

UNIVERSITÀ 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
2.4	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	29
4.4.3	Rimozione Libro	30
4.5	Trigger Gestione Stock	31
4.6	Gestione Notifiche	33
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:

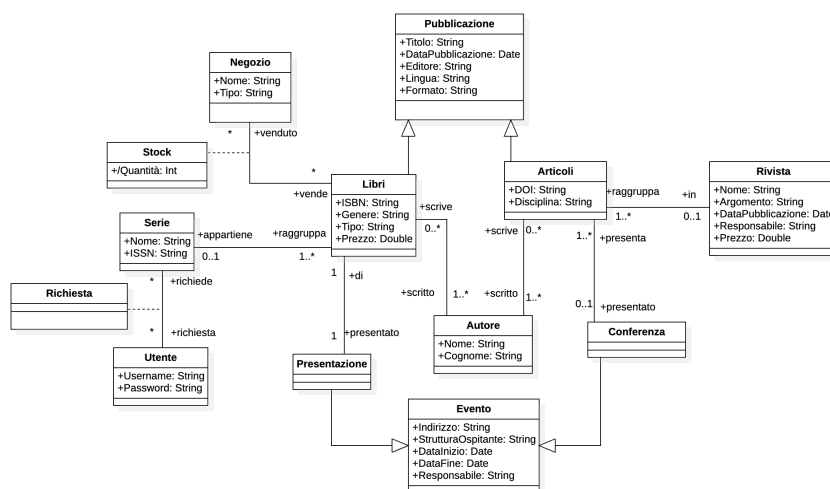
1. **Pubblicazione**: Generalizzazione di un libro o un articolo scientifico.
2. **Libro**: Specializzazione di una Pubblicazione.
3. **Articolo scientifico**: Specializzazione di una Pubblicazione.
4. **Rivista**: Entità che identifica un insieme di articoli.
5. **Evento**: Generalizzazione di una Conferenza 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.

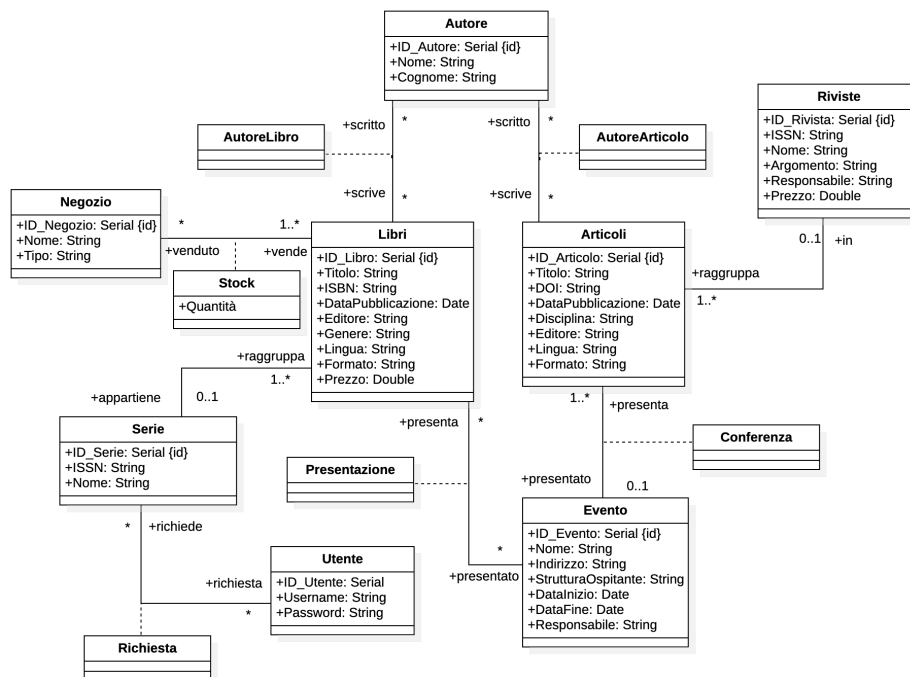
2.2.6 Rimozione delle gerarchie

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

Per quanto riguarda la generalizzazione **Pubblicazione**, si è scelto di accoppiare 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



2.4 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
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

Classe	Attributi	Descrizione
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
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	Negozi, Stock, Libri.	Uno o più negozi vendono uno o più libri. [*,*].
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

```
1 | CREATE TABLE b.Articoli
2 | ID_Articolo SERIAL,
3 | Titolo VARCHAR(128),
4 | DOI VARCHAR(128),
5 | DataPubblicazione DATE,
6 | Disciplina VARCHAR(128),
7 | Editore VARCHAR(128),
8 | Lingua VARCHAR(128),
9 | Formato VARCHAR(128),
10 |
11 | CONSTRAINT PK_Articoli PRIMARY KEY (ID_Articolo),
12 | CONSTRAINT UK_Articolo UNIQUE (DOI);
```

