Manuale Database

BOOK RECOMMENDER

App

Indice:

Introduzione

"Book Recommender" è un'applicazione Java che permette la gestione di librerie personali, la valutazione di libri e la raccomandazione di letture ad altri utenti. Il sistema mantiene un catalogo di libri che gli utenti possono aggiungere alle proprie librerie personali, valutare secondo diversi criteri e consigliare ad altri utenti.

Questo manuale descrive il meccanismo di persistenza utilizzato, documentando le diverse fasi del suo sviluppo, inclusi gli schemi e le query SQL utilizzate.

Raccolta ed Analisi dei Requisiti

In base alle specifiche, sono state individuate le seguenti entità:

- **Utenti**: persone che utilizzano il sistema, identificati da un ID utente univoco. Gli utenti possono creare librerie personali, aggiungere libri, valutarli e consigliare libri ad altri.
    - Attributi: userID, nome, cognome, codice fiscale, email, password
- **Libri**: elementi principali del sistema che possono essere aggiunti alle librerie personali.
    - Attributi: ID, titolo, autori, categoria, editore, anno di pubblicazione
- **Librerie**: collezioni di libri create dagli utenti.
    - Attributi: ID, nome, userID (utente proprietario)
- **Valutazioni**: giudizi espressi dagli utenti sui libri secondo vari criteri.
    - Attributi: userID, bookID, valutazioni per stile, contenuto, piacevolezza, originalità, edizione, commenti
- **Raccomandazioni**: suggerimenti di libri fatti da un utente ad altri.
    - Attributi: userID (chi raccomanda), bookID (libro origine), recommendedBookID (libro consigliato)

- Schema Scheletro

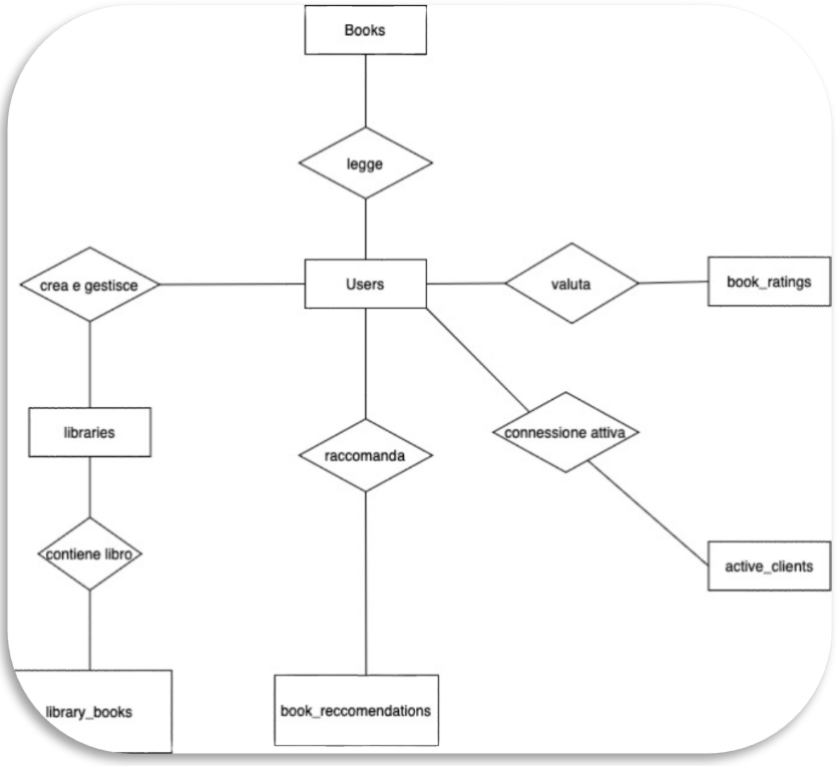

Figura 1: Schema scheletro del database che rappresenta la struttura fondamentale del sistema. Per una visualizzazione più dettagliata, fare clic sul pulsante "Visualizza su draw.io.

Visualizza su draw.io

Sulla base delle specifiche, è stato elaborato lo schema scheletro mostrato nella Figura 1. Questo schema rappresenta la struttura fondamentale del sistema Book Recommender e include tutte le entità e le relazioni necessarie, senza dettagli di attributi o vincoli di cardinalità.

Lo schema scheletro presentato nella Figura 1 rappresenta la struttura completa del sistema Book Recommender, comprensiva del componente di tracciamento delle connessioni. È possibile osservare:

1. Entità principali

Come evidenziato nella Figura 1, il sistema si compone di sette entità fondamentali:

- **Books**: rappresenta i libri nel catalogo del sistema, posizionata nella parte superiore del diagramma
- **Users**: rappresenta gli utenti registrati nel sistema, collocata al centro del diagramma come fulcro di tutte le relazioni
- **libraries**: rappresenta le collezioni personali di libri create dagli utenti, visibile sul lato sinistro
- **book_ratings**: rappresenta le valutazioni assegnate dagli utenti ai libri, posizionata sul lato destro
- **library_books**: tabella di join che collega libri e librerie (implementazione della relazione molti-a-molti), visibile in basso a sinistra
- **book_recommendations**: rappresenta i consigli di libri fatti dagli utenti, posizionata in basso al centro
- **active_clients**: entità che rappresenta le connessioni attive al sistema, permettendo di tracciare quali client sono attualmente connessi, visibile in basso a destra

