UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Basi di Dati 2

Professoressa: Genoveffa Tortora

Dottor Luigi di Biasi

Event Planner: un Sistema Informativo per un Gruppo di Organizzatori di Eventi Pubblici

AUTORE: Santangelo Angelo

Indice

1 Introduzione **1**

1.1 Descrizione della Realtà d'Interesse 1

1.2 Raccolta ed Analisi dei Requisiti 2

2 Progettazione Concettuale **3**

2.1 Schema EER 3

2.2 Dizionario dell'Entità 4

2.3 Dizionario delle Relazioni 5

2.4 Vincoli non Esprimibili 5

2.5 Insieme Campione di Operazioni 5

3 Progettazione Logica **7**

3.1 Eliminazione delle Gerarchie 7

3.2 Schema EER ristrutturato 8

3.3 Mapping in Schema Logico 8

3.4 Normalizzazione dello Schema Logico 8

3.4.1 Prima Forma Normale (1NF) 9

3.4.2 Seconda Forma Normale (2NF) 9

3.4.3 Terza Forma Normale (3NF) 9

3.4.4 Forma Normale di Boyce-Codd (BCNF) 9

3.4.5 Quarta Forma Normale (4NF) 10
3.4.6 Conclusioni sulla Normalizzazione 10

4 Implementazione **11**
4.1 MongoDB 11
4.2 Tecnologie Utilizzate 12
4.2.1 Front-End 12
4.2.2 Memorizzazione 12
4.2.3 Back-End 12
4.3 Importazione dei Dati in MongoDB 13
4.4 Modellazione NoSQL 13
4.4.1 Scelte di Progettazione 15

5 Conclusioni **16**

CAPITOLO 1

Introduzione

1.1 Descrizione della Realtà d'Interesse

La realtà che andiamo a rappresentare riguarda la gestione di eventi pubblici. Negli ultimi anni, il settore degli eventi e delle serate a tema ha visto una crescita esponenziale, alimentata anche grazie ai social media e alle piattaforme di comunicazione digitale, le quali hanno reso la promozione di eventi più immediata, raggiungendo un pubblico molto ampio in poco tempo. Oggi giorno sempre più persone, e in particolar modo i giovani, sono interessati a partecipare a tali eventi. I proprietari dei locali, insieme agli organizzatori, hanno reso possibile la diffusione di queste serate di divertimento, infatti, ad oggi è facile trovare serate con temi di ogni tipo, come ad esempio "schiuma party", "color party" o anche feste organizzate nel periodo natalizio, nel periodo di Halloween, con relativi temi. Gli organizzatori di eventi affrontano numerose sfide, tra cui la gestione delle prenotazioni, la gestione finanziaria, la gestione artistica, ecc. Un gestionale ben progettato può aiutare a snellire questi processi, riducendo il rischio di errori e migliorando l'efficienza complessiva dell'organizzazione.

1.2 Raccolta ed Analisi dei Requisiti

Si vuole progettare una base di dati per la gestione degli eventi. Di ogni evento, identificato con un codice, vogliamo memorizzare il tema, la data di svolgimento, il prezzo del biglietto e il nome del locale. Quest'ultimi vengono identificati tramite un codice e, inoltre, si vuole memorizzare il nome, il cognome e il numero di telefono, eventualmente per tenersi in contatto. In un Evento possono esserci più partecipanti e un Partecipante può presenziare in più eventi. Le varie serate sono condotte da uno o più artisti che verranno identificati tramite un codice. Un'artista, invece, può condurre più serate. Di tali artisti si vuole memorizzare il nome d'arte, il prezzo richiesto per la conduzione dell'evento e il numero di telefono. Gli artisti possono essere di vario tipo: DJ, Speaker, Cantante, Fotografo.