4.1.2 Tabella Autore

```
1 | CREATE TABLE b.Autore
2 | ID_Autore SERIAL,
3 | Nome VARCHAR(128),
4 | Cognome VARCHAR(128),
5 |
6 | CONSTRAINT PK_Autore PRIMARY KEY (ID_Autore);
```

4.1.3 Tabella AutoreArticolo

```
1 CREATE TABLE b.AutoreArticolo
2   ID_Autore SERIAL,
3   ID_Articolo SERIAL,
4
5   CONSTRAINT PK_AutoreArticolo PRIMARY KEY (ID_Autore, ID_Articolo),
6   CONSTRAINT FK_AutoreArticolo_Autore FOREIGN KEY (ID_Autore)
7     REFERENCES b.Autore (ID_Autore) ON DELETE CASCADE,
8   CONSTRAINT FK_AutoreArticolo_Articoli FOREIGN KEY (ID_Articolo)
9     REFERENCES b.Articoli (ID_Articolo) ON DELETE CASCADE;
```

4.1.4 Tabella Riviste

```
1 CREATE TABLE b.Riviste
2   ID_Rivista SERIAL,
3   ISSN VARCHAR(128),
4   Nome VARCHAR(128),
5   Argomento VARCHAR(128),
6   Responsabile VARCHAR(128),
7   Prezzo FLOAT,
8
9   CONSTRAINT PK_Riviste PRIMARY KEY (ID_Rivista);
```

4.1.5 Tabella ArticoliInRiviste

```
1 CREATE TABLE b.ArticoliInRiviste
2   ID_Articolo SERIAL,
3   ID_Rivista SERIAL,
4
5   CONSTRAINT PK_ArticoliInRiviste PRIMARY KEY (ID_Articolo,
6     ID_Rivista),
7   CONSTRAINT FK_ArticoliInRiviste_Articolo FOREIGN KEY (ID_Articolo)
8     REFERENCES b.Articoli (ID_Articolo) ON DELETE CASCADE,
9   CONSTRAINT FK_ArticoliInRiviste_Rivista FOREIGN KEY (ID_Rivista)
10     REFERENCES b.Riviste (ID_Rivista) ON DELETE CASCADE;
```

4.1.6 Tabella Evento

```
1  ID_Evento      SERIAL,
2  Nome           VARCHAR(128),
3  Indirizzo      VARCHAR(128),
4  StrutturaOspitante VARCHAR(128),
5  DataInizio     DATE,
6  DataFine       DATE,
7  Responsabile   VARCHAR(128),
8
9  CONSTRAINT PK_Evento PRIMARY KEY (ID_Evento),
10 CONSTRAINT CK_Date CHECK (DataInizio <= DataFine),
11 CONSTRAINT UK_Nome_Indirizzo_DataInizio UNIQUE (Nome, Indirizzo,
    DataInizio)
```

4.1.7 Tabella Conferenza

```
1  CREATE TABLE b.Conferenza
2  ID_Articolo SERIAL,
3  ID_Evento SERIAL,
4
5  CONSTRAINT PK_Conferenza PRIMARY KEY (ID_Articolo, ID_Evento),
6  CONSTRAINT FK_Conferenza_Articolo FOREIGN KEY (ID_Articolo)
    REFERENCES b.Articoli (ID_Articolo) ON DELETE CASCADE,
7  CONSTRAINT FK_Conferenza_Evento FOREIGN KEY (ID_Evento) REFERENCES
    b.Evento (ID_Evento) ON DELETE CASCADE;
```

4.1.8 Tabella Libri

```
1  CREATE TABLE b.Libri
2  ID_Libro       SERIAL,
3  Titolo         VARCHAR(128),
4  ISBN           VARCHAR(128),
5  DataPubblicazione DATE,
6  Editore        VARCHAR(128),
7  Genere         VARCHAR(128),
8  Lingua         VARCHAR(128),
9  Formato        VARCHAR(128),
10 Prezzo         FLOAT,
11
12 CONSTRAINT PK_Libri PRIMARY KEY (ID_Libro),
13 CONSTRAINT UK_Libro UNIQUE (ISBN),
14 CONSTRAINT CK_Libri CHECK (Prezzo > 0),
15 CONSTRAINT CK_Titolo (Titolo IS NOT NULL);
```