2. Relazioni fondamentali

Il diagramma (Figura 1) mostra chiaramente le sei relazioni fondamentali che collegano le entità:

- **legge**: relazione tra Users e Books, indica l'azione di accesso ai libri da parte degli utenti
- **crea e gestisce**: relazione tra Users e libraries, mostra che un utente può creare e gestire collezioni personali
- **contiene libro**: relazione tra libraries e library_books, poi verso Books, indica quali libri sono contenuti in ciascuna libreria
- **valuta**: relazione tra Users e book_ratings, rappresenta le valutazioni degli utenti sui libri
- **raccomanda**: relazione tra Users e book_recommendations, mostra la possibilità degli utenti di consigliare libri
- **connessione attiva**: relazione tra Users e active_clients, rappresenta le sessioni attive degli utenti nel sistema

3. Osservazioni sull'implementazione

Analizzando lo schema scheletro (Figura 1) possiamo fare le seguenti osservazioni:

- La relazione "connessione attiva" è posizionata tra Users e active_clients, evidenziando che un utente può avere una connessione attiva nel sistema
- L'entità Users è posizionata al centro dello schema, sottolineando il suo ruolo centrale come collegamento tra tutte le altre entità
- La rappresentazione a rombo delle relazioni è appropriata per uno schema E-R e chiarisce la natura di ciascuna connessione tra entità

Relazioni nello Schema E-R

Basandosi sullo schema scheletro (Figura 1), sono state individuate le seguenti relazioni semantiche nell'architettura del sistema:

- **Crea e Gestisce**: un utente possiede da zero a molte librerie, una libreria appartiene a un solo utente.
- **Contiene Libro**: una libreria contiene da zero a molti libri, un libro può essere presente in molte librerie.
- **Valuta**: un utente valuta da zero a molti libri, un libro può essere valutato da molti utenti.
- **Raccomanda**: un utente consiglia da zero a molti libri ad altri utenti.
- **Legge**: un utente può leggere molti libri, un libro può essere letto da molti utenti.
- **Connessione Attiva**: un utente può avere da zero a una connessione attiva, ogni connessione attiva è associata a esattamente un utente.

Questa ultima relazione ("Connessione Attiva") merita particolare attenzione in quanto permette di implementare funzionalità critiche come il monitoraggio degli utenti online e la gestione delle sessioni, prevenendo login multipli dello stesso utente.

Lo schema scheletro (Figura 1) riflette con accuratezza l'implementazione presente nel codice del sistema, dove la tabella *active_clients* viene utilizzata per tenere traccia dei client connessi, informazione essenziale per la gestione della concorrenza, il monitoraggio del sistema e la prevenzione di conflitti di stato.

Partendo da questo schema scheletro, la progettazione procede verso uno schema concettuale completo con l'aggiunta di attributi, vincoli di cardinalità e identificatori, come illustrato nel successivo diagramma ER (Figura 2)

▪ Schema Concettuale

Il modello dati del sistema implementa relazioni chiare e ben definite tra le diverse entità che compongono il dominio applicativo, come illustrato nel diagramma ER (Figura 2).

