UNIVERSTITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"
Scuola Politecnica e Delle Scienze di Base
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea in Informatica
Progetto di Basi di Dati

Progettazione e sviluppo di una Base di Dati relazionale per la gestione di una biblioteca online

Mario Penna N86003308
Simone Parente Martone N86004297
Davide Santi N86004773

Indice

1 Requisiti identificati **4**

2 Progettazione concettuale **6**
2.1 Class Diagram 6
2.2 Analisi della ristrutturazione del Class Diagram 6
2.2.1 Analisi delle ridondanze 6
2.2.2 Analisi degli identificativi 7
2.2.3 Rimozione degli attributi multivalore 7
2.2.4 Rimozione degli attributi composti 7
2.2.5 Partizione/Accorpamento delle associazioni 7
2.2.6 Rimozione delle gerarchie 8
2.3 Class Diagram ristrutturato 8
2.3.1 UML Diagram 8
2.3.2 ER Diagram 9
2.4 Dizionario delle classi 9
2.5 Dizionario delle associazioni 11

3 Schema logico **12**

4 Schema Fisico **14**
4.1 Creazione Tabelle 14
4.1.1 Tabella Articoli 14
4.1.2 Tabella Autore 14
4.1.3 Tabella AutoreArticolo 15
4.1.4 Tabella Riviste 15
4.1.5 Tabella ArticoliInRiviste 15
4.1.6 Tabella Evento 16
4.1.7 Tabella Conferenza 16
4.1.8 Tabella Libri 16
4.1.9 Tabella AutoreLibro 17
4.1.10 Tabella Presentazione 17
4.1.11 Tabella Serie 17
4.1.12 Tabella LibriInSerie 18
4.1.13 Tabella Negozio 18

4.1.14 Tabella Stock 18
4.1.15 Tabella Utente 18
4.1.16 Tabella Richiesta 19
4.1.17 Tabella Jolly 19
4.2 Creazione Funzioni 19
4.2.1 Procedura inserimento Autori 19
4.2.2 Funzione Disponibilità Libro 20
4.2.3 Funzione Disponibilità Serie 20
4.3 Trigger Gestione Articoli 21
4.3.1 Inserimento Articolo e Rivista 21
4.3.2 Inserimento Articolo e Conferenza 23
4.3.3 Rimozione Articolo 25
4.4 Trigger Gestione Libri 26
4.4.1 Inserimento Libro 27
4.4.2 Inserimento Presentazione di un libro 28
4.4.3 Rimozione Libro 30
4.5 Trigger Gestione Stock 31
4.6 Gestione Notifiche 32
4.6.1 View Notifiche 33

Capitolo 1

Requisiti identificati

Si vuole sviluppare un sistema informativo di gestione di una biblioteca digitale contenente **Libri** e **Articoli scientifici**.

I libri possono essere **Didattici** o **Romanzi**.

In particolare, questi ultimi possono essere parte di **Collane**, raggruppate per caratteristiche comuni, e appartenere ad una **Serie** se hanno uno o più seguiti, gli articoli possono essere parte di una **Rivista** oppure essere presentati durante una **Conferenza**.

Il sistema dovrà inoltre permettere ad un **Utente** la ricerca di un libro e recuperare la lista di **Negozi** in cui sia possibile acquistare quest'ultimo. L'utente potrà inoltre ricercare una serie (o collana) di libri e un negozio in cui quest'ultima potrà essere acquistata nel caso in cui al momento della ricerca non ci fosse alcun negozio idoneo, l'utente potrà inoltrare una **Richiesta** di notifica nel momento in cui uno dei negozi avrà tutti i libri appartenenti alla serie.

In particolare sono state identificate le seguenti entità:

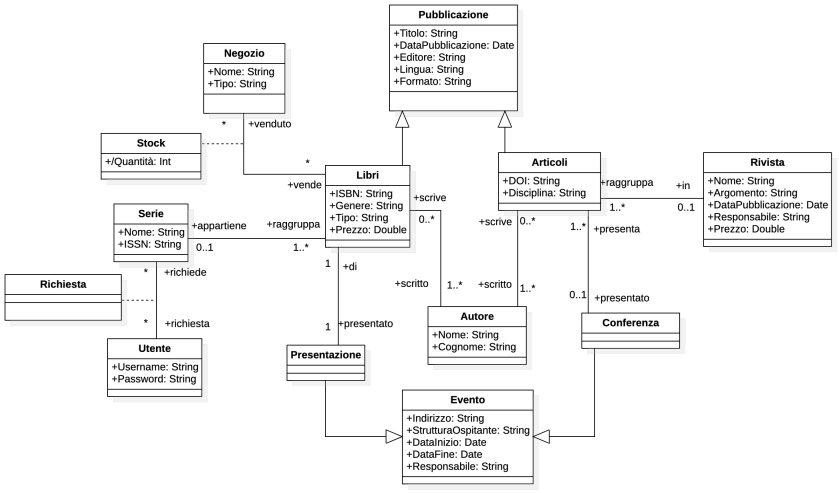
1. **Pubblicazione**: Generalizzazione di un libro o un articolo scientifico.
2. **Libro**: Specializzazione di una Pubblicazione.
3. **Articolo scientifico**: Specializzazione di una Pubblicazione.
4. **Rivista**: Entità che identifica un insieme di articoli.
5. **Evento**: Generalizzazione di una Conferenza o di una Presentazione.
6. **Conferenza**: Specializzazione di un Evento.
7. **Presentazione**: Specializzazione di un Evento.
8. **Autore**: Entità che identifica l'autore di un Libro o di un Articolo.
9. **Negozio**: Entità che identifica un Negozio.
10. **Serie**: Entità che identifica un insieme di libri con caratteristiche simili.

11. **Richiesta**: Entità che identifica la richiesta di disponibilità di una serie da parte di un utente.

Capitolo 2

Progettazione concettuale

2.1 Class Diagram



2.2 Analisi della ristrutturazione del Class Diagram

Durante questa fase, verranno apportate alcune modifiche al diagramma delle classi al fine di renderlo più adatto per una traduzione al modello logico.

2.2.1 Analisi delle ridondanze

L'unica ridondanza presente nel diagramma è l'attributo *DataPubblicazione* nella classe **Rivista**. Dato che un articolo scientifico deve essere pubblicato in una

rivista oppure in una conferenza, abbiamo deciso di rimuovere l'attributo dalla classe **Rivista** e conservare solo quello della classe **Articolo**, che può essere comunque accessibile tramite una JOIN.

2.2.2 Analisi degli identificativi

In questa fase andremo a scegliere un attributo per identificare univocamente le varie entità presenti nello schema precedente, in particolare:

1. L'entità **Libro** presenta l'attributo *ISBN* che rappresenta una possibile chiave primaria, tuttavia è stato scelto di aggiungere un attributo *ID_Libro* in modo tale da aumentare la velocità di accesso agli indici e garantire l'immutabilità della base di dati.

