UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea
in Ingegneria Informatica

Relazione Finale
Titolo 1
Titolo 2
Titolo 3

Relatore: Chiar.mo Prof. Melchiori Michele

Laureando:
Rizzo Matteo
Matricola n. 727499

Anno Accademico 2021/2022

Sommario

Ringraziamenti

Indice

Introduzione

Analisi database noSQL chiave-valore, in particolare *REDIS* progettazione concettuale/logica database chiave-valore

Capitolo 1

DataBase NoSQL

I database *NoSQL*, che sta per *not only SQL*, sono database non tabellari che archiviano i dati in maniera completamente differente dai classici relazionali. Le caratteristiche principali sono la progettazione specifica per carichi elevati e il supporto nativo per la scalabilitá orizzontale, la tolleranza agli errori e la memorizzazione dei dati in modo denormalizzato. Infatti ogni elemento viene archiviato singolarmente con una chiave univoca, e la coerenza dei dati non viene garantita. Questa impostazione fornisce un approccio molto piú flessibile alla memorizzazione dei dati rispetto a un database relazionale, un controllo migliore e una maggiore semplicitá nelle applicazioni.

1.1 Perche é nato *NoSQL*?

A partire dagli anni 2000 si é passati da un modello in cui le persone principali dell'IT erano sistemisti ad un modello in cui le persone principali sono diventate gli sviluppatori. Tale passaggio ha comportato la nascita di database NoSQL che sono fortemente orientati agli sviluppatori ed allo sviluppo Agile. Inoltre i dati si sono trasformati passando dai classici strutturati a dati non strutturati (di differenti dimensioni, semistrutturati, polimorfici..) che non permettevano di definire un modello relazionale organico e cosí i database NoSQL sono diventati estremamente popolari perché permettono di lavorare principalmente con dati non strutturati anche di enormi dimensioni.

1.2 Perché dovrei utilizzare un database *NoSQL*?

I database NoSQL sono una soluzione ideale per molte applicazioni moderne, quali dispositivi mobili, Web e videogiochi che richiedono strutture dati flessibili, scalabili, con prestazioni elevate ed altamente funzionali.

- `Flessibilitá`: vengono offerti schemi flessibile che consentono uno sviluppo piú veloce. Quindi é una soluzione ideale per i dati semi-strutturati e non strutturati. É possibile arricchire le applicazioni di nuovi dati e informazioni senza dover sottostare ad una rigida struttura dei dati;

- `Scalabilitá`: grazie alla semplicitá vi é la possibilitá di *scalare in orizzontale* in maniera estremamente efficiente. Infatti, si predilige l'utilizzo di cluster con molti nodi distribuiti, rispetto all'utilizzo di server centralizzati. Inoltre, vi é la possibilitá di aggiungere nodi a caldo in maniera completamente trasparente per l'utente finale;

- `Elevate Prestazioni`: grazie alla mancanza di operazioni di aggregazione dei dati("join") ed anche grazie all'introduzione di semplificazioni, come il mancato supporto delle transazioni ACID, si ha una elevata velocitá computazionale.

- `Altamente funzionali`: non vi é piú un linguaggio generale (SQL) come nei database relazionali, ma vi sono API in base al database specifico che si va ad utilizzare.

1.3 Tipologie di *NoSQL*

Tipologie principali di database *NoSQL*:

- `documentali`
- `key-value`
- `colonnari`
- `a grafo`

Capitolo 2

Progettazione Concettuale/Logica

Sebbene i database *NoSQL* vengono definiti *schemaless*, la progettazione dell'organizzazione dei dati richiede di prendere decisioni significative. Infatti, i dati persistenti delle applicazioni hanno un impatto sui principali requisiti di qualitá che devono essere soddisfatti in un'applicazione vera e propria (scalabilitá, prestazioni, coerenza). Il mondo NoSQL é altamente eterogeneo, quindi questa attivitá di progettazione di solito si basa su pratiche e linee guida da seguire in base al sistema selezionato. Peró, sono stati studiati diversi approcci che vogliono generalizzare il problema di progettazione che é alla base di ogni sistema di persistenza.