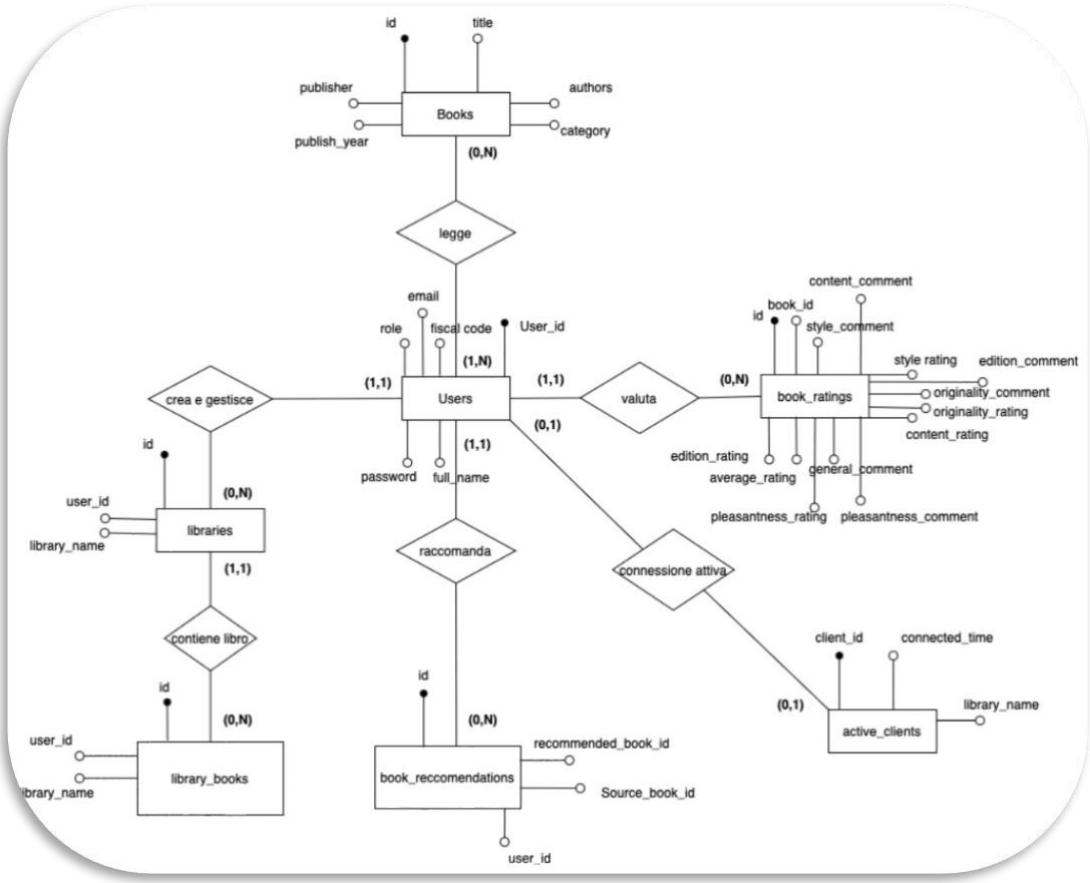

Figura 2: Diagramma ER del database con vincoli di cardinalità, attributi e identificatori. Per una visualizzazione più dettagliata, fare clic sul pulsante "Visualizza su draw.io".

Visualizza su draw.io

Nel passaggio da schema scheletro a concettuale sono stati aggiunti i vincoli di cardinalità, gli attributi e i vincoli di identificazione, come chiaramente visibile nel diagramma ER (Figura 2).La Figura 2 può essere riassunta cosi:

1) **Books**: L'entità centrale per il catalogo dei libri presenta:

- Un identificatore primario (id, evidenziato con un punto nero nel diagramma)
- Attributi descrittivi: title, publisher, publish_year, authors e category
- Cardinalità (0,N) verso la relazione "legge", indicando che un libro può essere letto da zero o più utenti

2) **Users**: L'entità che rappresenta gli utenti registrati include:

- Un identificatore primario (User_id, evidenziato con punto nero)
- Attributi personali: email, role, fiscal_code, password e full_name
- Multiple relazioni con cardinalità specifiche verso altre entità:
    - (1,1) verso "crea e gestisce" (ogni utente gestisce almeno una libreria)
    - (1,1) verso "valuta" (ogni utente può valutare libri)
    - (1,1) verso "raccomanda" (ogni utente può fare raccomandazioni)
    - (0,1) verso "connessione attiva" (un utente può avere al massimo una connessione attiva)
    - (1,N) verso "legge" (ogni utente legge almeno un libro)

3) **book_ratings**: L'entità che raccoglie le valutazioni degli utenti presenta:

- Un identificatore primario (id)
- Il riferimento al libro valutato (book_id)
- Numerosi attributi per i vari aspetti della valutazione:
    - style_rating e style_comment per lo stile di scrittura
    - content_rating e content_comment per i contenuti
    - originality_rating e originality_comment per l'originalità
    - pleasantness_rating e pleasantness_comment per la gradevolezza
    - edition_rating e edition_comment per la qualità dell'edizione
    - general_comment per commenti generali
    - average_rating calcolato come media delle valutazioni
- Cardinalità (0,N) verso la relazione "valuta", indicando che una valutazione appartiene a un solo utente

4) **libraries**: L'entità che gestisce le collezioni di libri include:

- Un identificatore primario (id)
- Gli attributi user_id e library_name
- Cardinalità (0,N) verso "crea e gestisce", indicando che una libreria appartiene a un solo utente
- Cardinalità (1,1) verso "contiene libro", mostrando che ogni libreria contiene almeno un libro

5) **library_books**: L'entità di associazione tra librerie e libri presenta:

- Un identificatore primario (id)
- I riferimenti user_id e library_name
- Cardinalità (0,N) verso "contiene libro", indicando che un record può appartenere a una sola libreria

6) **book_recommendations**: L'entità per le raccomandazioni include:

- Un identificatore primario (id)
- I riferimenti ai libri (recommended_book_id e Source_book_id)
- Il riferimento all'utente che ha fatto la raccomandazione (user_id)
- Cardinalità (0,N) verso "raccomanda", indicando che una raccomandazione appartiene a un solo utente

7) **active_clients**: L'entità che gestisce le sessioni attive include:

- Un identificatore primario (client_id)
- Gli attributi connected_time e library_name
- Cardinalità (0,1) verso "connessione attiva", garantendo che una sessione appartenga a un solo utente

Come evidenziato nel diagramma ER (Figura 2) tramite i punti neri sugli attributi, sono stati implementati i seguenti identificatori:

- Per le entità **books**, **libraries** e **book_ratings** sono stati aggiunti ID interi auto-incrementanti che fungono da chiavi primarie
- Per gli utenti è stato scelto il **fiscal_code** come identificatore, poiché campo obbligatorio e di lunghezza fissa
- Sono state individuate **email** e **userID** come chiavi alternative per gli utenti, garantendo unicità

Il diagramma ER (Figura 2) evidenzia chiaramente i vincoli di cardinalità per ogni relazione:

- Relazioni (1,1): indicano associazioni obbligatorie one-to-one
- Relazioni (1,N) o (N,1): rappresentano associazioni obbligatorie one-to-many
- Relazioni (0,1): indicano associazioni opzionali one-to-one
- Relazioni (0,N) o (N,0): rappresentano associazioni opzionali one-to-many

Questi vincoli garantiscono l'integrità referenziale e la correttezza delle operazioni sul database, assicurando che le relazioni tra entità rispettino i vincoli logici dell'applicazione.