2. Per **Articolo scientifico** la situazione è analoga, è stato quindi aggiunto un attributo *ID_Articolo*.

3. Nel caso dell'entità **Rivista**, la quale presenta un attributo ISSN che è chiave candidata, di inserire un ulteriore attributo *ID_Rivista*.

4. Sarebbe possibile identificare un **Evento** tramite un insieme piuttosto ampio di attributi, è stato quindi aggiunto un attributo *ID_Evento*.

5. Per lo stesso motivo di Evento, è stato aggiunto alla tabella **Autore** un attributo *ID_Autore*.

6. Dato che l'entità **Negozio** non presenta alcuna chiave candidata, è stato aggiunto l'attributo *ID_Negozio*.

7. È stato deciso di aggiungere a **Serie** un attributo *ID_Serie* per ridurre il volume degli indici associati.

2.2.3 Rimozione degli attributi multivalore

Non sono presenti attributi multivalore.

2.2.4 Rimozione degli attributi composti

Non sono presenti attributi composti.

2.2.5 Partizione/Accorpamento delle associazioni

L'unica associazione 1:1 presente in questo Class Diagram è quella tra un **Libro** e una **Presentazione**, nonostante ciò, è stato deciso di mantenere la classe associativa tra le classi perché abbiamo ritenuto più idoneo avere due classi diverse per associare due tipi di pubblicazione diversi a un Evento.

2.2.6 Rimozione delle gerarchie

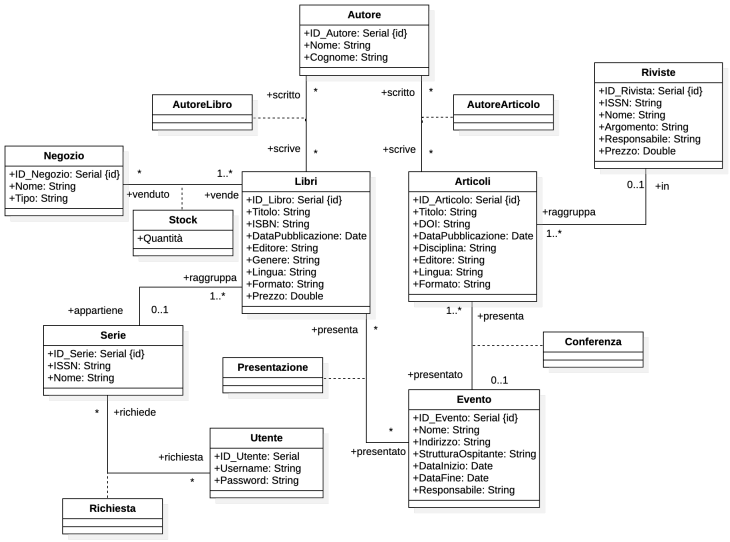
In questo diagramma sono presenti 2 generalizzazioni e 4 relative specializzazioni. In particolare:

Per quanto riguarda la generalizzazione **Pubblicazione**, si è scelto di accorpare l'entità padre nelle entità figlie, ottenendo come risultato:

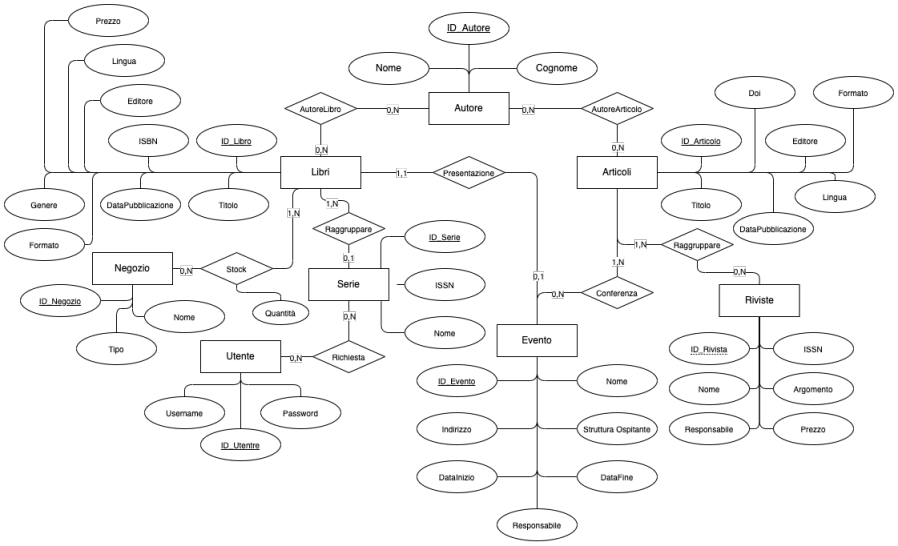
- Una entità **Libro** aventi tutti gli attributi di **Pubblicazione** più gli attributi della precedente entità **Libro**.
- Analogamente, l'entità **Articolo** avrà come attributi, quelli di **Pubblicazione** uniti agli attributi di **Articolo**.

2.3 Class Diagram ristrutturato

2.3.1 UML Diagram



2.3.2 ER Diagram



2.4 Dizionario delle classi

Classe	Attributi	Descrizione
Articolo	ID_Articolo Titolo Doi DataPubblicazione Disciplina Editore Lingua Formato	Classe che conserva le informazioni relative agli articoli presenti nel sistema
Autore	ID_Autore Nome Cognome	Classe che conserva le informazioni relative agli autori presenti nel sistema
Rivista	ID_Rivista ISSN Nome Argomento Responsabile Prezzo	Classe che conserva le informazioni relative alle riviste scientifiche presenti nel sistema

Classe	Attributi	Descrizione
AutoreArticolo		Classe che conserva le informazioni sugli autori degli articoli scientifici
ArticoloInRivista		Classe che mette in relazione Articoli e Riviste a cui i primi appartengono
Evento	ID_Evento Indirizzo StrutturaOspitante DataInizio DataFine Responsabile	Classe che conserva le informazioni sugli eventi relativi a presentazioni di libri oppure a conferenze scientifiche
Conferenza		Classe che mette in relazione eventi e articoli scientifici
Libro	ID_Libro Titolo ISBN DataPubblicazione Editore Genere Lingua Formato Prezzo	Classe che conserva le principali informazioni sui libri presenti nel sistema
AutoreLibro		Classe che mette in relazione Autori e Libri
Presentazione		Classe che conserva le informazioni relative a presentazioni di libri
Serie	ID_Serie ISSN Nome	Classe che conserva le informazioni relative alle Serie di libri
Negozio	ID_Negozio Nome Tipo	Classe che conserva le informazioni relative ai Negozi presenti nel sistema
Stock	Quantità	Classe che conserva le informazioni relative ai libri acquistabili da determinati negozi e la relativa quantità

Classe	Attributi	Descrizione
Utente	ID_Utente Username Password	Classe che conserva le informazioni relative agli Utenti registrati sul sistema
Richiesta		Classe che gestisce le richieste fatte dagli utenti relativamente alle disponibilità di serie

