciate a seconda dei casi.

- **Basically Available:** Ho progettato il sistema tenendo a mente l'obiettivo di garantire la disponibilità delle informazioni, anche in scenari di carico o errori parziali. La disponibilità è stata comunque assicurata tramite un'architettura leggera, con API sempre raggiungibili e un frontend reattivo.
- **Soft state:** La replicazione asincrona significa che lo stato del database cambia nel tempo, anche senza nuove scritture dirette. Ad esempio, subito dopo una modifica a una lega, alcune richieste potrebbero ancora vedere i dati vecchi, finché la replica non si aggiorna. Ho accettato una temporanea perdita di **Consistency (C)** per mantenere la disponibilità.
- **Eventual consistency:** Le modifiche si propagano nel tempo. Per esempio, dopo aver aggiornato una partita, la consistenza si raggiunge in modo eventuale. Questo è un compromesso che, nel contesto sportivo, è accettabile perché non si richiede aggiornamento in tempo reale millisecondo per millisecondo.
- **Partition tolerance:** Il sistema è progettato per continuare a funzionare anche in presenza di partizioni di rete tra nodi MongoDB. Se un nodo si disconnette, gli altri continuano a servire i dati.

- **Availability:** Ho scelto di privilegiare la disponibilità: le API e il frontend rispondono sempre, anche in condizioni di degrado.
- **Consistency:** La consistenza forte è parzialmente sacrificata per garantire disponibilità e tolleranza alle partizioni.

Esempi pratici

- Quando un utente aggiorna una lega, la modifica è visibile con un leggero ritardo dovuto alla replica asincrona. Qui la disponibilità è più importante della consistenza immediata.
- La sidebar con le nazioni è sempre disponibile, anche se qualche nodo replica è offline.
- Le query di join tra partite e squadre vengono eseguite usando `$lookup` in MongoDB, che funziona bene anche con dati replicati e distribuiti.

5. Metodologia utilizzata per sviluppare la soluzione

5.1 Analisi del dataset

Ho iniziato dal dataset CSV originale, che ho convertito in JSON per facilitarne l'importazione in MongoDB. Ho effettuato una pulizia dati per uniformare i campi e rimuovere incongruenze.

5.2 Modellazione NoSQL

Ho individuato le collection chiave, bilanciando la normalizzazione per ridurre ridondanze con la denormalizzazione per migliorare le performance delle query.

5.3 Sviluppo backend

Il backend è stato realizzato in Node.js, esponendo API REST per gestire le operazioni CRUD e le query di aggregazione. Ho curato endpoint per squadre, partite, campionati e paesi.

5.4 Sviluppo frontend

Il frontend, basato su React e Next.js, offre un'interfaccia semplice, con visualizzazione dati aggregati, inserimento e aggiornamento tramite form e filtri di ricerca.

5.5 Testing e validazione

Ho testato tutte le API e il frontend per verificare correttezza e coerenza delle operazioni CRUD e delle join, oltre a valutare il rispetto delle proprietà BASE e la gestione delle partizioni.

6. Conclusioni

Questo progetto mi ha permesso di mettere in pratica l'uso di database NoSQL su un dataset reale, affrontando le sfide di consistenza e scalabilità. Ho dimostrato come si possa implementare una soluzione efficace, bilanciando disponibilità e consistenza, e offrendo un'interfaccia utente funzionale e reattiva per la gestione dei dati dei campionati di calcio.