NoAM é uno dei principali strumenti di modellazione astratto per database NoSQL. Grazie ad esso riusciamo a definire una progettazione che é indipendente dal sistema specifico in cui viene usata. Viene utilizzato un modello dei dati intermedio e astratto, che, a sua volta, viene utilizzato per rappresentare i dati dell'applicazione come raccolte di oggetti aggregati.

2.1 Modellazione NoAM → NoSQL abstract model

La metodologia *NoAM* é composta da:

- Modellazione Concettuale e design degli Aggregati
- Partizionamento degli Aggregati e modellazione NoSQL
- Implementazione

2.1.1 Modellazione Concettuale e design degli Aggregati

Riguarda la vera e propria progettazione del modello di dominio, e comporta l'identificazione delle diverse classi di aggregati necessari in un'applicazione. Sono possibili diversi approcci per identificare classi di aggregati per una particolare applicazione. L'approccio Domain-Driven Design(*DDD*), attraverso il quale viene generato un diagramma UML delle classi, é guidato dai casi d'uso, ovvero dai requisiti funzionali, e da esigenze di scalabilitá e coerenza all'interno dell'aggregato. Si procede nel modo seguente:

- I dati persistenti di un'applicazione sono modellati in termini di entitá, oggetti di valore e relazioni. Un'entitá é un oggetto persistente che ha un'esistenza indipendente ed é caratterizzata da un'identificatore univoco, mentre un oggetto di valore é caratterizzato appunto da un suo valore senza un proprio identificatore

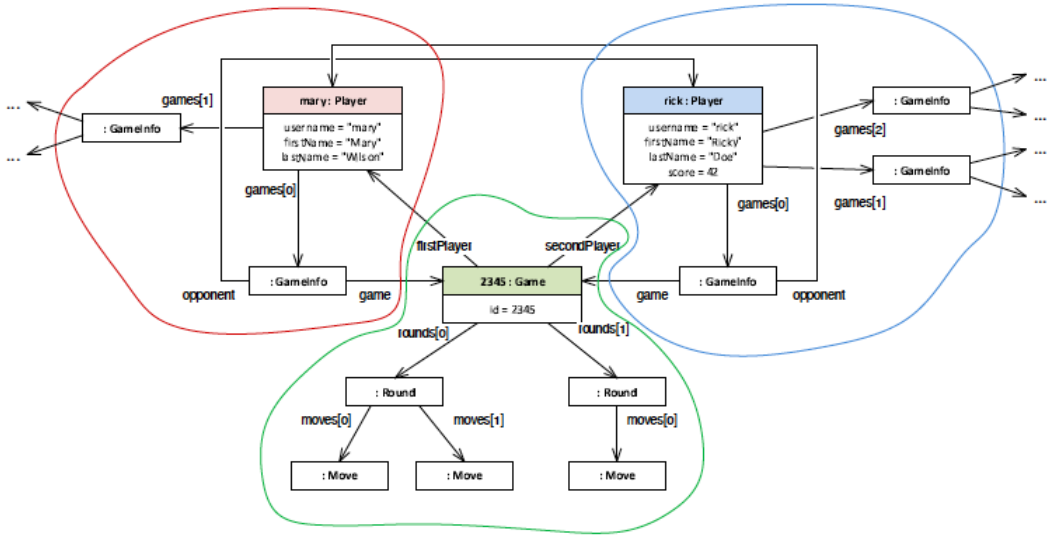
- Entitá e oggetti di valore vengono raggruppati in *aggregati*. Un aggregato ha un'entitá come radice e puó contenere molti oggetti di valore.

A causa delle loro caratteristiche, la progettazione degli aggregati comporta un compromesso per quanto riguarda la loro granularitá. infatti:

- Gli aggregati dovrebbero essere abbastanza grandi per poter includere tutti i dati coinvolti da certi vincoli di integritá.

- Gli aggregati dovrebbero essere i piú piccoli possibile, in quanto dimensioni ridotte consentono di soddisfare requisiti di prestazioni e scalabilitá.