CAPITOLO 2

Progettazione Concettuale

2.1 Schema EER

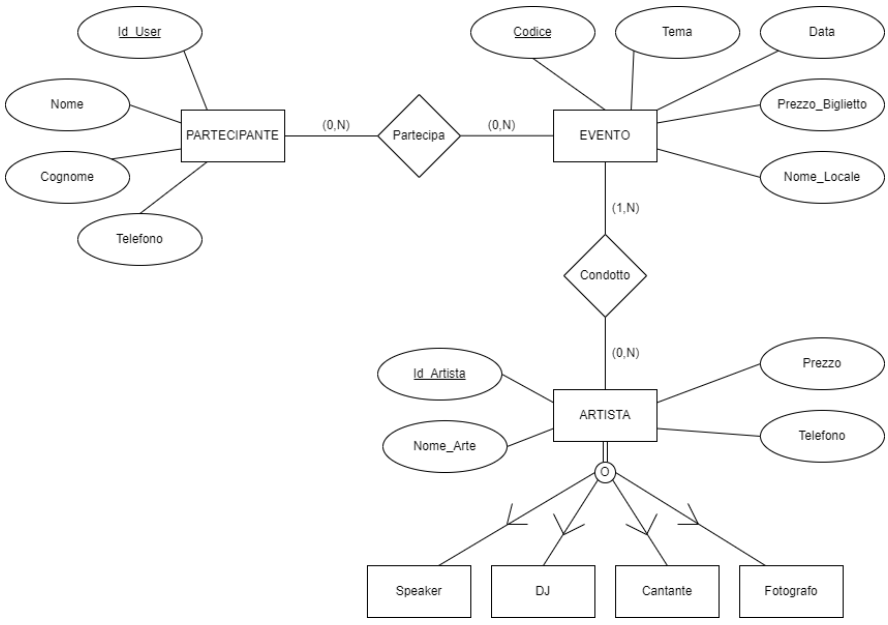

Figura 2.1: Schema EER della Realtà d'Interesse

2.2 Dizionario dell'Entità

Entità	Attributi	Identificatori
Partecipante	• Id_User • Nome • Cognome • Telefono	Id_User
Evento	• Codice • Tema • Data • Prezzo_Biglietto • Nome_Locale	Codice
Artista	• Id_Artista • Nome_Arte • Prezzo • Telefono	Id_Artista
Speaker	/	/
DJ	/	/
Cantante	/	/
Fotografo	/	/

2.3 Dizionario delle Relazioni

Relazione	Descrizione	Entità Coinvolte	Attributi
Partecipa	Un partecipante partecipa ad un evento	Partecipante (0,N) Evento (0,N)	/
Condotta	Un evento è condotto da artisti	Evento (1,N) Artista (0,N)	/

Tabella 2.1: Dizionario delle Relazioni

2.4 Vincoli non Esprimibili

Oltre ciò che è deducibile dallo schema EER, si tenga conto dei seguenti vincoli:

- Si assume che il prezzo del biglietto sia unico per tutti i partecipanti.
- Si assume che il prezzo richiesto dall'artista sia lo stesso per ogni evento.

2.5 Insieme Campione di Operazioni

- **Operazione 1**: Recupera l'elenco di tutti gli eventi con le relative informazioni. Inoltre, per ogni evento, verranno visualizzati gli artisti che hanno condotto tale evento e la somma che hanno guadagnato.
- **Operazione 2**: Inserisci un nuovo evento.
- **Operazione 3**: Recupera l'elenco di tutti gli eventi con le relative spese totali dovute al pagamento degli artisti.
- **Operazione 4**: Recupera l'elenco di tutti gli eventi con il relativo numero di partecipanti. Inoltre, per ogni evento, verrà visualizzato anche l'incasso totale dai biglietti acquistati.
- **Operazione 5**: Recupera l'elenco di tutti gli eventi effettuati da una data X ad una data Y inserita dall'utente.

