

Libreria Online - Basi di dati

Alessio Morselli e Alex Covizzi

Luglio 2017

Abstract

Questo documento contiene tutta la documentazione riguardo la struttura del database dell'applicazione Web da noi realizzata. Si presentano inoltre i codici SQL della creazione delle tabelle e delle viste che compongono la base di dati, nonché alcune query di ricerca e inserimento di dati ritenute particolarmente importanti e interessanti.

1 Introduzione

L'applicazione Web di cui viene presentato la parte della base di dati realizza il sito di e-commerce di una libreria online. Il sito offre diverse funzionalità, di cui viene presentata una breve descrizione:

- Ricerca nel catalogo: il sito mette a disposizione diversi parametri di ricerca, permettendo di filtrare il catalogo in diversi modi: ricerca per titolo, codice ISBN, autore, editore, genere, prezzo e voto minimo.
- Registrazione: un utente può decidersi di iscriversi al sito per ottenere l'accesso a diverse funzioni aggiuntive (presentate di seguito a questa).
- Gestione del proprio account: comprende la gestione di una lista dei desideri (aggiunta e rimozione), la gestione del carrello (aggiunta, rimozione o modifica alla quantità), il tracking dei propri ordini, comprensivo dello storico degli ordini effettuati, la gestione delle proprie recensioni (modifica o rimozione) e la possibilità di cambiare le informazioni personali dell'account (nome, cognome, indirizzo e-mail e password).
- Acquisto: un utente può comprare le merci aggiunte al carrello, specificando indirizzo di consegna, coordinate per il pagamento ed eventualmente un codice di un coupon sconto.
- Recensione: un utente può recensire un libro tramite un voto (mi è piaciuto / non mi è piaciuto) e un commento facoltativo.
- Funzioni amministratore: un amministratore ha diverse responsabilità, come mantenere aggiornato il catalogo (aggiunta, modifica o rimozione di libri), la gestione degli utenti (può bloccare o sbloccare un utente), la gestione degli altri amministratori (aggiunta e rimozione), la gestione degli ordini (aggiornamento dello stato degli ordini effettuati), la gestione delle recensioni e la gestione dei coupon.

2 Struttura della base di dati

2.1 Modello concettuale

Nel creare la base di dati abbiamo individuato i tipi di entità dell'applicazione, trovandone otto (di cui uno debole). Come si può notare dal diagramma ER sottostante, sono tre i principali tipi di entità: User, Order e Book sono infatti le entità su cui si basa l'applicazione e che la denotano come un sito di e-commerce di libri.