2.5 Dizionario delle associazioni

Associazione	Classi coinvolte	Descrizione
scritto ... scrive	Libri e Autore Articoli e Autore.	Uno o più Libri/Articoli vengono scritti da uno o più Autori. [*, *]
venduto ... vende	Negozio, Stock, Libri.	Uno o più negozi vendono uno o più libri. [*, 1..*].
raggruppa ... in	Riviste e Articoli.	Almeno un articolo è pubblicato in una rivista. Una rivista raggruppa diversi articoli. [1..*, 0..1]
presentato ... presenta	Articoli, Conferenza, Evento. Libri, Presentazione, Evento.	Uno o più articoli possono essere presentati in una conferenza. Un libro può essere presentato durante un evento. [0..1, 1...*]
raggruppa ... appartiene	Libri, Serie	Una serie raggruppa diversi libri, un libro può appartenere al più a una serie. [1..*, 0..1]
richiede ... richiesta	Utente, Serie	Un utente può richiedere diverse serie. Una serie può essere richiesta da diversi utenti. [*..*]

Capitolo 3

Schema logico

Articolo: ID_Articolo, Titolo, DOI, DataPubblicazione, Editore, Lingua, Formato

Autore: ID_Autore, Nome, Cognome

Rivista: ID_Rivista, ISSN, Nome, Argomento, Responsabile, Prezzo

AutoreArticolo: *ID_Autore, ID_Articolo*

ID_Autore $\hookrightarrow$ Autore(ID_Autore)
ID_Articolo $\hookrightarrow$ Articolo(ID_Articolo)

ArticoloInRivista: *ID_Rivista, ID_Articolo*

ID_Rivista $\hookrightarrow$ Rivista(ID_Rivista)
ID_Articolo $\hookrightarrow$ Articolo(ID_Articolo)

Evento: ID_Evento, Indirizzo, StrutturaOspitante, DataInizio, DataFine, Responsabile

Conferenza: *ID_Evento, ID_Articolo*

ID_Evento $\hookrightarrow$ Evento(ID_Evento)
ID_Articolo $\hookrightarrow$ Articolo(ID_Articolo)

Libro: ID_Libro, Titolo, ISBN, DataPubblicazione, Editore, Genere, Lingua, Formato, Prezzo

AutoreLibro: *ID_Autore, ID_Libro*

ID_Autore $\hookrightarrow$ Autore(ID_Autore)
ID_Libro $\hookrightarrow$ Libro(ID_Libro)

Presentazione: *ID_Evento, ID_Libro*

ID_Evento $\hookrightarrow$ Evento(ID_Evento)
ID_Libro $\hookrightarrow$ Libro(ID_Libro)

Serie: ID_Serie, ISSN, Nome

LibroInSerie: ID_Serie, *ID_Libro*

ID_Serie $\hookrightarrow$ Serie(ID_Serie)
ID_Libro $\hookrightarrow$ Libro(ID_Libro)

Negozio: ID_Negozio, Nome, Tipo

Stock: *ID_Negozio, ID_Libro*, Quantità

ID_Negozio $\hookrightarrow$ Negozio(ID_Negozio)
ID_Libro $\hookrightarrow$ Libro(ID_Libro)

Utente: ID_Utente, Username, Password

Richiesta: *ID_Utente, ID_Serie*

ID_Utente $\hookrightarrow$ Utente(ID_Utente)
ID_Serie $\hookrightarrow$ Serie(ID_Serie)

Capitolo 4

Schema Fisico

In questo ultimo capitolo esamineremo i meccanismi necessari per la traduzione di uno schema logico in uno schema fisico. Andremo a definire le tabelle con i relativi attributi e tipi dei dati, le funzioni, le procedure, i trigger e i vincoli. Con questi elementi, sarà possibile creare un database relazionale con una struttura specifica che soddisfi i requisiti identificati nel Capitolo 1.

4.1 Creazione Tabelle

4.1.1 Tabella Articoli

```
CREATE TABLE b.Articoli
ID_Articolo     SERIAL,
Titolo          VARCHAR(128),
DOI             VARCHAR(128),
DataPubblicazione DATE,
Disciplina      VARCHAR(128),
Editore         VARCHAR(128),
Lingua          VARCHAR(128),
Formato         VARCHAR(128),

CONSTRAINT PK_Articoli PRIMARY KEY (ID_Articolo),
CONSTRAINT UK_Articolo UNIQUE (DOI);
```

4.1.2 Tabella Autore

```
CREATE TABLE b.Autore
ID_Autore SERIAL,
Nome     VARCHAR(128),
Cognome  VARCHAR(128),

CONSTRAINT PK_Autore PRIMARY KEY (ID_Autore);
```

4.1.3 Tabella AutoreArticolo

```
    CREATE TABLE b.AutoreArticolo
    ID_Autore  SERIAL,
    ID_Articolo SERIAL,

    CONSTRAINT PK_AutoreArticolo PRIMARY KEY (ID_Autore, ID_Articolo),
    CONSTRAINT FK_AutoreArticolo_Autore FOREIGN KEY (ID_Autore)
        REFERENCES b.Autore (ID_Autore) ON DELETE CASCADE,
    CONSTRAINT FK_AutoreArticolo_Articoli FOREIGN KEY (ID_Articolo)
        REFERENCES b.Articoli (ID_Articolo) ON DELETE CASCADE;
```

4.1.4 Tabella Riviste

```
    CREATE TABLE b.Riviste
    ID_Rivista      SERIAL,
    ISSN            VARCHAR(128),
    Nome            VARCHAR(128),
    Argomento       VARCHAR(128),
    Responsabile    VARCHAR(128),
    Prezzo          FLOAT,

    CONSTRAINT PK_Riviste PRIMARY KEY (ID_Rivista);
```

4.1.5 Tabella ArticoliInRiviste

```
CREATE TABLE b.ArticoliInRiviste
    ID_Articolo SERIAL,
    ID_Rivista SERIAL,

    CONSTRAINT PK_ArticoliInRiviste PRIMARY KEY (ID_Articolo,
        ID_Rivista),
    CONSTRAINT FK_ArticoliInRiviste_Articolo FOREIGN KEY (ID_Articolo)
        REFERENCES b.Articoli (ID_Articolo) ON DELETE CASCADE,
    CONSTRAINT FK_ArticoliInRiviste_Rivista FOREIGN KEY (ID_Rivista)
        REFERENCES b.Riviste (ID_Rivista) ON DELETE CASCADE
```

4.1.6 Tabella Evento

```
    ID_Evento        SERIAL,
    Nome             VARCHAR(128),
    Indirizzo        VARCHAR(128),
    StrutturaOspitante VARCHAR(128),
    DataInizio       DATE,
    DataFine         DATE,
    Responsabile     VARCHAR(128),

    CONSTRAINT PK_Evento PRIMARY KEY (ID_Evento),
    CONSTRAINT CK_Date CHECK (DataInizio <= DataFine),
    CONSTRAINT UK_Evento UNIQUE CONSTRAINT UK_Evento UNIQUE (Nome,
        Indirizzo, DataInizio)
```