- **Operazione 6**: Recupera l'elenco di tutti gli artisti con un budget al di sotto di un prezzo inserito dall'utente.
- **Operazione 7**: Elimina un evento. L'utente deve inserire l'ID esatto dell'evento.
- **Operazione 8**: Recupera l'elenco di tutti gli eventi in cui ha partecipato un artista specifico. L'utente deve inserire il nome d'arte completo dell'artista.
- **Operazione 9**: Recupera l'elenco di tutti gli eventi che si sono svolti in un determinato locale. L'utente deve inserire il nome del locale.
- **Operazione 10**: Recupera l'elenco di tutti gli eventi con le relative informazioni. Per ogni evento verrà calcolato la differenza tra l'incasso totale e il budget speso per gli artisti in modo da verificare il guadagno veritiero.

Si procede con la ristrutturazione dello schema concettuale.

CAPITOLO 3

Progettazione Logica

3.1 Eliminazione delle Gerarchie

L'unica gerarchia presente nel nostro schema è quella tra l'entità **ARTISTA** e le entità figlie **DJ**, **Speaker**, **Fotografo** e **Cantante**. Abbiamo deciso di adottare la tecnica del "Portare le figlie al padre" per semplificare il modello.
Questa scelta è motivata dal fatto che le entità figlie non posseggono attributi propri aggiuntivi che le distinguano in modo significativo l'una dall'altra, né hanno relazioni specifiche con altre entità che non siano già condivise con l'entità padre **ARTISTA**. Pertanto, non forniscono informazioni aggiuntive utili allo schema EER.
Per rappresentare questa gerarchia, verrà introdotto un nuovo attributo chiamato **TIPO** all'interno dell'entità **ARTISTA**. Questo attributo sarà utilizzato per indicare la tipologia dell'artista, specificando se si tratta di un **DJ**, **Speaker**, **Fotografo** o **Cantante**.
Questa soluzione non solo semplifica il nostro schema EER, ma permette anche una gestione più efficiente dei dati, riducendo la complessità delle query necessarie per estrarre informazioni sugli artisti. Inoltre, facilita eventuali modifiche future, come l'aggiunta di nuove categorie di artisti, rendendo il modello più flessibile e scalabile. Basterà semplicemente aggiungere nuovi valori possibili per l'attributo **TIPO** senza

necessità di alterare la struttura del database o introdurre nuove entità.

3.2 Schema EER ristrutturato

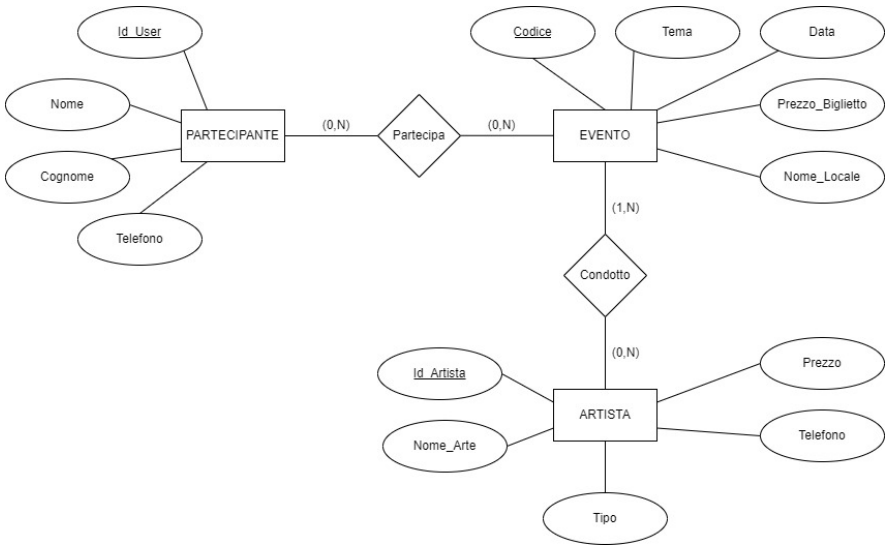

Figura 3.1: Schema EER Ristrutturato

3.3 Mapping in Schema Logico

- **Partecipante**(Id_User, Nome, Cognome, Telefono);
- **Evento**(Codice, Tema, Data, Prezzo_Biglietto, Nome_Locale);
- **Artista**(Id_Artista, Nome_Arte, Tipo, Prezzo, Telefono);
- **Partecipa**(Partecipante.Id_User↑, Evento.Codice↑);
- **Condotto**(Evento.Codice↑, Artista.Id_Artista↑)

3.4 Normalizzazione dello Schema Logico

Lo schema logico è stato normalizzato fino alla forma normale più alta possibile, ovvero la 4NF. Di seguito, esaminiamo dettagliatamente il processo e le motivazioni.