Preso come esempio un dominio in cui vanno salvati in modo persistente dati su giocatori e giochi



Nella figura sopra riportata l'oggetto con lo scomparto superiore colorato é un'entitá, altrimenti é un oggetto di valore. La linea chiusa denota il confine di un aggregato. Pertanto avremo due classi aggregate principali: Player e Game.

2.1.2 Partizionamento degli Aggregati e modellazione NoSQL

In questa fase viene utilizzato *NoAM* come modello intermedio tra gli aggregati e i database NoSQL, quindi potrebbe essere visto come un equivalente della *progettazione concettuale* fatta nei database relazionali. Il modello NoAM → modello di dati astratti, ha il compito di sfruttare i punti in comune dei vari modelli di dati, ma introduce anche astrazioni per bilanciare le variazioni che vi sono tra diversi modelli di NoSQL. Vi sono due nozioni distinte di *unitá* di accesso ai dati:

- `blocco`, unitá di dimensione maggiore, ha massima consistenza
- `entry`, unitá di dimensione minore, che permetto l'accesso ai dati

Con riferimento in particolare ai database chiave-valore una `entry` corrisponde a una coppia chiave-valore, mentre un `blocco` corrisponde a un gruppo di coppie chiave-valore correlate tra loro.

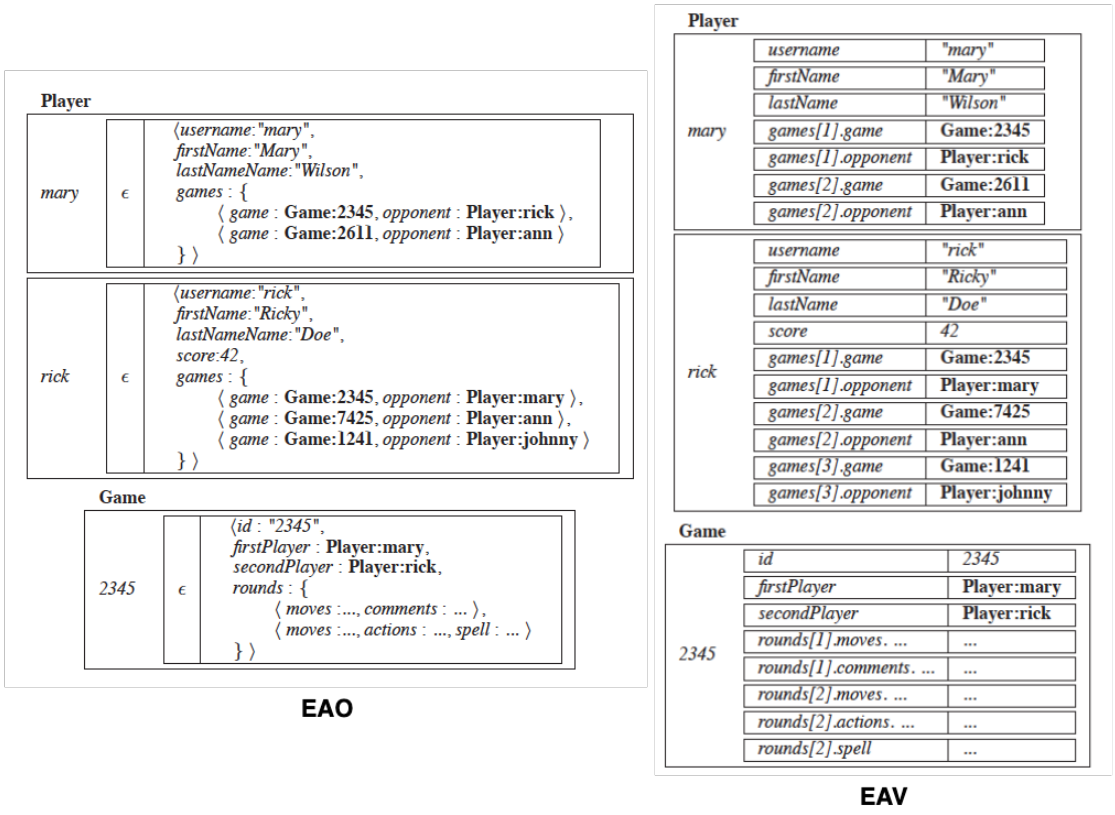
Quindi, si ha che: Un `database` é un insieme di `collections`. Ogni `collection` ha un nome distinto; Una `collection` é un insieme di `blocchi`, ogni `blocco` all'interno della `collection` é identificato da una chiave di blocco, che deve essere