4.1.7 Tabella Conferenza

```
    CREATE TABLE b.Conferenza
    ID_Articolo SERIAL,
    ID_Evento  SERIAL,

    CONSTRAINT PK_Conferenza PRIMARY KEY (ID_Articolo, ID_Evento),
    CONSTRAINT FK_Conferenza_Articolo FOREIGN KEY (ID_Articolo)
        REFERENCES b.Articoli (ID_Articolo) ON DELETE CASCADE,
    CONSTRAINT FK_Conferenza_Evento FOREIGN KEY (ID_Evento) REFERENCES
        b.Evento (ID_Evento) ON DELETE CASCADE;
```

4.1.8 Tabella Libri

```
    CREATE TABLE b.Libri
    ID_Libro        SERIAL,
    Titolo          VARCHAR(128),
    ISBN            VARCHAR(128),
    DataPubblicazione DATE,
    Editore         VARCHAR(128),
    Genere          VARCHAR(128),
    Lingua          VARCHAR(128),
    Formato         VARCHAR(128),
    Prezzo          FLOAT,

    CONSTRAINT PK_Libri PRIMARY KEY (ID_Libro),
    CONSTRAINT UK_Libro UNIQUE (ISBN),
    CONSTRAINT CK_Libri CHECK (Prezzo > 0),
    CONSTRAINT CK_Titolo (Titolo IS NOT NULL);
```

4.1.9 Tabella AutoreLibro

```
CREATE TABLE b.AutoreLibro
ID_Autore SERIAL,
ID_Libro SERIAL,

CONSTRAINT PK_AutoreLibro PRIMARY KEY (ID_Autore, ID_Libro),
CONSTRAINT FK_AutoreLibro_Autore FOREIGN KEY (ID_Autore)
    REFERENCES b.Autore (ID_Autore) ON DELETE CASCADE,
CONSTRAINT FK_AutoreLibro_Libro FOREIGN KEY (ID_Libro) REFERENCES
    b.Libri (ID_Libro) ON DELETE CASCADE;
```

4.1.10 Tabella Presentazione

```
CREATE TABLE b.Presentazione
ID_Evento SERIAL,
ID_Libro SERIAL,

CONSTRAINT PK_Presentazione PRIMARY KEY (ID_Evento, ID_Libro),
CONSTRAINT FK_Presentazione_Evento FOREIGN KEY (ID_Evento)
    REFERENCES b.Evento (ID_Evento) ON DELETE CASCADE,
CONSTRAINT FK_Presentazione_Libro FOREIGN KEY (ID_Libro)
    REFERENCES b.Libri (ID_Libro) ON DELETE CASCADE;
```

4.1.11 Tabella Serie

```
CREATE TABLE b.Serie
ID_Serie SERIAL,
ISSN    VARCHAR(128),
Nome    VARCHAR(128),

CONSTRAINT PK_Serie PRIMARY KEY (ID_Serie),
CONSTRAINT UK_Serie UNIQUE (ISSN);
```

4.1.12 Tabella LibriInSerie

```
CREATE TABLE b.LibriInSerie
ID_Serie INTEGER,
ID_Libro INTEGER,

CONSTRAINT PK_LibriInSerie PRIMARY KEY (ID_Serie, ID_Libro),
CONSTRAINT FK_Libri_Serie FOREIGN KEY (ID_Serie) REFERENCES
    b.Serie (ID_Serie) ON DELETE CASCADE,
CONSTRAINT FK_Serie_Libri FOREIGN KEY (ID_Libro) REFERENCES
    b.Libri (ID_Libro) ON DELETE CASCADE;
```

4.1.13 Tabella Negozio

```
CREATE TABLE b.Negozio
ID_Negozio SERIAL,
Nome      VARCHAR(128),
Tipo      VARCHAR(128),

CONSTRAINT PK_Negozio PRIMARY KEY (ID_Negozio);
```

4.1.14 Tabella Stock

```
CREATE TABLE b.Negozio
ID_Negozio SERIAL,
Nome      VARCHAR(128),
Tipo      VARCHAR(128),

CONSTRAINT PK_Negozio PRIMARY KEY (ID_Negozio);
```

4.1.15 Tabella Utente

```
CREATE TABLE b.Utente
ID_Utente SERIAL,
Username VARCHAR(128),
Password VARCHAR(128),

CONSTRAINT PK_Utente PRIMARY KEY (ID_Utente),
CONSTRAINT UK_Utente UNIQUE (Username);
```

TipoUtente

```
CREATE TYPE b.TipoUtente AS ENUM ('0', '1', '2');
```

4.1.16 Tabella Richiesta

```
    CREATE TABLE b.Richiesta
    ID_Utente    SERIAL,
    ID_Serie     SERIAL,

    CONSTRAINT PK_Richiesta PRIMARY KEY (ID_Utente, ID_Serie),
    CONSTRAINT FK_Richiesta_Utente FOREIGN KEY (ID_Utente) REFERENCES
        b.Utente (ID_Utente) ON DELETE CASCADE,
    CONSTRAINT FK_Richiesta_Serie FOREIGN KEY (ID_Serie) REFERENCES
        b.Serie (ID_Serie) ON DELETE CASCADE;
```

4.1.17 Tabella Jolly

La tabella Jolly è una tabella che contiene un solo attributo di tipo text, che permette di inserire una stringa di lunghezza arbitraria negli inserimenti tramite view, se necessario.

```
    CREATE TABLE b.Jolly
    Text TEXT;
```

4.2 Creazione Funzioni

4.2.1 Procedura inserimento Autori

```
CREATE OR REPLACE PROCEDURE b.insAutori(stringAutori text, idRisorsa
    INTEGER, tipoRisorsa INTEGER) AS
$$
DECLARE
    autori       text[] = string_to_array(stringAutori, ' ');
    numAutori    INTEGER = array_length(autori, 1);
    autoreNome   b.autore.nome%TYPE;
    autoreCognome b.autore.cognome%TYPE;
    idAutore     b.autore.id_autore%TYPE;
    BEGIN
    FOR i IN 1..numAutori
        LOOP
            autoreNome = split_part(autori[i], '_', 1);
            autoreCognome = split_part(autori[i], '_', 2);
            IF NOT EXISTS(SELECT * FROM b.autore WHERE nome =
                autoreNome AND cognome = autoreCognome) THEN
                RAISE NOTICE 'Autore non presente, verr\'a inserito';
                INSERT INTO b.autore (nome, cognome) VALUES
                    (autoreNome, autoreCognome);
            END IF;
            idAutore = (SELECT id_autore FROM b.autore WHERE nome =
                autoreNome AND cognome = autoreCognome);
```

```
            IF(tipoRisorsa = 1) THEN
                INSERT INTO b.autorelibro (id_autore, id_libro) VALUES
                    (idAutore, idRisorsa);
            ELSEIF(tipoRisorsa = 0) THEN
                INSERT INTO b.autorearticolo (id_autore, id_articolo)
                    VALUES (idAutore, idRisorsa);
            END IF;
        END LOOP;
END
$$ LANGUAGE plpgsql;
```