3.4.1 Prima Forma Normale (1NF)

Uno schema è in Prima Forma Normale (1NF) se tutti i campi contengono solo valori atomici, ovvero valori indivisibili. Il nostro schema soddisfa questa condizione poiché:

- Non sono presenti attributi che contengono liste o insiemi di valori.
- Ogni attributo contiene solo valori singoli e indivisibili.

3.4.2 Seconda Forma Normale (2NF)

Uno schema è in Seconda Forma Normale (2NF) se è in 1NF e ogni attributo non chiave è completamente dipendente dalla chiave primaria. Il nostro schema soddisfa questa condizione perché:

- È già in 1NF.
- Tutti gli attributi non chiave dipendono interamente dalla chiave primaria e non da una sua parte.

3.4.3 Terza Forma Normale (3NF)

Uno schema è in Terza Forma Normale (3NF) se è in 2NF e nessun attributo non chiave dipende transitivamente dalla chiave primaria. Il nostro schema soddisfa questa condizione perché:

- È già in 2NF.
- Non esistono dipendenze funzionali transitive tra gli attributi non chiave e la chiave primaria. Ogni attributo non chiave dipende direttamente dalla chiave primaria.

3.4.4 Forma Normale di Boyce-Codd (BCNF)

Uno schema è in Forma Normale di Boyce-Codd (BCNF) se per ogni dipendenza funzionale $X \rightarrow Y$, X è una superchiave. Il nostro schema soddisfa questa condizione perché:

- È già in 3NF.
- Ogni determinante di una dipendenza funzionale è una superchiave.

3.4.5 Quarta Forma Normale (4NF)

Uno schema è in Quarta Forma Normale (4NF) se, essendo in BCNF, non esistono dipendenze multivalore non banali. Il nostro schema soddisfa questa condizione perché:

- È già in BCNF.
- Non esistono attributi multivalore che dipendono dalla chiave primaria.

3.4.6 Conclusioni sulla Normalizzazione

La normalizzazione fino alla 4NF garantisce che il nostro schema:

- Riduca al minimo la ridondanza dei dati.
- Eviti anomalie di inserimento, aggiornamento e cancellazione.
- Mantenga la coerenza e l'integrità dei dati.

Questo processo di normalizzazione è essenziale per garantire un database efficiente e affidabile, migliorando le prestazioni delle query e facilitando la manutenzione dei dati nel tempo.

Si procede con la descrizione della fase di Implementazione.

CAPITOLO 4

Implementazione

Si implementi l'applicativo per effettuare i vari tipi di query, da quelle elementari, alle operazioni più complesse. Si supponga di voler convertire lo schema appena normalizzato in un modello non relazionale per prepararlo all'implementazione su un DBMS NoSQL, quale **MongoDB**.

4.1 MongoDB

MongoDB è un database NoSQL orientato ai documenti, progettato per gestire grandi quantità di dati non strutturati o semi-strutturati. Caratteristiche principali:

- **Modello di Dati**: utilizza documenti in formato BSON (Binary JSON), che consente di memorizzare facilmente dati complessi e gerarchici rispetto ai tradizionali database relazionali. I documenti sono simili a JSON e possono contenere vari tipi di dati, come stringhe, numeri, date, e persino array e altri documenti nidificati;
- **Scalabilità**: è progettato per essere scalabile in modo orizzontale, il che significa che può gestire grandi volumi di dati distribuendo il carico su più server.