Come evidenziato dai vincoli di cardinalità nel diagramma ER (Figura 2), il sistema implementa rigorosi meccanismi di gestione delle transazioni e di integrità referenziale. I vincoli di integrità referenziale sono applicati attraverso chiavi esterne con operazioni in cascata per gli eventi di DELETE, assicurando che non rimangano riferimenti orfani quando un'entità principale viene rimossa.

Questo approccio garantisce che operazioni complesse, come la creazione di una libreria con la contestuale aggiunta di libri (rappresentata dalle relazioni "crea e gestisce" con cardinalità (1,1) e "contiene libro" con cardinalità (1,1) nel diagramma), siano eseguite in modo atomico. In caso di errore durante l'esecuzione di una di queste operazioni composite, il sistema è in grado di effettuare un rollback completo, ripristinando lo stato precedente del database e prevenendo così inconsistenze nei dati.

Inizializzazione del Database

L'inizializzazione del database implementa nella pratica lo schema concettuale mostrato nel diagramma ER (Figura 2), con tutti i suoi attributi, vincoli di cardinalità e identificatori.

La procedura di inizializzazione, gestita dal metodo initializeDatabase() nella classe ServerInterfaceController, segue questi passaggi:

1. Eliminazione delle tabelle esistenti in ordine inverso rispetto alle dipendenze di integrità referenziale
2. Creazione delle tabelle con tutti gli attributi mostrati nel diagramma ER (Figura 5)
3. Implementazione dei vincoli di chiave primaria per gli identificatori evidenziati con punto nero
4. Creazione di indici per ottimizzare le query più frequenti
5. Popolamento con dati iniziali da file CSV

Il diagramma ER (Figura 2) rappresenta il blueprint completo su cui si basa questa inizializzazione, garantendo che il database fisico implementi fedelmente il modello concettuale progettato.

▪ Schema Logico

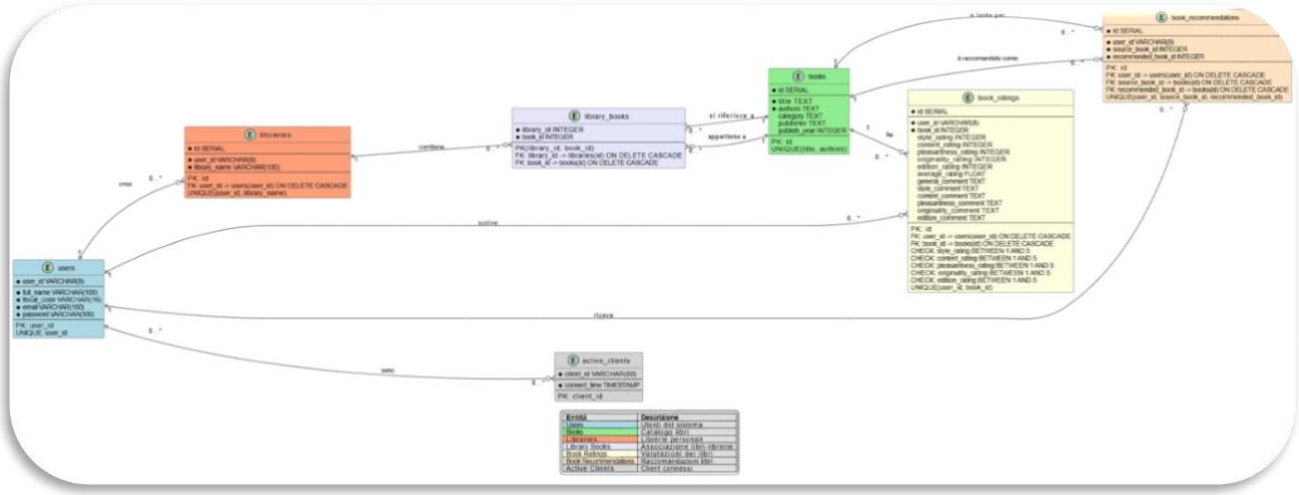

Figura 3: Schema ER del database dell'applicazione di gestione librerie. Per una visualizzazione più dettagliata, fare clic sul pulsante "Visualizza su draw.io".

Visualizza su draw.io

Lo schema logico finale implementa l'architettura del database dell'applicazione di gestione librerie tramite sei tabelle principali, chiaramente visibili nella Figura 3, dove sono rappresentate come entità con i relativi attributi e vincoli:

- **books**

Rappresenta il catalogo centrale dei libri disponibili nel sistema, visibile al centro della Figura 3 con sfondo verde

- id SERIAL PRIMARY KEY - Identificatore univoco autoincrementale, evidenziato come PK nella Figura 3
- title VARCHAR(256) NOT NULL - Titolo del libro (campo obbligatorio)
- authors VARCHAR(256) NOT NULL - Autori del libro (campo obbligatorio)
- category VARCHAR(256) - Genere o categoria del libro
- publisher VARCHAR(256) - Casa editrice
- publish_year INTEGER - Anno di pubblicazione
- UNIQUE(title, authors) - Vincolo di unicità evidenziato nella Figura 3 come "UNIQUE(title, authors)"

- **users (operatori_registrati)**:

Gestisce gli utenti registrati nel sistema, posizionata in basso a sinistra nella Figura 3 con sfondo azzurro