4.2.2 Funzione Disponibilità Libro

```
CREATE OR REPLACE FUNCTION b.getDisponibilitaLibro(inputLibro
    b.libri.id_libro%TYPE) RETURNS boolean AS
$$
DECLARE
BEGIN
    IF EXISTS(SELECT * FROM b.stock s WHERE s.id_libro = inputLibro)
        THEN
        return true;
    ELSE
        return false;
    END IF;
END;
$$ language plpgsql;
```

4.2.3 Funzione Disponibilità Serie

```
CREATE OR REPLACE FUNCTION b.getDisponibilitaSerie(inputSerie
    b.Serie.id_serie%TYPE) RETURNS boolean AS
$$
DECLARE
    scorrilibri b.libri.id_libro%TYPE;
    cursoreLibri CURSOR FOR (SELECT id_libro
                             FROM b.libriinserie
                             WHERE id_serie = inputSerie);
BEGIN
    OPEN cursorelibri;
    LOOP
        FETCH cursoreLibri INTO scorrilibri;
        EXIT WHEN NOT FOUND;
        IF NOT b.getDisponibilitaLibro(scorrilibri) THEN
            CLOSE cursoreLibri;
            return false;
        END IF;
    END LOOP;
```

```
    CLOSE cursoreLibri;
    return true;
END;
$$ language plpgsql;
```

4.3 Trigger Gestione Articoli

Per gestire l'inserimento degli articoli, vengono utilizzati due trigger, `trig_ArticoliRivista` e `trig_ArticoliConferenze`, che agiscono sulle view `ins_ArticoliRivista` e `ins_ArticoliConferenza`. Questi trigger verificano che la rivista o la conferenza non siano già presenti nel database e, in caso contrario, provvedono all'inserimento. Successivamente, l'articolo viene inserito e la tabella di relazione tra articoli e riviste o conferenze viene aggiornata. Inoltre, viene richiamata la procedura `insAutori`, che verifica se gli autori sono già presenti nel database e, in caso contrario, procede con il loro inserimento, seguendo l'aggiornamento della tabella di relazione tra articoli ed autori. Per rimuovere gli articoli dal database, viene utilizzato il trigger `trig_RimozioneArticoli`, che agisce sulla tabella `Articoli` in `BEFORE DELETE`. Questo trigger verifica che gli autori dell'articolo non abbiano scritto altro e, in caso affermativo, procede con la loro rimozione. Controlla inoltre che la rivista in cui potrebbe essere stato presentato l'articolo non abbia altri articoli e, in questo caso, provvede alla sua rimozione. La stessa cosa avviene per le conferenze.

4.3.1 Inserimento Articolo e Rivista

```
    --View da dove viene inserito un articolo scientifico e la rivista
        dove \'e stato presentato
    CREATE OR REPLACE VIEW b.ins_ArticoliRivista AS
    SELECT a.doi,
           a.titolo,
           TEXT         as AutoriNome_Cognome, --'nome1 cognome1,
               nome2 cognome2'
           a.datapubblicazione,
           a.disciplina,
           a.editore,
           a.lingua,
           a.formato,
           r.nome       as nomeRivista,
           r.issn       as issnRivista,
           r.argomento  as argomentoRivista,
           r.responsabile as responsabileRivista,
           r.prezzo     as prezzoRivista
    FROM b.Articoli a,
         b.jolly,
         b.riviste r;

```

```
20      --Funzione del trigger
21      CREATE OR REPLACE FUNCTION b.ftrig_ArticoliRivista() RETURNS
            trigger AS
22      $$
23      DECLARE
24          idRivista b.riviste.id_rivista%TYPE;
25          idArticolo INTEGER;
26      BEGIN
27          --Controllo che l'articolo non sia gi\'a presente nel DataBase
28          IF EXISTS(SELECT * FROM b.articoli WHERE doi = NEW.doi) THEN
29              RAISE NOTICE 'Articolo gi\'a presente, non verr\'a
                    inserito';
30          ELSE
31              --Controllo che la rivista non sia gi\'a presente nel
                    DataBase in tal caso la inserisco
32              IF NOT EXISTS(SELECT * FROM b.riviste WHERE issn =
                    NEW.issnRivista) THEN
33                  RAISE NOTICE 'Rivista non presente, verr\'a inserita';
34                  INSERT INTO b.riviste (nome, issn, argomento,
                        datapubblicazione, responsabile, prezzo)
35                  VALUES (NEW.nomeRivista, NEW.issnRivista,
                        NEW.argomentoRivista, NEW.datapubblicazione,
                        NEW.responsabileRivista, NEW.prezzoRivista);
36                  --Controllo che la rivista presente nel database abbia
                        la stessa data di pubblicazione
37              ELSEIF NOT EXISTS(SELECT datapubblicazione
38                                FROM b.riviste
39                                WHERE issn = NEW.issnRivista
40                                  AND datapubblicazione =
                                        NEW.datapubblicazione) THEN
41                  RAISE NOTICE 'Rivista gi\'a presente ma con data di
                        pubblicazione diversa, l''articolo non verr\'a
                        inserito';
42                  RETURN NEW;
43              END IF;
44              --Inserisco l'articolo
45              INSERT INTO b.articoli (doi, titolo, datapubblicazione,
                    disciplina, editore, lingua, formato)
46              VALUES (NEW.doi, NEW.titolo, NEW.datapubblicazione,
                    NEW.disciplina, NEW.editore, NEW.lingua, NEW.formato);
47
48              --Recupero l'id dell'articolo appena inserito
49              idArticolo = (SELECT id_articolo FROM b.articoli WHERE doi
                    = NEW.doi);
50
51              --Inserisco gli autori richiamando la procedura insAutori
52              CALL b.insAutori(NEW.AutoriNome_Cognome, idArticolo, 0);
53
54              --Inserisco l'articolo nella rivista
```

```
55              idRivista = (SELECT id_rivista FROM b.riviste WHERE issn =
                    NEW.issnRivista);
56              INSERT INTO b.articoliInRiviste (id_articolo, id_rivista)
                    VALUES (idArticolo, idRivista);
57          END IF;
58          RETURN NEW;
59      END;
60      $$ LANGUAGE plpgsql;
61
62      --Trigger per l'inserimento di un articolo scientifico e la
            rivista dove \'e stato presentato
63      CREATE OR REPLACE TRIGGER trig_ArticoliRivista
64          INSTEAD OF INSERT
65          ON b.ins_ArticoliRivista
66          FOR EACH ROW
67      EXECUTE FUNCTION b.ftrig_ArticoliRivista();
```