univoca; Un `blocco` é un insieme non vuoto di `entries`, ogni `entry` é composta da una coppia chiave-valore, la quale é univoca all'interno del blocco, il valore puó essere anche complesso.

Per effettuare il passaggio da aggregati a modellazione NoSQL ogni classe di aggregati viene rappresentata da una `collection` ed ogni singolo aggregato viene rappresentato da un `blocco`.

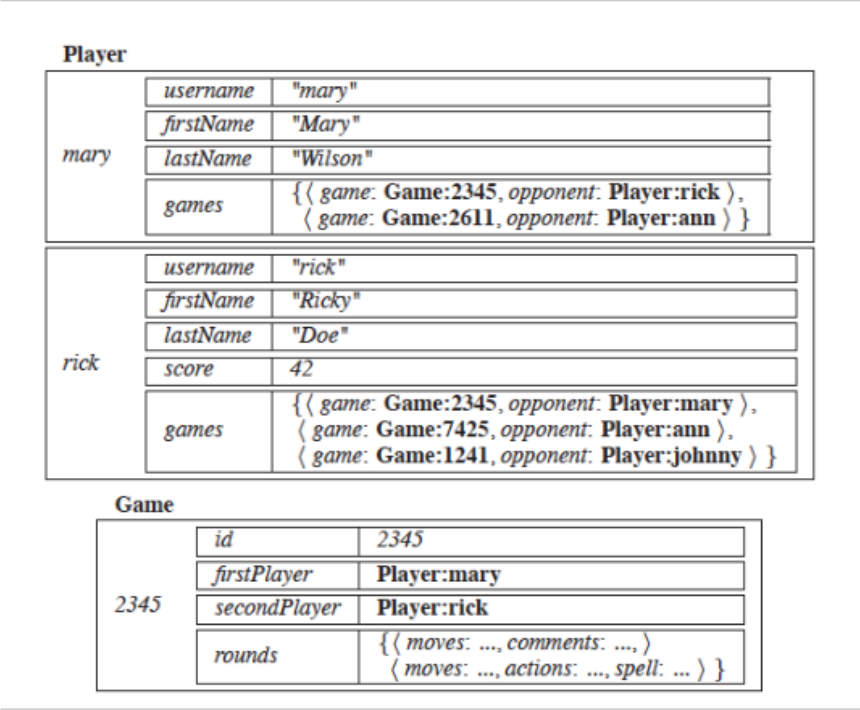
Vi sono due modalitá principali per rappresentare gli aggregati:

- *Entry per Aggregate Object (EAO)*: Rappresenta ogni aggregato utilizzando una singola `entry`, la chiave della `entry` é vuota, il valore contiene l'intero aggregato.
- *Entry per Atomic Value (EAV)*: Rappresenta ogni aggregato per mezzo di piú `entry`, le chiavi di ogni `entry` devono rappresentare il percorso per accedere al valore di un certo componente(quindi possono esserci anche chiavi con nomi strutturati a piú livelli), i valori di ogni `entry` sono atomici.



Nella figura *EAO* si ha per ogni `blocco` *Player* o *Game* una singola `entry` con chiave vuota, valore strutturato in modo complesso
Nella figura *EAV* si ha per ogni `blocco` piú `entry` in modo da avere valori atomici al suo interno

Ovviamente queste due sono le rappresentazioni piú estreme che si possono ottenere. É possibile adottare una strategia di rappresentazione intermedia, denominata *Entry per Top level Field (ETF)*, in cui viene utilizzata una `entry` distinta per ogni campo di livello superiore, quindi si ha una sorta di struttura mista.