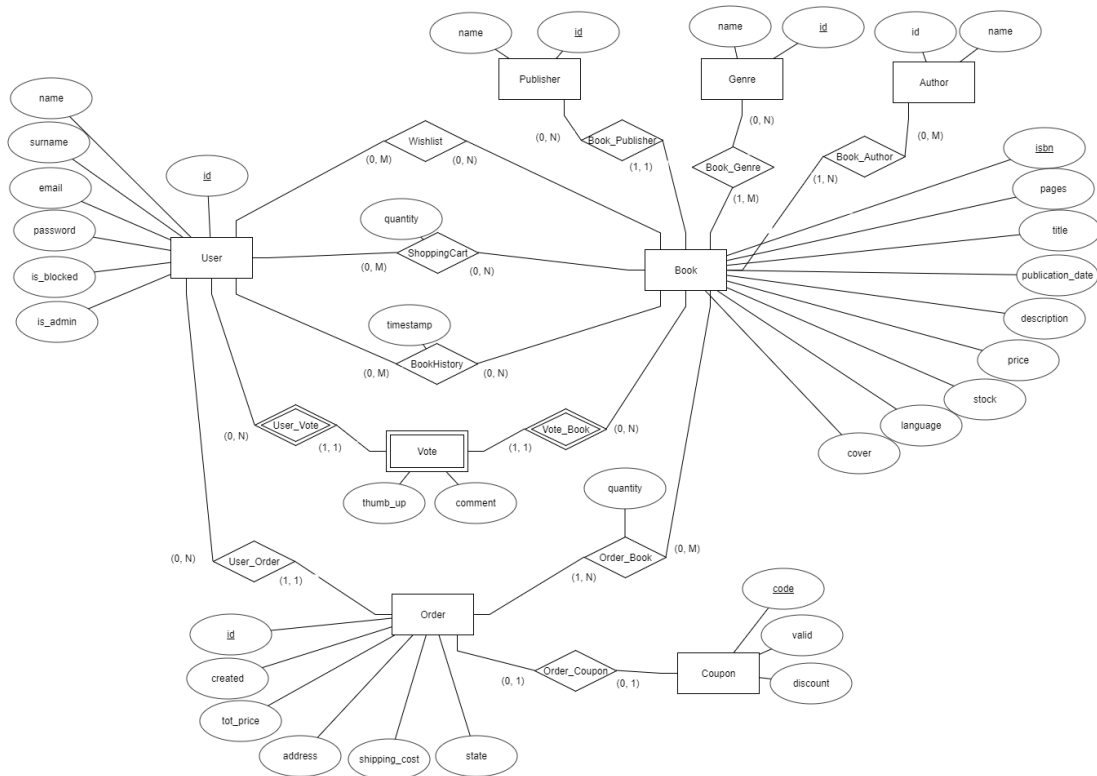


Figure 1: Il diagramma concettuale della base di dati.

Si è deciso di assegnare come chiave primaria della maggior parte dei tipi di entità un id numerico, con qualche eccezione: un'entità di tipo Book, infatti, ha come chiave primaria il proprio codice ISBN (di 13 cifre), in quanto è unico per ciascun libro, mentre un'entità di tipo Coupon viene identificata dal codice del coupon stesso, in quanto non potranno esistere coupon che condividono lo stesso codice sconto.

Anche l'entità Vote fa eccezione, ma in questo caso è perché si tratta di un tipo di entità debole: ogni entità di questo tipo è identificata dall'id utente e dal codice ISBN del libro, in modo che fin dal livello di database si impedisce ad ogni utente di avere più recensioni per libro.

2.2 Modello logico

Nonostante il modello concettuale sia teoricamente da realizzare per primo, è stato il diagramma relazionale quello su cui in realtà abbiamo lavorato la maggior parte del tempo e che è stato realizzato anche per primo. Per realizzarlo ci siamo serviti del programma MySQLWorkbench, che ci ha fornito un modo semplice per creare la nostra base di dati tramite il diagramma relazionale.