4.3.2 Inserimento Articolo e Conferenza

```
1   --View da dove viene inserito un articolo scientifico e la conferenza
        dove \'e stato presentato
2   CREATE OR REPLACE VIEW b.ins_articoliConferenze AS
3   SELECT a.doi,
4          a.titolo,
5          TEXT               as AutoriNome_Cognome, --'nome1 cognome1
               nome2 cognome2'
6          a.datapubblicazione,
7          a.disciplina,
8          a.editore,
9          a.lingua,
10         a.formato,
11         e.nome             as nomeConferenza,
12         e.indirizzo        as indirizzoConferenza,
13         e.strutturaospitante as strutturaospitanteConferenza,
14         e.datainizio       as datainizioConferenza,
15         e.datafine         as datafineConferenza,
16         e.responsabile     as responsabileConferenza
17  FROM b.Articoli a,
18       b.jolly,
19       b.evento e;
20
21  --Funzione del trigger
22  CREATE OR REPLACE FUNCTION b.ftrig_ArticoliConferenze() RETURNS
        TRIGGER AS
23  $$
24  DECLARE
25      idArticolo  INTEGER;
26      idConferenza b.evento.id_evento%TYPE;
```

```
BEGIN
    --Controllo che l'articolo non sia gi\'a presente nel DataBase
    IF EXISTS(SELECT * FROM b.articoli WHERE doi = NEW.doi) THEN
        RAISE NOTICE 'Articolo gi\'a presente, non verr\'a inserito';
        --Controllo se la data di pubblicazione dell'articolo \'e
            compresa tra la data di inizio e la data di fine della
            conferenza
    ELSEIF (NEW.datapubblicazione < NEW.datainizioConferenza OR
        NEW.datapubblicazione > NEW.datafineConferenza) THEN
        RAISE NOTICE 'La data di pubblicazione dell''articolo non \'e
            compresa tra la data di inizio e la data di fine della
            conferenza, l''articolo non verr\'a inserito';
    ELSE
        --Controllo che la conferenza non sia gi\'a presente nel
            DataBase in tal caso la inserisco
        IF NOT EXISTS(SELECT *
                    FROM b.evento
                    WHERE nome = NEW.nomeConferenza
                      AND indirizzo = NEW.indirizzoConferenza
                      AND datainizio = NEW.dataInizioConferenza) THEN
            RAISE NOTICE 'Conferenza non presente, verr\'a inserita';
            INSERT INTO b.evento (nome, indirizzo, strutturaospitante,
                datainizio, datafine, responsabile)
            VALUES (NEW.nomeConferenza, NEW.indirizzoConferenza,
                NEW.strutturaospitanteConferenza,
                   NEW.datainizioConferenza, NEW.datafineConferenza,
                       NEW.responsabileConferenza);
        END IF;
        --Inserisco l'articolo
        INSERT INTO b.articoli (doi, titolo, datapubblicazione,
            disciplina, editore, lingua, formato)
        VALUES (NEW.doi, NEW.titolo, NEW.datapubblicazione,
            NEW.disciplina, NEW.editore, NEW.lingua, NEW.formato);

        --Recupero l'id dell'articolo appena inserito
        idArticolo = (SELECT id_articolo FROM b.articoli WHERE doi =
            NEW.doi);

        --Inserisco gli autori richiamando la procedura insAutori
        CALL b.insAutori(NEW.AutoriNome_Cognome, idArticolo, 0);

        --Inserisco l'articolo nella conferenza
        idConferenza = (SELECT id_evento
                       FROM b.evento
                       WHERE nome = NEW.nomeConferenza AND indirizzo =
                           NEW.indirizzoConferenza);
        INSERT INTO b.Conferenza (id_articolo, id_evento) VALUES
            (idArticolo, idConferenza);
    END IF;
    RETURN NEW;
```

```
END ;
$$ LANGUAGE plpgsql;

--Trigger per l'inserimento di un articolo scientifico e la conferenza dove \'e stato presentato
CREATE OR REPLACE TRIGGER trig_ArticoliConferenze
    INSTEAD OF INSERT
    ON b.ins_ArticoliConferenze
    FOR EACH ROW
EXECUTE FUNCTION b.ftrig_ArticoliConferenze();
```

4.3.3 Rimozione Articolo

```
    CREATE OR REPLACE FUNCTION b.ftrig_rimozineArticoli() RETURNS trigger AS
$$
DECLARE
    idAutoreArticolo b.autore.id_autore%TYPE;
    idAutoreArticoli CURSOR FOR SELECT id_autore
                            FROM b.autorearticolo
                            WHERE id_articolo = OLD.id_articolo;
    idRivista       b.riviste.id_rivista%TYPE = (SELECT id_rivista
                                                 FROM b.articoliinriviste
                                                 WHERE id_articolo =
                                                     OLD.id_articolo);
    IdConferenza    b.evento.id_evento%TYPE = (SELECT id_evento
                                               FROM b.conferenza
                                               WHERE id_articolo =
                                                   OLD.id_articolo);
BEGIN
    --Rimuovo gli autori se non hanno scritto altri articoli o libri
    OPEN idAutoreArticoli;
    LOOP
        FETCH idAutoreArticoli INTO idAutoreArticolo;
        EXIT WHEN NOT FOUND;
        IF NOT EXISTS(SELECT id_autore
                      FROM b.autorearticolo
                      WHERE id_autore = idAutoreArticolo
                        AND id_articolo <> OLD.id_articolo) THEN
            IF NOT EXISTS(SELECT * FROM b.autorelibro WHERE id_autore =
                idAutoreArticolo) THEN
                DELETE FROM b.autore WHERE id_autore = idAutoreArticolo;
            END IF;
        END IF;
    END LOOP;

    --Rimuovo la Rivista se non ha altri articoli
    IF NOT EXISTS(SELECT *
```

```
                FROM b.articoliinriviste
                WHERE id_articolo <> old.id_articolo
                  AND id_rivista = idRivista) THEN
        DELETE FROM b.riviste WHERE id_rivista = idRivista;
    END IF;

    --Rimuovo Conferenza se non ha altri articoli
    IF NOT EXISTS(SELECT *
                FROM b.conferenza
                WHERE id_articolo <> old.id_articolo
                  AND id_evento = IdConferenza) THEN
        DELETE FROM b.evento WHERE id_evento = IdConferenza;
    END IF;

    CLOSE idAutoreArticoli;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

--Trigger per la rimozione di un articolo scientifico
CREATE TRIGGER trig_rimozioneArticoli
    BEFORE DELETE
    ON b.articoli
    FOR EACH ROW
EXECUTE PROCEDURE b.ftrig_rimozineArticoli();
```