Questo lo rende adatto per applicazioni che devono crescere rapidamente dal punto di vista del traffico;

- **Alta Disponibilità**: supporta la replica dei dati attraverso set di repliche, garantendo che i dati siano disponibili anche in caso di guasti hardware o di rete;
- **Schema-Less**: i dati non possiedono una struttura ben precisa, il che consente di modificarla facilmente per adattarsi ai cambiamenti e alle esigenze dell'applicazione;
- **Potenti Capacità di Query**: supporta una varietà di operazioni di query, tra cui filtri, ordinamenti, aggregazioni complesse, ecc. MongoDB offre anche un linguaggio di query potente e flessibile per estrarre dati in modo efficiente.

4.2 Tecnologie Utilizzate

4.2.1 Front-End

Il front-end dell'applicazione è stato progettato utilizzando **HTML**, **CSS** e **JavaScript**, integrando **Bootstrap** per la creazione di un'interfaccia utente responsiva e accattivante. Le pagine web sono state create con l'intento di essere semplici da navigare e di fornire un'interazione fluida con l'utente. Le principali funzionalità del front-end includono:

- Form per l'inserimento e l'eliminazione di eventi e artisti.
- Tabelle dinamiche per la visualizzazione dei dati degli eventi e degli artisti.

4.2.2 Memorizzazione

Per la memorizzazione dei dati è stato utilizzato **MongoDB**, scopo di tale progetto.

4.2.3 Back-End

Il back-end sarà costituito da un server implementato in **Python** utilizzando il framework **Flask**, il quale si tratta di un micro framework utilizzato da Python

per il web. Utilizza un approccio tramite definizione di API sul Web Server, le quali verranno invocate dal Front-End. Spesso tali tencologie vengono utilizzate cooperativamente in stack di progettazione dedicati: Flask + MongoDB, ad esempio, è una scelta comune, per via della presenza di librerie dedicate che consentono di effettuare operazioni CRUD sui dati. Un esempio è dato dalla libreria **PyMongo**, della quale ci si avvarrà per le funzionalità descritte. PyMongo consente di interagire con MongoDB e definire le strutture dei dati persistenti. Le collezioni e i documenti in MongoDB permettono di specificare il tipo dei campi con eventuali vincoli (e.g. unicità, obbligatorietà) oltre che array di documenti per l'annidamento e riferimenti ad altre collezioni. PyMongo facilita la creazione di query complesse attraverso l'uso di funzioni già prestabilite e la connessione con **MongoDB Compass**, uno strumento grafico per MongoDB. Compass fornisce una interfaccia utente grafica per visualizzare e modificare i dati all'interno di un database MongoDB.

4.3 Importazione dei Dati in MongoDB

Per importare i dati in MongoDB, è stato utilizzato il tool online **Mockaroo**, uno strumento che aiuta a generare dataset fittizi coerenti e integri tra loro in vari formati, tra cui CSV e JSON. Nel caso specifico, è stato utilizzato il formato JSON per popolare le tabelle "artista", "evento" e "partecipante" con dati fittizi ma integrati in modo significativo.
Una volta esportati i dati nel formato JSON, è stato sviluppato uno script Python per automatizzare il processo di importazione in MongoDB. Questo script naviga attraverso i file JSON e crea automaticamente le collection su MongoDB, facilitando l'inserimento e la gestione dei dati.
Il codice sorgente dello script è disponibile sul profilo GitHub.

4.4 Modellazione NoSQL

L'approccio generale seguito nella progettazione del database NoSQL è stato quello di scegliere il contenuto di una collection e la struttura di un documento in

funzione delle operazioni di lettura e scrittura più frequenti. Esistono due principali approcci per mappare le relazioni in un database NoSQL:

- **Embedded Documents**: in questo approccio, un documento viene inserito interamente all'interno di un altro documento. Questo metodo è tipicamente utilizzato quando:
  - Le relazioni tra i dati sono di tipo uno-a-molti o molti-a-molti.
  - Le operazioni di retrieve (lettura) da una collection prevedono anche il retrieve dei dati di un'altra collection.
  - Si desidera migliorare le prestazioni delle query eliminando la necessità di join complessi.
  - Si vuole garantire l'atomicità delle operazioni di lettura e scrittura.
- **References**: in questo approccio, solo gli ID dei documenti vengono inseriti in un altro documento, mantenendo i dati separati nelle rispettive collection. Questo metodo è tipicamente utilizzato quando:
  - Le relazioni tra i dati sono di tipo molti-a-molti.
  - Le operazioni di retrieve da una collection non richiedono necessariamente il retrieve immediato dei dati dall'altra collection.
  - Si desidera mantenere la modularità e la normalizzazione dei dati.
  - È importante ridurre la duplicazione dei dati e facilitare l'aggiornamento indipendente dei documenti correlati.