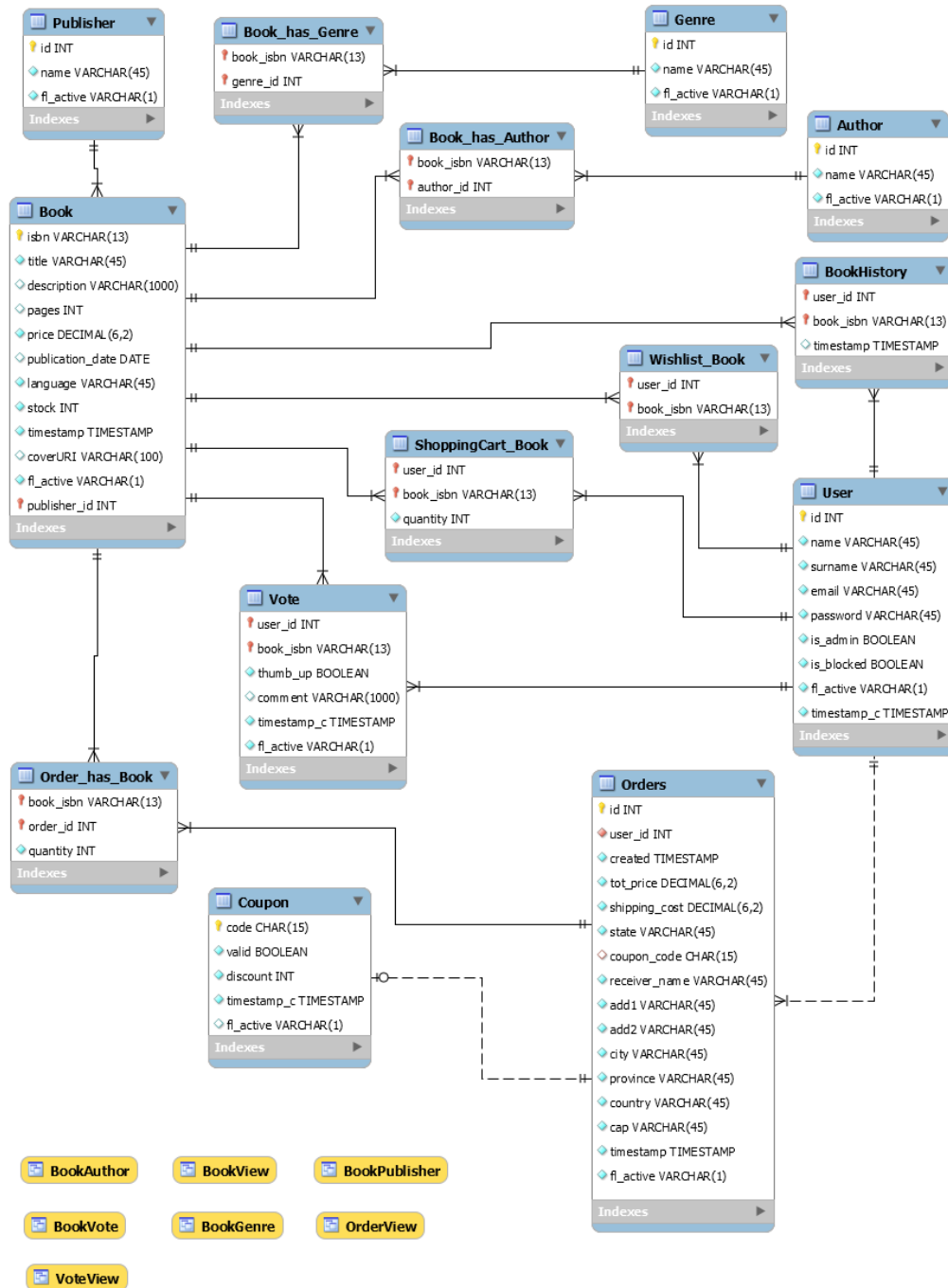






Figure 2: Il diagramma relazionale della base di dati.

Presento qui una breve legenda per comprendere meglio il diagramma:

- - Chiave primaria.
- - Attributo che non può assumere valore null.

-  - Attributo che può assumere valore null.
-  - Chiave esterna che assume valore di chiave nella relazione.
-  - Chiave esterna che non assume valore di chiave nella relazione, ma che deve assumere obbligatoriamente un valore non null.
-  - Chiave esterna che non assume valore di chiave nella relazione e che può assumere valore null.

Per quanto riguarda il riferimento dei vincoli di integrità referenziale, è il nome della chiave esterna che indica a quale relazione fa riferimento tale chiave esterna (ad esempio, "book_isbn" di Vote fa riferimento alla chiave di Book "isbn").

Infine, nel diagramma abbiamo compreso anche le viste che abbiamo utilizzato nell'applicazione, la cui struttura è compresa nel codice SQL presentato nella sezione successiva.

Abbiamo deciso di inserire nelle relazioni più importanti un flag che segnala se il dato è o meno da visualizzare: in questo modo, quando viene eseguito un'operazione di rimozione di un dato non viene effettivamente eliminata la riga dalla tabella (tranne alcune eccezioni), ma solo impostato il flag in modo che la riga venga ignorata dalle query di ricerca. In questo modo, si vuole dare la possibilità di memorizzare comunque uno storico dei dati inseriti accedendo direttamente al database.