- codice_fiscale CHAR(16) PRIMARY KEY - Codice fiscale italiano come identificatore primario
- nome VARCHAR(256) NOT NULL - Nome dell'utente
- cognome VARCHAR(256) NOT NULL - Cognome dell'utente
- email VARCHAR(256) UNIQUE NOT NULL - Indirizzo email univoco
- user_id VARCHAR(256) UNIQUE NOT NULL - Username univoco per l'accesso, identificato come "UNIQUE: user_id" nella Figura 3
- password VARCHAR(256) NOT NULL - Password crittografata

- **libraries**:

Contiene le librerie personali create dagli utenti, rappresentata con sfondo arancione nella parte sinistra della Figura 3

- id SERIAL PRIMARY KEY - Identificatore univoco autoincrementale
- user_id VARCHAR(8) REFERENCES users(user_id) ON DELETE CASCADE - Chiave esterna che collega la libreria all'utente proprietario, visualizzata nella Figura 3 con la relazione "crea" (cardinalità 1 a 0..*)
- library_name VARCHAR(100) NOT NULL - Nome della libreria
- UNIQUE(user_id, library_name) - Vincolo che impedisce a un utente di creare più librerie con lo stesso nome, indicato nella Figura 3

- **library_books**

Tabella di associazione per la relazione many-to-many tra librerie e libri, visibile al centro della Figura 3 con sfondo viola

- library_id INTEGER REFERENCES libraries(id) ON DELETE CASCADE - Riferimento alla libreria, collegato con la relazione "contiene" (cardinalità 0..* a 0..*)
- book_id INTEGER REFERENCES books(id) ON DELETE CASCADE - Riferimento al libro, collegato con la relazione "appartiene a" (cardinalità 0..* a 1)
- PRIMARY KEY (library_id, book_id) - Chiave composita che impedisce duplicati

- **book_ratings**:

Implementa il sistema di valutazione multicriterio dei libri, rappresentata in alto a destra nella Figura 3 con sfondo giallo chiaro

- id SERIAL PRIMARY KEY - Identificatore univoco autoincrementale

- user_id VARCHAR(8) REFERENCES users(user_id) ON DELETE CASCADE - Utente che ha inserito la valutazione, collegato dalla relazione "scrive" (cardinalità 0..* a 1)
- book_id INTEGER REFERENCES books(id) ON DELETE CASCADE - Libro valutato, collegato dalla relazione "si riferisce a" (cardinalità 0..* a 1)
- Campi di valutazione (style_rating, content_rating, ecc.) - Tutti con vincoli CHECK mostrati nella Figura 3
- Campi di commento (general_comment, style_comment, ecc.) - Rappresentati come attributi TEXT
- UNIQUE(user_id, book_id) - Vincolo che permette una sola valutazione per libro per utente, indicato come "UNIQUE(user_id, book_id)"

- **book_recommendations**

Gestisce i consigli di lettura tra libri correlati, visibile in alto a destra nella Figura 3 con sfondo beige

- id SERIAL PRIMARY KEY - Identificatore univoco autoincrementale
- user_id VARCHAR(8) REFERENCES users(user_id) ON DELETE CASCADE - Utente che ha inserito il consiglio
- source_book_id INTEGER REFERENCES books(id) ON DELETE CASCADE - Libro di origine, collegato con la relazione "è fonte per" (cardinalità 0..* a 1)
- recommended_book_id INTEGER REFERENCES books(id) ON DELETE CASCADE - Libro consigliato, collegato con la relazione "è raccomandato come" (cardinalità 0..* a 1)
- UNIQUE(user_id, source_book_id, recommended_book_id)

Completando il modello, nella Figura 3 è presente anche la tabella "active_clients" (in grigio) che tiene traccia degli utenti attualmente connessi al sistema, con la relazione "sono" (cardinalità 0..* a 1) verso la tabella users. In basso nel diagramma è anche visibile una legenda che descrive tutte le entità rappresentate.

▪ Indici

Oltre agli indici generati automaticamente dal DBMS per le chiavi primarie e attributi UNIQUE, sono stati aggiunti degli indici sugli attributi su cui ci si aspetta che vengano eseguite il maggior numero di operazioni:

sql

```
CREATE INDEX title_idx ON books (title);
CREATE INDEX authors_idx ON books (authors);
CREATE INDEX user_id_idx ON users (user_id);
CREATE INDEX library_name_idx ON libraries (library_name);
CREATE INDEX book_ratings_book_id_idx ON book_ratings (book_id);
CREATE INDEX book_ratings_user_id_idx ON book_ratings (user_id);
CREATE INDEX library_books_book_id_idx ON library_books (book_id);
CREATE INDEX library_books_library_id_idx ON library_books (library_id);
```

▪ Query SQL

1) Creazione del database

sql

```
CREATE DATABASE "book_recommender";
```

Descrizione: Crea il database principale del sistema con nome "book_recommender", che conterrà tutte le tabelle dell'applicazione.

2) Creazione delle tabelle

2.1) books

sql

```
CREATE TABLE books
(
    id SERIAL PRIMARY KEY,
    title VARCHAR(256) NOT NULL,
    authors VARCHAR(256) NOT NULL,
    category VARCHAR(256),
    publisher VARCHAR(256),
    publish_year INTEGER,
    UNIQUE(title, authors)
);
```

Descrizione: Crea la tabella dei libri con un ID autoincrementante come chiave primaria. Include campi per titolo e autori (obbligatori), categoria, editore e anno di pubblicazione. La combinazione di titolo e autori deve essere unica per evitare duplicati nel catalogo.