4.4 Trigger Gestione Libri

Il trigger `trig_Libri` agisce sulla view `ins_Libri` per l'inserimento di un libro. Questo trigger controlla se il libro è già presente nel database, e se fa parte di una serie. Se la serie non è presente nel database, la inserisce. Inoltre, riempie la tabella di relazione tra libri e serie, e richiama la procedura `insAutori`, che verifica che gli autori siano già presenti nel database; in caso negativo, li inserisce e riempie la tabella di relazione tra libri ed autori. Per l'inserimento di una possibile presentazione di un libro si usa il trigger `trig_presentazione`, che agisce sulla view `ins_presentazione`. Tale trigger controlla che il libro sia presente nel DB, e se non abbia già abbia una presentazione. In tal caso, aggiunge la presentazione nella tabella `Evento`, riempiendo poi la tabella `Presentazione` (la tabella associativa tra `Libro` ed `Evento`). Per la rimozione dei libri dal DB utilizziamo il trigger `trig_rimozionelibri`, che agisce sulla tabella `Libri` in `BEFORE DELETE`. Questo trigger controlla che gli autori di quel libro non abbiano scritto altro, ed in tal caso rimuove tali autori. Controlla inoltre se la serie in cui è possibile che il libro sia presente non ha altri libri, rimuovendo la serie stessa. Infine, rimuove l'eventuale presentazione di quel libro.

4.4.1 Inserimento Libro

```
--View da dove viene inserito un libro
CREATE OR REPLACE VIEW b.ins_Libri AS
SELECT l.titolo,
       l.ISBN,
       j.TEXT as AutoriNome_Cognome, --'Nome1_Cognome1 Nome2_Cognome2'
       l.datapubblicazione,
       l.Editore,
       l.Genere,
       l.Lingua,
       l.Formato,
       l.Prezzo,
       s.nome as NOME_Serie_di_Appartenenza,
       s.ISSN as ISSN_Serie_di_Appartenenza
FROM b.libri as l,
     b.serie as s,
     b.jolly as j;

--Funzione del trigger per l'inserimento di un libro
CREATE OR REPLACE FUNCTION b.ftrig_Libri() RETURNS TRIGGER AS
$$
DECLARE
    idLibro b.libri.ID_Libro%TYPE;
    idSerie b.serie.ID_Serie%TYPE;
BEGIN
    --Controllo che il libro non sia gi\'a presente nel DataBase
    IF EXISTS(SELECT * FROM b.libri WHERE isbn = NEW.isbn) THEN
        RAISE NOTICE 'Libro gi\'a presente';
    ELSE
        --Controllo che la serie di appartenenza del libro non sia gi\'a presente nel DataBase in tal caso la inserisco
        IF NOT EXISTS(SELECT * FROM b.riviste WHERE issn = NEW.issn_serie_di_appartenenza) THEN
        RAISE NOTICE 'Serie non presente';
        IF NEW.nome_serie_di_appartenenza IS NOT NULL THEN
            INSERT INTO b.serie(nome, issn) values
                (NEW.nome_serie_di_appartenenza,
                NEW.issn_serie_di_appartenenza);
        END IF;
        --Controllo che il formato del libro sia compatibile con la serie gi\'a presente nel DataBase
        ELSEIF NOT EXISTS(SELECT *
                          FROM (b.serie s NATURAL JOIN libriinserie ls)
                              JOIN libri l ON ls.id_libro = l.id_libro
                          WHERE l.formato = NEW.formato) THEN
```

```
            RAISE NOTICE 'Il formato del libro non \'e compatibile
                con la serie, libro non inserito';
            RETURN NEW;
        END IF;
        --Inserisco il libro
        INSERT INTO b.libri (titolo, isbn, datapubblicazione,
            editore, genere, lingua, formato, prezzo)
        VALUES (NEW.titolo, NEW.isbn, NEW.datapubblicazione,
            NEW.editore, NEW.genere, NEW.lingua, NEW.formato,
               NEW.prezzo);
        --Recupero l'id del libro appena inserito
        idLibro = (SELECT id_libro FROM b.libri WHERE isbn =
            NEW.isbn);

        --Inserisco gli autori richiamando la procedura insAutori
        CALL b.insAutori(NEW.autoriNome_cognome, idLibro, 1);

        --Inserisco il libro nella serie
        idSerie = (SELECT id_serie FROM b.serie WHERE issn =
            NEW.issn_serie_di_appartenenza);
        RAISE NOTICE 'idSerie: %', idSerie;
        IF idSerie IS NOT NULL THEN
        INSERT INTO b.libriinserie (id_libro, id_serie) VALUES
            (idLibro, idSerie);
        END IF;
    END IF;
    RETURN NEW;
END
$$ LANGUAGE plpgsql;

--Trigger per l'inserimento di un libro
CREATE OR REPLACE TRIGGER trig_Libri
    INSTEAD OF INSERT
    ON b.ins_libri
    FOR EACH ROW
EXECUTE FUNCTION b.ftrig_Libri();
```

4.4.2 Inserimento Presentazione di un libro

```
--View da dove viene inserito una presentazione
CREATE OR REPLACE VIEW b.ins_presentazione AS
SELECT l.ISBN,
       e.nome,
       e.Indirizzo,
       e.StrutturaOspitante,
       e.DataInizio,
       e.DataFine,
       e.Responsabile
```

```
10      FROM b.evento as e,
11           b.libri as l;
12
13      --Funzione del trigger per l'inserimento di una presentazione
14      CREATE OR REPLACE FUNCTION b.ftrig_presentazione()
15          RETURNS trigger AS
16      $$
17      DECLARE
18      BEGIN
19          IF NOT EXISTS(SELECT * FROM b.libri WHERE isbn = NEW.ISBN)
                THEN --Controllo se il libri esiste
20              RAISE NOTICE 'Il libri non esiste!! Presentazione non
                    inserita';
21          ELSEIF EXISTS(SELECT *
22                        FROM (b.evento as e NATURAL JOIN b.presentazione
                              as p) --Controllo se esiste gi\'a una
                              presentazione per quel libri
23                                JOIN b.libri as l ON p.id_libro =
                                      l.id_libro
24                        WHERE ISBN = NEW.ISBN) THEN
25              RAISE NOTICE 'Esista gi\'a una presentazione per questo
                    libro!! Presentazione non inserita';
26          ELSE --Inserisco la presentazione
27              INSERT INTO b.evento (nome, indirizzo, strutturaospitante,
                    datainizio, datafine,
28                                    responsabile) --Inserisco l'evento
29              VALUES (NEW.nome, NEW.Indirizzo, NEW.StrutturaOspitante,
                    NEW.DataInizio, NEW.DataFine, NEW.Responsabile);
30              INSERT INTO b.presentazione (id_evento, id_libro)
                    --Inserisco la presentazione
31              SELECT e.ID_evento, l.ID_libro --Trasformo l'ISBN in un ID
                    e recupero l'ID dell'evento
32              FROM b.evento e,
33                   b.libri l
34              WHERE l.ISBN = NEW.ISBN
35                AND e.nome = NEW.nome
36                AND e.indirizzo = NEW.Indirizzo
37                AND e.strutturaospitante = NEW.StrutturaOspitante
38                AND e.datainizio = NEW.DataInizio
39                AND e.datafine = NEW.DataFine
40                AND e.responsabile = NEW.Responsabile;
41          END IF;
42          RETURN NEW;
43      END
44      $$
45          language plpgsql;
46
47      --Trigger per l'inserimento di una presentazione
48      CREATE OR REPLACE TRIGGER trig_presentazione
49          INSTEAD OF INSERT
```

```
        ON b.ins_presentazione
        FOR EACH ROW
    EXECUTE FUNCTION b.ftrig_presentazione();
```