3 Definizione di tabelle e viste

Passiamo ora al codice che ci ha permesso di creare le tabelle e le viste della base di dati. Come già detto, abbiamo creato la struttura del database tramite l'interfaccia grafica offerta dal programma MySQLWorkbench, che ha generato automaticamente, a partire dalle tabelle inserite, il codice SQL. Invece, il codice SQL per creare le viste non è stato generato automaticamente, ma abbiamo specificato noi come queste dovessero essere strutturate.

3.1 Definizione di tabelle

Di seguito, il codice SQL usato per definire le tabelle.

```
-- Table 'e-commerce'. 'User'
```

```
CREATE TABLE IF NOT EXISTS 'e-commerce'. 'User' (  
  'id' INT NOT NULL,  
  'name' VARCHAR(45) NOT NULL,  
  'surname' VARCHAR(45) NOT NULL,  
  'email' VARCHAR(45) NOT NULL,  
  'password' VARCHAR(45) NOT NULL,  
  'is_admin' TINYINT(1) NOT NULL DEFAULT 0,  
  'is_blocked' TINYINT(1) NOT NULL DEFAULT 0,  
  'fl_active' VARCHAR(1) NOT NULL DEFAULT 'S',  
  'timestamp_c' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  PRIMARY KEY ('id'));
```

```
-- Table 'e-commerce'. 'Publisher'
```

```
CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Publisher' (  
  'id' INT NOT NULL,  
  'name' VARCHAR(45) NOT NULL,  
  'fl_active' VARCHAR(1) NOT NULL DEFAULT 'S',  
  PRIMARY KEY ('id'));
```

```
-- Table 'e-commerce'. 'Book'
```

```
CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Book' (  
  'isbn' VARCHAR(13) NOT NULL,  
  'title' VARCHAR(45) NOT NULL,  
  'description' VARCHAR(1000) NULL,  
  'pages' INT UNSIGNED NULL,  
  'price' DECIMAL(6,2) UNSIGNED NOT NULL,  
  'publication_date' DATE NULL,  
  'language' VARCHAR(45) NOT NULL,  
  'stock' INT UNSIGNED NOT NULL,  
  'timestamp' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  'coverURI' VARCHAR(100) NULL,  
  'fl_active' VARCHAR(1) NOT NULL DEFAULT 'S',  
  'publisher_id' INT NULL,  
  PRIMARY KEY ('isbn', 'publisher_id'),  
  INDEX 'fk_Book_Publisher1_idx' ('publisher_id' ASC),  
  CONSTRAINT 'fk_Book_Publisher1'  
    FOREIGN KEY ('publisher_id')  
    REFERENCES 'e-commerce'. 'Publisher' ('id')  
    ON DELETE CASCADE  
    ON UPDATE CASCADE);
```

-- Table 'e-commerce'. 'Genre'

```
CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Genre' (  
  'id' INT NOT NULL,  
  'name' VARCHAR(45) NOT NULL,  
  'fl_active' VARCHAR(1) NOT NULL DEFAULT 'S',  
  PRIMARY KEY ('id'));
```

-- Table 'e-commerce'. 'Author'

```
CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Author' (  
  'id' INT NOT NULL,  
  'name' VARCHAR(45) NOT NULL,  
  'fl_active' VARCHAR(1) NOT NULL DEFAULT 'S',  
  PRIMARY KEY ('id'));
```

-- Table 'e-commerce'. 'Coupon'

```
CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Coupon' (  
  'code' CHAR(15) NOT NULL,  
  'valid' TINYINT(1) NOT NULL DEFAULT 1,  
  'discount' INT NOT NULL DEFAULT 0,  
  'timestamp_c' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  'fl_active' VARCHAR(1) NULL,  
  PRIMARY KEY ('code'));
```