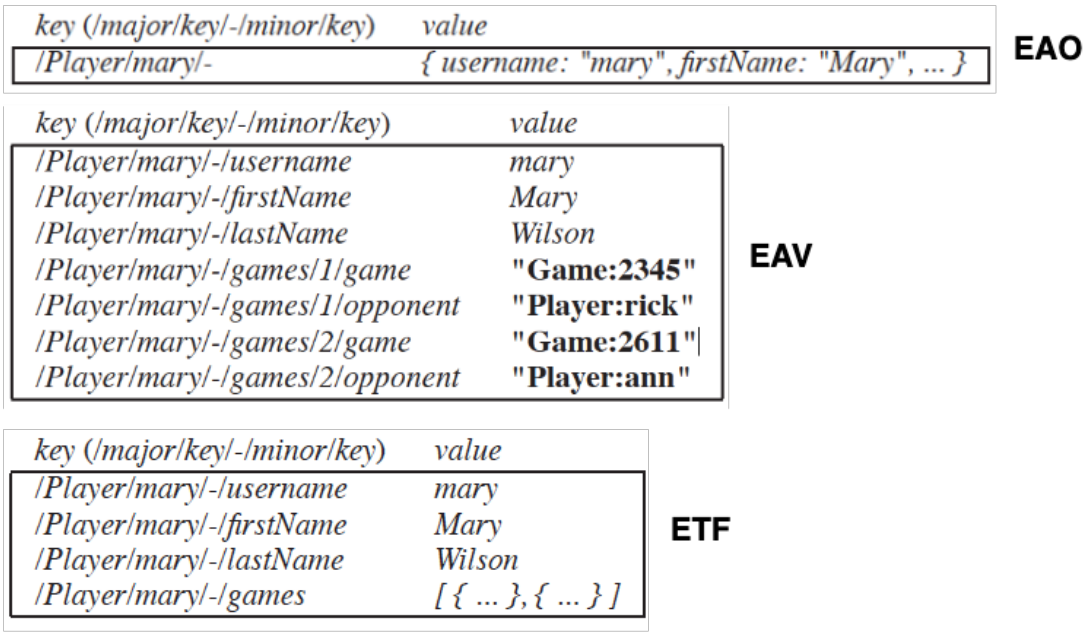


ETF

Se dovessimo aver bisogno di una rappresentazione degli aggregati ancora piú flessibile é possibile effettuarne il *partizionamento*, ovvero si possono raggruppare `entry` che vengono accedute insieme, oppure separare certe `entry` per avere dei valori meno complessi.

2.1.3 Implementazione

Consiste nel tradurre i modelli NoAM ottenuti nella fase precedente in strutture corrette per il database specifico che stiamo utilizzando. Per quanto riguarda i database chiave-valore si utilizzerá una coppia chiave-valore per ogni `entry` ottenuta nella struttura precedente. In base alle scelte di progetto si decide quale modello NoAM utilizzare (EAO/EAV/ETF), praticamente bisogna decidere che livello di complessitá vogliamo avere sulle chiavi e sui valori.



Capitolo 3

Approfondimento Redis

Parlare di Redis in generale vantaggi/svantaggi perché utilizzarlo strutture dati String Map Set Json ...

Capitolo 4

Interrogazioni SQL $\rightarrow$ *Key-Value*

parlare del tipo di interrogazioni che vengono fatte, c'é un vero e proprio linguaggio strutturato come SQL?

Capitolo 5

proprietá ACID

Parlare delle proprietá *ACID* per quanto riguarda questo tipo di database Redis quali di proprietá dispone? É possibile decidere se avere una certa proprietá o meno ...

Capitolo 6

Caso Concreto - IoT sensore Temperatura

implementazione librerie Redis in linguaggi di programmazione Parlare di Jedis, libreria Java. Riportare esempi di codice che implementa DataBase

Bibliografia

Bibliografia