2.2) users (operatori_registrati)

sql

```
CREATE TABLE operatori_registrati
(
    codice_fiscale CHAR(16) PRIMARY KEY,
    nome VARCHAR(256) NOT NULL,
    cognome VARCHAR(256) NOT NULL,
    email VARCHAR(256) UNIQUE NOT NULL,
    user_id VARCHAR(256) UNIQUE NOT NULL,
    password VARCHAR(256) NOT NULL,
    CONSTRAINT codice_fiscale_check CHECK (
        codice_fiscale ~ '^[a-zA-Z]{6}[0-9]{2}[abcdehlmprstABCDEHLMPRST]{1}[0-9]{2}([a-zA-Z]{1}[0-9]{3})[a-zA-Z]{1}$'
```

```
    ),
    CONSTRAINT email_check CHECK (
        email ~ '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
    )
);
```

Descrizione: Crea la tabella degli utenti registrati con il codice fiscale come chiave primaria. Implementa validazioni per il formato del codice fiscale italiano e dell'indirizzo email tramite espressioni regolari. Garantisce l'unicità di email e user_id per evitare registrazioni duplicate.

2.3) libraries

sql

```
CREATE TABLE libraries
(
    id SERIAL PRIMARY KEY,
    user_id VARCHAR(8) REFERENCES operatori_registrati(user_id) ON DELETE CASCADE,
    library_name VARCHAR(100) NOT NULL,
    UNIQUE(user_id, library_name)
);
```

Descrizione: Crea la tabella delle librerie personali degli utenti. Ogni libreria ha un ID autoincrementante, un riferimento all'utente proprietario e un nome. La combinazione di user_id e library_name deve essere unica (un utente non può avere due librerie con lo stesso nome). L'opzione CASCADE garantisce che se un utente viene eliminato, vengano eliminate anche tutte le sue librerie.

2.4) library_books

sql

```
CREATE TABLE library_books
(
    library_id INTEGER REFERENCES libraries(id) ON DELETE CASCADE,
```

```
    book_id INTEGER REFERENCES books(id) ON DELETE CASCADE,
    PRIMARY KEY (library_id, book_id)
);
```

Descrizione: Implementa la relazione many-to-many tra librerie e libri. Ogni riga rappresenta l'inclusione di un libro in una libreria specifica. La chiave primaria composta impedisce duplicati (lo stesso libro non può essere inserito due volte nella stessa libreria). Le opzioni CASCADE garantiscono la pulizia automatica dei record quando vengono eliminate librerie o libri.

2.5) book_ratings

sql

```
CREATE TABLE book_ratings
(
    id SERIAL PRIMARY KEY,
    user_id VARCHAR(8) REFERENCES operatori_registrati(user_id) ON DELETE CASCADE,
    book_id INTEGER REFERENCES books(id) ON DELETE CASCADE,
    style_rating INTEGER CHECK (style_rating >= 1 AND style_rating <= 5),
    content_rating INTEGER CHECK (content_rating >= 1 AND content_rating <= 5),
    pleasantness_rating INTEGER CHECK (pleasantness_rating >= 1 AND pleasantness_rating <= 5),
    originality_rating INTEGER CHECK (originality_rating >= 1 AND originality_rating <= 5),
    edition_rating INTEGER CHECK (edition_rating >= 1 AND edition_rating <= 5),
    average_rating FLOAT,
    general_comment TEXT,
    style_comment TEXT,
    content_comment TEXT,
    pleasantness_comment TEXT,
    originality_comment TEXT,
    edition_comment TEXT,
    UNIQUE(user_id, book_id),
    CHECK ((style_rating IS NOT NULL) OR (content_rating IS NOT NULL) OR (pleasantness_rating IS NOT NULL)
        OR (originality_rating IS NOT NULL) OR (edition_rating IS NOT NULL))
);
```

Descrizione: Crea la tabella per le valutazioni dei libri con un sistema multicriterio. Include valutazioni da 1 a 5 per stile, contenuto, gradevolezza, originalità ed edizione, con vincoli per garantire valori validi. Memorizza commenti separati per ciascun criterio e un commento generale. Garantisce che un utente possa valutare un libro una sola volta e che almeno uno dei criteri di valutazione sia specificato.

2.6) book_recommendations

sql

```
CREATE TABLE book_recommendations
(
    id SERIAL PRIMARY KEY,
    user_id VARCHAR(8) REFERENCES operatori_registrati(user_id) ON DELETE CASCADE,
    source_book_id INTEGER REFERENCES books(id) ON DELETE CASCADE,
    recommended_book_id INTEGER REFERENCES books(id) ON DELETE CASCADE,
    UNIQUE(user_id, source_book_id, recommended_book_id)
);
```

Descrizione: Crea la tabella per le raccomandazioni dei libri. Consente agli utenti di suggerire libri (recommended_book_id) in associazione a un libro di origine (source_book_id). Il vincolo UNIQUE garantisce che un utente non possa creare raccomandazioni duplicate.

3) Operazioni CRUD

3.1) books

- **Inserimento**:

sql

```
INSERT INTO books (title, authors, category, publisher, publish_year) VALUES (?, ?, ?, ?, ?);
```

Descrizione: Inserisce un nuovo libro nel catalogo con i suoi dettagli bibliografici.

- **Ricerca per titolo**:

sql

```
SELECT * FROM books WHERE title LIKE ? ORDER BY title LIMIT 100;
```

Descrizione: Cerca libri il cui titolo corrisponde parzialmente al pattern fornito, ordinandoli alfabeticamente e limitando i risultati a 100 per ottimizzare le prestazioni.