-- Table 'e-commerce'. 'Orders'

```
CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Orders' (  
  'id' INT NOT NULL,  
  'user_id' INT NOT NULL,  
  'created' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  'tot_price' DECIMAL(6,2) UNSIGNED NOT NULL,  
  'shipping_cost' DECIMAL(6,2) NOT NULL,  
  'state' VARCHAR(45) NOT NULL,  
  'coupon_code' CHAR(15) NULL,  
  'receiver_name' VARCHAR(45) NOT NULL,  
  'add1' VARCHAR(45) NOT NULL,  
  'add2' VARCHAR(45) NOT NULL,  
  'city' VARCHAR(45) NOT NULL,  
  'province' VARCHAR(45) NOT NULL,  
  'country' VARCHAR(45) NOT NULL,  
  'cap' VARCHAR(45) NOT NULL,  
  'timestamp' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
  'fl_active' VARCHAR(1) NOT NULL DEFAULT 'S',  
  PRIMARY KEY ('id'),  
  INDEX 'fk_Order_User1_idx' ('user_id' ASC),  
  INDEX 'fk_Order_Coupon1_idx' ('coupon_code' ASC),  
  CONSTRAINT 'fk_Order_User1'  
    FOREIGN KEY ('user_id')  
    REFERENCES 'e-commerce'. 'User' ('id')  
    ON DELETE CASCADE  
    ON UPDATE CASCADE,  
  CONSTRAINT 'fk_Order_Coupon1'  
    FOREIGN KEY ('coupon_code')  
    REFERENCES 'e-commerce'. 'Coupon' ('code')  
    ON DELETE CASCADE
```


ON UPDATE CASCADE);

-- Table 'e-commerce'. 'Order_has_Book'

CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Order_has_Book' (
 'book_isbn' **VARCHAR**(13) **NOT NULL**,
 'order_id' **INT NOT NULL**,
 'quantity' **INT UNSIGNED NOT NULL DEFAULT** 1,
 PRIMARY KEY ('book_isbn', 'order_id'),
 INDEX 'fk_Order_Book1_idx' ('book_isbn' **ASC**),
 INDEX 'fk_OrderBook_Order1_idx' ('order_id' **ASC**),
 CONSTRAINT 'fk_Order_Book1'
 FOREIGN KEY ('book_isbn')
 REFERENCES 'e-commerce'. 'Book' ('isbn')
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 CONSTRAINT 'fk_OrderBook_Order1'
 FOREIGN KEY ('order_id')
 REFERENCES 'e-commerce'. 'Orders' ('id')
 ON DELETE CASCADE
 ON UPDATE CASCADE);

-- Table 'e-commerce'. 'Book_has_Genre'

CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Book_has_Genre' (
 'book_isbn' **VARCHAR**(13) **NOT NULL**,
 'genre_id' **INT NOT NULL**,
 PRIMARY KEY ('book_isbn', 'genre_id'),
 INDEX 'fk_Book_has_Genre_Genre1_idx' ('genre_id' **ASC**),
 INDEX 'fk_Book_has_Genre_Book1_idx' ('book_isbn' **ASC**),
 CONSTRAINT 'fk_Book_has_Genre_Book1'
 FOREIGN KEY ('book_isbn')
 REFERENCES 'e-commerce'. 'Book' ('isbn')
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 CONSTRAINT 'fk_Book_has_Genre_Genre1'
 FOREIGN KEY ('genre_id')
 REFERENCES 'e-commerce'. 'Genre' ('id')
 ON DELETE CASCADE
 ON UPDATE CASCADE);

-- Table 'e-commerce'. 'Book_has_Author'

CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Book_has_Author' (
 'book_isbn' **VARCHAR**(13) **NOT NULL**,
 'author_id' **INT NOT NULL**,
 PRIMARY KEY ('book_isbn', 'author_id'),
 INDEX 'fk_Book_has_Author_Author1_idx' ('author_id' **ASC**),
 INDEX 'fk_Book_has_Author_Book1_idx' ('book_isbn' **ASC**),
 CONSTRAINT 'fk_Book_has_Author_Book1'
 FOREIGN KEY ('book_isbn')
 REFERENCES 'e-commerce'. 'Book' ('isbn')
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 CONSTRAINT 'fk_Book_has_Author_Author1'
 FOREIGN KEY ('author_id')
 REFERENCES 'e-commerce'. 'Author' ('id')