4.4.3 Rimozione Libro

```
    CREATE OR REPLACE FUNCTION b.ftrig_rimozineLibri() RETURNS trigger
        AS
$$
DECLARE
    idAutoreLibro b.autore.id_autore%TYPE;
    idAutoriLibri CURSOR FOR (SELECT id_autore
                              FROM b.autorelibro
                              WHERE id_libro = OLD.id_libro);
    idEvento     b.evento.id_evento%TYPE = (SELECT id_evento
                                            FROM b.presentazione
                                            WHERE id_libro =
                                                OLD.id_libro);
    idSerie      b.serie.id_serie%TYPE = (SELECT id_serie
                                          FROM b.libriinserie
                                          WHERE id_libro =
                                              OLD.id_libro);
BEGIN
    --Rimuovo gli autori se non hanno scritto altri articoli o libri
    OPEN idAutoriLibri;
    LOOP
        FETCH idAutoriLibri INTO idAutoreLibro;
        EXIT WHEN NOT FOUND;
        IF NOT EXISTS(SELECT * FROM b.autorelibro WHERE id_autore =
            idAutoreLibro AND id_libro <> OLD.id_libro) THEN
            IF NOT EXISTS(SELECT * FROM b.autorearticolo WHERE
                id_autore = idAutoreLibro) THEN
               DELETE FROM b.autore WHERE id_autore = idAutoreLibro;
            END IF;
        END IF;
    END LOOP;

    --Rimuovo la presentazione del libro
    DELETE FROM b.evento WHERE id_evento = idEvento;

    --Rimuovo la serie se non ha altri libri
    IF NOT EXISTS(SELECT * FROM b.libriinserie WHERE id_serie =
        idSerie AND id_libro <> OLD.id_libro) THEN
        DELETE FROM b.serie WHERE id_serie = idSerie;
    END IF;

    CLOSE idAutoriLibri;
    RETURN NEW;
```

```
END;
$$ LANGUAGE plpgsql;

--Trigger per la rimozione di un libro
CREATE TRIGGER trig_rimozioneLibri
    BEFORE DELETE
    ON b.libri
    FOR EACH ROW
EXECUTE PROCEDURE b.ftrig_rimozineLibri();
```

4.5 Trigger Gestione Stock

Utilizziamo il trigger *trig_stock* che agisce sulla view *ins_stock* per gestire lo stock. Il trigger verifica che il libro sia presente nel database, poi controlla se il libro sia già in stock in quel negozio ed, in tal caso, aggiorna la quantità dello stock. Per rimuovere un libro dallo stock c'è il trigger *trig_RimozioneDaStock* che aggiorna la quantità del libro in stock, e se questa diventa zero elimina la tupla riferita a quel libro dalla tabella stock.

```
    --View da dove inserisco i dati per aggiungere un libro allo stock
    CREATE VIEW b.ins_stock AS
    SELECT id_negozio,
           isbn,
           quantita
    FROM b.libri,
         b.stock;

    --Funzione del trigger per lo stock di un negozio
    CREATE OR REPLACE FUNCTION b.ftrig_stock() RETURNS TRIGGER AS
    $$
    DECLARE
        idLibro b.libri.id_libro%TYPE = (SELECT id_libro
                                         FROM b.libri
                                         WHERE isbn = NEW.isbn);
    BEGIN
        --Controllo se il libro \'e presente nel database
        IF NOT EXISTS(SELECT * FROM b.libri WHERE isbn = NEW.isbn) THEN
            RAISE NOTICE 'Libro non presente, inserimento non
                possibile';
            --Controllo se il negozio \'e presente nel database
        ELSEIF NOT EXISTS(SELECT * FROM b.negozio WHERE id_negozio =
            NEW.id_negozio) THEN
            RAISE NOTICE 'Negozio non presente, inserimento non
                possibile';
        ELSE
            --Controllo se il libro non \'e presente nello stock del
                negozio ed in tal caso lo inserisco
```

```
25              IF NOT EXISTS(SELECT * FROM b.stock WHERE id_negozio =
                    NEW.id_negozio AND id_libro = idLibro) THEN
26                  INSERT INTO b.stock (id_negozio, id_libro, quantita)
                        VALUES (NEW.id_negozio, idLibro, NEW.quantita);
27                  --Altrimenti aggiorno la quantit\‘a del libro nello
                        stock del negozio
28              ELSE
29                  UPDATE b.stock
30                  SET quantita = quantita + NEW.quantita
31                  WHERE id_negozio = NEW.id_negozio AND id_libro =
                        idLibro;
32              END IF;
33          END IF;
34          RETURN NEW;
35      END;
36      $$ language plpgsql;
37
38      CREATE OR REPLACE TRIGGER trig_Stock
39          INSTEAD OF INSERT
40          ON b.ins_stock
41          FOR EACH ROW
42      EXECUTE FUNCTION b.ftrig_stock();
43
44      --Funzione del trigger per l’aggiornamento dello stock di un
            negozio
45      CREATE OR REPLACE FUNCTION b.ftrig_RimozioneDaStock() RETURNS
            trigger AS
46      $$
47      BEGIN
48          --Controllo se la quantit\‘a \‘e 0 e in tal caso rimuovo il
                libro dallo stock
49          if (NEW.quantita = 0) then
50              DELETE FROM b.stock WHERE id_libro = OLD.id_libro;
51          end if;
52      END;
53      $$ language plpgsql;
54
55      --Trigger per l’aggiornamento dello stock di un negozio
56      CREATE TRIGGER trig_RimozioneDaStock
57          AFTER UPDATE OF quantita
58          ON b.stock
59          FOR EACH ROW
60      EXECUTE FUNCTION b.ftrig_RimozioneDaStock();
```

4.6 Gestione Notifiche

L’utente può richiedere la disponibilità di una serie, riempiendo la tabella *richieste*. Qualora l’utente debba essere notificato, si utilizza la view *notifiche*.

4.6.1 View Notifiche

```
1  CREATE VIEW b.notifiche AS
2  SELECT *, b.getDisponibilitaSerie(id_serie) AS disponibilita
3  FROM b.serie NATURAL JOIN b.richiesta
4  WHERE b.getDisponibilitaSerie(id_serie) IS true;
```