- **Ricerca per autore**:

sql

```
SELECT * FROM books WHERE authors LIKE ? ORDER BY authors LIMIT 100;
```

Descrizione: Cerca libri il cui autore corrisponde parzialmente al pattern fornito, ordinandoli per nome dell'autore e limitando i risultati a 100.

- **Ricerca per ID**:

sql

```
SELECT * FROM books WHERE id = ?;
```

Descrizione: Recupera un libro specifico tramite il suo ID univoco.

Aggiornamento:

sql

```
UPDATE books SET title = ?, authors = ?, category = ?, publisher = ?, publish_year = ? WHERE id = ?;
```

Descrizione: Aggiorna i dettagli di un libro esistente, identificato dal suo ID.

- **Cancellazione**:

sql

```
DELETE FROM books WHERE id = ?;
```

Descrizione: Rimuove un libro dal catalogo in base al suo ID.

3.2) users (operatori_registrati)

- **Inserimento**:

sql

```
INSERT INTO operatori_registrati (codice_fiscale, nome, cognome, email, user_id, password) VALUES (?, ?, ?, ?, ?, ?);
```

Descrizione: Registra un nuovo utente nel sistema con i suoi dati personali e credenziali.

- **Ricerca per user_id**:

sql

```
SELECT * FROM operatori_registrati WHERE user_id = ?;
```

Descrizione: Recupera le informazioni di un utente specifico tramite il suo ID utente, usato principalmente per l'autenticazione.

- **Ricerca per email**:

sql

```
SELECT * FROM operatori_registrati WHERE email = ?;
```

Descrizione: Cerca un utente specifico tramite il suo indirizzo email, utile per il recupero password o la verifica dell'unicità dell'email.

- **Aggiornamento**:

sql

```
UPDATE operatori_registrati SET nome = ?, cognome = ?, email = ?, password = ? WHERE codice_fiscale = ?;
```

Descrizione: Aggiorna i dati personali di un utente esistente, mantenendo il codice fiscale come riferimento immutabile.

- **Cancellazione**:

sql

```
DELETE FROM operatori_registrati WHERE codice_fiscale = ?;
```

Descrizione: Rimuove un utente dal sistema in base al suo codice fiscale, eliminando a cascata anche tutte le sue librerie, valutazioni e raccomandazioni.

3.3) Libraries

- **Inserimento**:

sql

```
INSERT INTO libraries (user_id, library_name) VALUES (?, ?);
```

Descrizione: Crea una nuova libreria personale per un utente specifico con il nome fornito.

- **Ricerca per utente**:

sql

```
SELECT * FROM libraries WHERE user_id = ?;
```

Descrizione: Recupera tutte le librerie appartenenti a un determinato utente.

- **Aggiornamento**:

sql

```
UPDATE libraries SET library_name = ? WHERE id = ?;
```

Descrizione: Modifica il nome di una libreria esistente, identificata dal suo ID.

- **Cancellazione**:

sql

```
DELETE FROM libraries WHERE id = ?;
```

Descrizione: Elimina una libreria e, a cascata, tutti i suoi contenuti (riferimenti ai libri).

3.4) library_books

- **Inserimento**:

sql

```
INSERT INTO library_books (library_id, book_id) VALUES (?, ?);
```

Descrizione: Aggiunge un libro specifico a una libreria, creando l'associazione tra i due.

- **Ricerca libri di una libreria**:

sql

```
SELECT b.* FROM library_books lb JOIN books b ON lb.book_id = b.id WHERE lb.library_id = ?;
```

Descrizione: Recupera tutti i libri contenuti in una determinata libreria, unendo le tabelle library_books e books.

- **Cancellazione**:

sql

```
DELETE FROM library_books WHERE library_id = ? AND book_id = ?;
```

Descrizione: Rimuove un libro specifico da una libreria, eliminando l'associazione senza cancellare né il libro né la libreria.

3.5) book_ratings

- **Inserimento**:

sql

```
INSERT INTO book_ratings (user_id, book_id, style_rating, content_rating, pleasantness_rating, originality_rating, edition_rating, average_rating, general_comment, style_comment, content_comment, pleasantness_comment, originality_comment, edition_comment) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
```

Descrizione: Memorizza una nuova valutazione completa di un libro, includendo punteggi per ciascun criterio e commenti relativi.

- **Ricerca valutazioni di un libro**:

sql

```
SELECT * FROM book_ratings WHERE book_id = ?;
```

Descrizione: Recupera tutte le valutazioni assegnate a un libro specifico da diversi utenti.

- **Ricerca valutazioni di un utente**:

sql

```
SELECT * FROM book_ratings WHERE user_id = ?;
```

Descrizione: Ottiene tutte le valutazioni create da un determinato utente per diversi libri.

- **Calcolo rating medio di un libro**:

sql

```
SELECT AVG(style_rating) as avg_style, AVG(content_rating) as avg_content, AVG(pleasantness_rating) as avg_pleasantness, AVG(originality_rating) as avg_originality, AVG(edition_rating) as avg_edition, AVG(average_rating) as avg_total FROM book_ratings WHERE book_id = ?;
```

Descrizione: Calcola il punteggio medio per ciascun criterio di valutazione di un libro specifico, aggregando le valutazioni di tutti gli utenti.