**ON DELETE CASCADE
ON UPDATE CASCADE);**

-- Table 'e-commerce'. 'Wishlist_Book'

CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Wishlist_Book' (
 'user_id' INT NOT NULL,
 'book_isbn' VARCHAR(13) NOT NULL,
 PRIMARY KEY ('user_id', 'book_isbn'),
 INDEX 'fk_User_has_Book_Book1_idx' ('book_isbn' ASC),
 INDEX 'fk_User_has_Book_User1_idx' ('user_id' ASC),
 CONSTRAINT 'fk_User_has_Book_User1'
 FOREIGN KEY ('user_id')
 REFERENCES 'e-commerce'. 'User' ('id')
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 CONSTRAINT 'fk_User_has_Book_Book1'
 FOREIGN KEY ('book_isbn')
 REFERENCES 'e-commerce'. 'Book' ('isbn')
 ON DELETE CASCADE
 ON UPDATE CASCADE);

-- Table 'e-commerce'. 'BookHistory'

CREATE TABLE IF NOT EXISTS 'e-commerce'. 'BookHistory' (
 'user_id' INT NOT NULL,
 'book_isbn' VARCHAR(13) NOT NULL,
 'timestamp' TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,
 PRIMARY KEY ('user_id', 'book_isbn'),
 INDEX 'fk_User_has_Book_Book2_idx' ('book_isbn' ASC),
 INDEX 'fk_User_has_Book_User2_idx' ('user_id' ASC),
 CONSTRAINT 'fk_User_has_Book_User2'
 FOREIGN KEY ('user_id')
 REFERENCES 'e-commerce'. 'User' ('id')
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 CONSTRAINT 'fk_User_has_Book_Book2'
 FOREIGN KEY ('book_isbn')
 REFERENCES 'e-commerce'. 'Book' ('isbn')
 ON DELETE CASCADE
 ON UPDATE CASCADE);

-- Table 'e-commerce'. 'ShoppingCart_Book'

CREATE TABLE IF NOT EXISTS 'e-commerce'. 'ShoppingCart_Book' (
 'user_id' INT NOT NULL,
 'book_isbn' VARCHAR(13) NOT NULL,
 'quantity' INT NOT NULL DEFAULT 1,
 PRIMARY KEY ('user_id', 'book_isbn'),
 INDEX 'fk_ShoppingCart_has_Book_Book1_idx' ('book_isbn' ASC),
 INDEX 'fk_ShoppingCart_has_Book_User1_idx' ('user_id' ASC),
 CONSTRAINT 'fk_ShoppingCart_has_Book_Book1'
 FOREIGN KEY ('book_isbn')
 REFERENCES 'e-commerce'. 'Book' ('isbn')
 ON DELETE CASCADE
 ON UPDATE CASCADE,
 CONSTRAINT 'fk_ShoppingCart_has_Book_User1'

```

FOREIGN KEY ('user_id')
REFERENCES 'e-commerce'. 'User' ('id')
ON DELETE NO ACTION
ON UPDATE NO ACTION);

```

```

-- Table 'e-commerce'. 'Vote'

```

```

CREATE TABLE IF NOT EXISTS 'e-commerce'. 'Vote' (
  'user_id' INT NOT NULL,
  'book_isbn' VARCHAR(13) NOT NULL,
  'thumb_up' TINYINT(1) NOT NULL,
  'comment' VARCHAR(1000) NULL,
  'timestamp_c' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  'fl_active' VARCHAR(1) NOT NULL DEFAULT 'S',
  PRIMARY KEY ('user_id', 'book_isbn'),
  INDEX 'fk_User_has_Book_Book3_idx' ('book_isbn' ASC),
  INDEX 'fk_User_has_Book_User3_idx' ('user_id' ASC),
  CONSTRAINT 'fk_User_has_Book_User3'
    FOREIGN KEY ('user_id')
    REFERENCES 'e-commerce'. 'User' ('id')
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT 'fk_User_has_Book_Book3'
    FOREIGN KEY ('book_isbn')
    REFERENCES 'e-commerce'. 'Book' ('isbn')
    ON DELETE CASCADE
    ON UPDATE CASCADE);