La scelta tra Embedded Documents e References è stata fatta in base ai seguenti criteri:

- **Atomicità delle Operazioni**: Quando è necessario garantire che le operazioni di lettura e scrittura siano atomiche, l'embedding dei documenti offre un vantaggio significativo.
- **Modularità e Scalabilità**: Per mantenere una struttura modulare e scalabile, le references sono più adatte, permettendo di aggiornare singolarmente i documenti correlati senza influenzare altri documenti.

- **Ridondanza e Coerenza dei Dati**: L'uso delle references riduce la ridondanza dei dati e facilita il mantenimento della coerenza tra i dati correlati in diverse collection.

Questi principi di progettazione sono stati applicati per garantire che il database sia ottimizzato sia per le operazioni di lettura e scrittura frequenti che per la manutenzione e la scalabilità a lungo termine.
Si noti infine che gli ID delle relazioni vengono ascritti agli ObjectID utilizzati da MongoDB, pena l'impossibilità di implementare in modo semplice e robusto i riferimenti.
Di Seguito, analizziamo le scelte, di cui sopra, passando al NoSQL.

4.4.1 Scelte di Progettazione

Evento - Condotto - Artista Poiché quasi tutte le operazioni implementate sul database prevedono che il recupero dei dati da un evento implichi anche il recupero dei dati degli artisti, è stata scelta l'opzione degli **Embedded Documents**. Questo approccio consente di ottenere tutti i dettagli degli artisti insieme ai dettagli dell'evento in un'unica operazione di lettura, migliorando così l'efficienza delle query e garantendo l'atomicità delle operazioni.

Evento - Partecipa - Partecipante Poiché nessuna delle operazioni implementate sul database prevede che il recupero dei dati da un evento implichi anche il recupero dei dati completi dei partecipanti, è stata scelta l'opzione delle **References**. Questo approccio permette di mantenere una struttura modulare e scalabile, riducendo la duplicazione dei dati e facilitando l'aggiornamento indipendente dei documenti. Si noti che in futuro potrebbero essere aggiunte nuove operazioni che richiedono il recupero dei dati completi dei partecipanti, nel qual caso l'approccio potrebbe essere rivisto.

CAPITOLO 5

Conclusioni

L'esperimento di modellazione di un miniworld volto a costruire una base di dati relazionale e successivamente mapparla per un'implementazione non relazionale costituisce anche la prima esperienza con i DBMS non relazionali dello scrivente. E' stato estremamente stimolante osservare la modellazione dei dati con un nuovo paradigma e lo è stato ancor di più dover effettuare le scelte e prendere le decisioni più opportune per convertire dati relazionali strutturati per utilizzarli in logica NoSQL. Durante lo sviluppo, abbiamo affrontato sfide significative come la progettazione di schemi dati complessi e l'ottimizzazione delle query per migliorare le prestazioni.
In conclusione, il presente progetto ha permesso di realizzare un sistema web robusto utilizzando tecnologie moderne come Flask, PyMongo e MongoDB. Attraverso l'adozione di un'architettura basata su microservizi e l'uso di riferimenti e documenti embedded per gestire le relazioni tra i dati, si è riusciti a sviluppare un'applicazione scalabile e performante.
Infine, guardando al futuro, il progetto potrebbe beneficiare di ulteriori miglioramenti come l'implementazione di funzionalità di autenticazione avanzate e l'integrazione con servizi esterni per arricchire le funzionalità offerte agli utenti.