- **Aggiornamento**:

sql

```
UPDATE book_ratings SET style_rating = ?, content_rating = ?, pleasantness_rating = ?, originality_rating = ?, edition_rating = ?, average_rating = ?, general_comment = ?, style_comment = ?, content_comment = ?, pleasantness_comment = ?, originality_comment = ?, edition_comment = ? WHERE user_id = ? AND book_id = ?;
```

Descrizione: Modifica una valutazione esistente, permettendo all'utente di aggiornare sia i punteggi che i commenti per un libro.

- **Cancellazione**:

sql

```
DELETE FROM book_ratings WHERE user_id = ? AND book_id = ?;
```

Descrizione: Rimuove la valutazione di un libro da parte di un utente specifico.

3.6) book_recommendations

- **Inserimento**:

sql

```
INSERT INTO book_recommendations (user_id, source_book_id, recommended_book_id) VALUES (?, ?, ?);
```

Descrizione: Crea una nuova raccomandazione di un libro in associazione a un libro di origine da parte di un utente.

- **Ricerca raccomandazioni per un libro di origine**:

sql

```
SELECT r.*, b.title, b.authors FROM book_recommendations r JOIN books b ON r.recommended_book_id = b.id
WHERE r.source_book_id = ?;
```

Descrizione: Trova tutti i libri consigliati in relazione a un libro specifico, includendo i dettagli dei libri consigliati.

- **Ricerca raccomandazioni di un utente**:

sql

```
SELECT r.*, b1.title as source_title, b1.authors as source_authors, b2.title as recommended_title, b2.authors as
recommended_authors FROM book_recommendations r JOIN books b1 ON r.source_book_id = b1.id JOIN books b2
ON r.recommended_book_id = b2.id WHERE r.user_id = ?;
```

Descrizione: Recupera tutte le raccomandazioni create da un utente specifico, includendo i dettagli completi sia dei libri di origine che di quelli consigliati.

- **Cancellazione**:

sql

```
DELETE FROM book_recommendations WHERE user_id = ? AND source_book_id = ? AND recommended_book_id
= ?;
```

Descrizione: Elimina una specifica raccomandazione identificata dalla combinazione di utente, libro di origine e libro consigliato.

Conclusioni

Il database del Book Recommender System rappresenta un elemento fondamentale dell'architettura complessiva dell'applicazione, implementando un sistema di persistenza robusto e ben strutturato che supporta tutte le funzionalità chiave del sistema.

Lo sviluppo del database ha seguito un approccio metodico, partendo dall'analisi dei requisiti e procedendo attraverso la creazione di schemi progressivamente più dettagliati: dallo schema scheletro iniziale che ha definito le entità principali e le loro relazioni, allo schema concettuale completo con attributi e vincoli di cardinalità, fino allo schema logico finale con la definizione dettagliata delle tabelle SQL.

Il modello realizzato garantisce:

- **Integrità dei dati**: attraverso vincoli di chiave primaria, chiavi esterne con azioni CASCADE, e controlli di validità sui dati inseriti.
- **Normalizzazione**: lo schema rispetta la Terza Forma Normale (3NF), eliminando ridondanze e dipendenze problematiche.
- **Prestazioni ottimizzate**: gli indici strategicamente posizionati sugli attributi più frequentemente interrogati migliorano significativamente i tempi di risposta delle query.
- **Flessibilità**: la struttura supporta facilmente l'espansione con nuovi attributi o funzionalità, mantenendo la compatibilità con l'architettura esistente.
- **Sicurezza**: sono implementati meccanismi per la verifica e validazione dei dati inseriti, come i controlli sul formato del codice fiscale e degli indirizzi email.

Il sistema di valutazione multicriterio rappresenta una caratteristica distintiva del database, permettendo agli utenti di esprimere giudizi dettagliati sui vari aspetti dei libri (stile, contenuto, gradevolezza, originalità, edizione) e arricchendo così significativamente la qualità delle informazioni disponibili per gli altri utenti.

La funzionalità di tracciamento delle connessioni attive, implementata attraverso la tabella active_clients, costituisce un elemento chiave per la gestione della concorrenza e la prevenzione di conflitti di stato, garantendo un'esperienza utente fluida e coerente anche in scenari di utilizzo intenso.

Le query SQL presentate in questo documento coprono tutte le operazioni CRUD (Create, Read, Update, Delete) necessarie per il funzionamento dell'applicazione, fornendo un'interfaccia completa tra il livello applicativo e il database sottostante.

In futuro, il database potrebbe essere ulteriormente evoluto per supportare funzionalità avanzate come:

- Sistema di hashing delle password per migliorare la sicurezza
- Meccanismi di audit trail per tracciare le modifiche ai dati
- Supporto multilingua per titoli, descrizioni e recensioni
- Integrazione con sistemi esterni di catalogazione (ISBN, classificazioni bibliotecarie)
- Implementazione di viste materializzate per ottimizzare query analitiche complesse

Il database del Book Recommender System, nella sua implementazione attuale, soddisfa pienamente i requisiti dell'applicazione, fornendo un solido fondamento per tutte le funzionalità di gestione delle librerie personali, valutazione dei libri e raccomandazione di letture.

Strumenti Utilizzati

- Draw.io / diagrams.net. (2024). *Online Diagram Software & Flowchart Maker*. https://www.diagrams.net/.

Contatti

Per ulteriori informazioni riguardo al database del sistema Book Recommender, contattare:

- Project manager : agervasini1@studenti.uninsubria.it