```

3.2 Definizione di viste

Viene ora presentato il codice di definizione delle viste utilizzate.

```
CREATE OR REPLACE VIEW 'BookView' AS
SELECT B.isbn, B.title, B.description, B.pages, B.price, B.publication_date, B.language,
P.name as publisher_name, B.stock, BV.vote, BV.total as n_votes, B.coverUri, B.timestamp
FROM Book as B
LEFT JOIN BookVote as BV ON BV.book_isbn = B.isbn
JOIN Publisher as P ON B.publisher_id = P.id
WHERE B.fl_active = 'S' AND P.fl_active = 'S';

-----
-- View 'e-commerce'.BookVote
-----

CREATE OR REPLACE VIEW 'BookVote' AS
SELECT book_isbn, SUM(thumb_up = 1) as thumbs_up, SUM(thumb_up = 0) as thumbs_down,
COUNT(*) as total, AVG(thumb_up=1) as vote
FROM Vote
GROUP BY book_isbn;

-----
-- View 'e-commerce'.BookAuthor
-----

CREATE OR REPLACE VIEW 'BookAuthor' AS
SELECT book_isbn, name as a_name FROM Book_has_Author, Author
WHERE id = author_id AND fl_active = 'S';

-----
-- View 'e-commerce'.BookGenre
-----

CREATE OR REPLACE VIEW 'BookGenre' AS
SELECT book_isbn, name as g_name FROM Book_has_Genre, Genre
WHERE id = genre_id AND fl_active = 'S';

-----
-- View 'e-commerce'.BookPublisher
-----

CREATE OR REPLACE VIEW 'BookPublisher' AS
SELECT B.isbn as book_isbn, P.name as p_name FROM Book as B, Publisher as P
WHERE B.publisher_id = P.id AND P.fl_active = 'S' AND B.fl_active = 'S';

-----
-- View 'e-commerce'.OrderView
-----

CREATE OR REPLACE VIEW 'OrderView' AS
SELECT O.id, O.user_id, U.name as user_name, U.surname as user_surname, O.created, O.tot_price,
O.shipping_cost, O.state, O.coupon_code, O.receiver_name, O.add1, O.add2, O.city,
O.province, O.country, O.cap
FROM Orders as O
JOIN User as U ON U.id = O.user_id
WHERE O.fl_active = 'S';

-----
-- View 'e-commerce'.VoteView
-----

CREATE OR REPLACE VIEW 'VoteView' AS
SELECT V.book_isbn, B.title, V.user_id, U.name, U.surname, V.thumb_up, V.comment, V.timestamp_c
FROM Vote as V
JOIN Book as B ON V.book_isbn = B.isbn
JOIN User as U ON V.user_id = U.id
WHERE B.fl_active = 'S' AND U.fl_active = 'S';
```

4 Interrogazioni al database

Vengono qui presentate alcune interrogazioni importanti svolte sul database.

4.1 Ricerca di un libro

Nota: i parametri di ricerca sono ovviamente passati a livello di linguaggio Java.

```
SELECT DISTINCT isbn, title, price, publisher_name, publication_date, stock, vote, n_votes, coverUri
FROM BookView
JOIN BookAuthor AS B_A ON ( B_A.book_isbn = isbn )
JOIN BookGenre AS B_G ON ( B_G.book_isbn = isbn )
WHERE ( title LIKE '%search%' OR isbn = 'search' )
      AND ( a_name = 'autori[0]' OR a_name = 'autori[1]' OR [...] )
      AND ( publisher_name = 'editori[0]' OR publisher_name = 'editori[1]' OR [...] )
      AND ( g_name = 'generi[0]' OR g_name = 'generi[1]' OR [...] )
      AND ( price BETWEEN prezzoMin AND prezzoMax )
      AND ( vote >= votoMin )
      AND ( language = 'linguaIndicata' )
      AND ( stock > 0 )
ORDER BY publication_date DESC
LIMIT libriPerPagina OFFSET libriPerPagina*pagina-libriPerPagina
```

4.2 Ricerca dei libri più venduti in questo mese

Nota: i parametri "oggi" e "meseScorso" sono calcolati a livello di linguaggio Java.

```
SELECT order_id, created, book_isbn, COUNT(*) AS n
FROM OrderView
JOIN Order_has_Book ON (order_id = id )
WHERE (created >= 'meseScorso')
      AND (created < 'oggi')
GROUP BY book_isbn
ORDER BY n LIMIT 9;
```

5 Operazioni sul database

Vengono qui presentate alcune operazioni di inserimento importanti svolte sul database.

5.1 Inserimento di un utente

Nota: i parametri inseriti sono passati a livello di linguaggio Java.

```
INSERT INTO User (id, email, name, surname, password, is_admin)
VALUES ('id', 'email', 'nome', 'cognome', 'password', amministratore);
```

5.2 Inserimento di un libro

L'inserimento di un libro coinvolge molte tabelle ed è esplicativo del metodo che abbiamo seguito per la gestione di editori ed autori: quando si inserisce un libro, viene controllato il nome di ogni autore e, se questo non è presente nel database, viene automaticamente aggiunto (per l'editore funziona allo stesso modo).

-- Controllo ed inserimento dell'editore --

```
SELECT id
FROM publisher
WHERE (fl_active='S') AND ( name = 'nomeEditore');
```

```
INSERT INTO publisher (id, name)
VALUES (id, 'nomeEditore');
```

-- Inserimento del libro --

```
INSERT INTO book (coverURI, title, description, pages, price, publication_date, stock, isbn,
language, publisher_id, timestamp)
VALUES ('coverURL', 'titolo', 'descrizione', numeroPAgine, prezzo, 'dataPubblicazione', stock, 'isbn',
'lingua', idEditore, DEFAULT);
```

-- Controllo ed inserimento degli autori (ognuna delle operazioni viene svolta una volta per ogni autore) --

```
SELECT id, name
FROM author
WHERE (fl_active='S') AND (name = 'nomeAutore');
```

```
INSERT INTO author (id, name)
VALUES (idAutore, 'nomeAutore');
```

```
INSERT INTO Book_has_author (book_isbn, author_id)
VALUES ('bookIsbn', idAutore);
```

-- Recupero gli id dei generi e li inserisco nella tabella che rappresenta (una volta per ogni genere selezionato) --

```
SELECT *
FROM genre
WHERE (fl_active='S') AND (name = 'nomeGenere');
```

```
INSERT INTO Book_has_genre (book_isbn, genre_id)
VALUES ('bookIsbn', idGenere);
```

Nota: alcune operazioni (come recuperare l'id del genere) vengono eseguite a livello di linguaggio Java.

6 Conclusioni

Durante lo svolgimento del progetto abbiamo cambiato numerose volte il database, per ripensamenti sulle funzionalità o perché abbiamo pensato nuove soluzioni, sempre cercando di cambiare il meno possibile il codice già scritto ai livelli più alti.

In conclusione, possiamo affermare che il database si è rivelato molto più dinamico di quanto pensassimo nella sua struttura, mentre nel funzionamento non abbiamo incontrato particolari problemi o difficoltà, se non nel riuscire a ordinare correttamente le operazioni da svolgere (come si può vedere dall'esempio sull'inserimento del libro, dove è necessario che l'editore venga inserito prima del libro per il vincolo di integrità referenziale).