4.1.9 Tabella AutoreLibro

```
1 CREATE TABLE b.AutoreLibro
2   ID_Autore SERIAL,
3   ID_Libro SERIAL,
4
5   CONSTRAINT PK_AutoreLibro PRIMARY KEY (ID_Autore, ID_Libro),
6   CONSTRAINT FK_AutoreLibro_Autore FOREIGN KEY (ID_Autore)
7     REFERENCES b.Autore (ID_Autore) ON DELETE CASCADE,
8   CONSTRAINT FK_AutoreLibro_Libro FOREIGN KEY (ID_Libro) REFERENCES
9     b.Libri (ID_Libro) ON DELETE CASCADE;
```

4.1.10 Tabella Presentazione

```
1 CREATE TABLE b.Presentazione
2   ID_Evento SERIAL,
3   ID_Libro SERIAL,
4
5   CONSTRAINT PK_Presentazione PRIMARY KEY (ID_Evento, ID_Libro),
6   CONSTRAINT FK_Presentazione_Evento FOREIGN KEY (ID_Evento)
7     REFERENCES b.Evento (ID_Evento) ON DELETE CASCADE,
8   CONSTRAINT FK_Presentazione_Libro FOREIGN KEY (ID_Libro)
9     REFERENCES b.Libri (ID_Libro) ON DELETE CASCADE;
```

4.1.11 Tabella Serie

```
1 CREATE TABLE b.Serie
2   ID_Serie SERIAL,
3   ISSN VARCHAR(128),
4   Nome VARCHAR(128),
5
6   CONSTRAINT PK_Serie PRIMARY KEY (ID_Serie),
7   CONSTRAINT UK_Serie UNIQUE (ISSN);
```

4.1.12 Tabella LibriInSerie

```
1 | CREATE TABLE b.LibriInSerie
2 | ID_Serie INTEGER,
3 | ID_Libro INTEGER,
4 |
5 | CONSTRAINT PK_LibriInSerie PRIMARY KEY (ID_Serie, ID_Libro),
6 | CONSTRAINT FK_Libri_Serie FOREIGN KEY (ID_Serie) REFERENCES
   | b.Serie (ID_Serie) ON DELETE CASCADE,
7 | CONSTRAINT FK_Serie_Libri FOREIGN KEY (ID_Libro) REFERENCES
   | b.Libri (ID_Libro) ON DELETE CASCADE;
```

4.1.13 Tabella Negozio

```
1 | CREATE TABLE b.Negozio
2 | ID_Negozio SERIAL,
3 | Nome VARCHAR(128),
4 | Tipo VARCHAR(128),
5 |
6 | CONSTRAINT PK_Negozio PRIMARY KEY (ID_Negozio);
```

4.1.14 Tabella Stock

```
1 | CREATE TABLE b.Negozio
2 | ID_Negozio SERIAL,
3 | Nome VARCHAR(128),
4 | Tipo VARCHAR(128),
5 |
6 | CONSTRAINT PK_Negozio PRIMARY KEY (ID_Negozio);
```

4.1.15 Tabella Utente

```
1 | CREATE TABLE b.Utente
2 | ID_Utente SERIAL,
3 | Username VARCHAR(128),
4 | Password VARCHAR(128),
5 |
6 | CONSTRAINT PK_Utente PRIMARY KEY (ID_Utente),
7 | CONSTRAINT UK_Utente UNIQUE (Username);
```

4.1.16 Tabella Richiesta

```
1 CREATE TABLE b.Richiesta
2   ID_Utente    SERIAL,
3   ID_Serie     SERIAL,
4
5   CONSTRAINT PK_Richiesta PRIMARY KEY (ID_Utente, ID_Serie),
6   CONSTRAINT FK_Richiesta_Utente FOREIGN KEY (ID_Utente) REFERENCES
7     b.Utente (ID_Utente) ON DELETE CASCADE,
8   CONSTRAINT FK_Richiesta_Serie FOREIGN KEY (ID_Serie) REFERENCES
9     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.

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

4.2 Creazione Funzioni

4.2.1 Procedura inserimento Autori

```
1 CREATE OR REPLACE PROCEDURE b.insAutori(stringAutori text, idRisorsa
2   INTEGER, tipoRisorsa INTEGER) AS
3 $$
4 DECLARE
5   autori          text[] = string_to_array(stringAutori, ' ');
6   numAutori       INTEGER = array_length(autori, 1);
7   autoreNome      b.autore.nome%TYPE;
8   autoreCognome   b.autore.cognome%TYPE;
9   idAutore        b.autore.id_autore%TYPE;
10 BEGIN
11   FOR i IN 1..numAutori
12     LOOP
13       autoreNome = split_part(autori[i], '_', 1);
14       autoreCognome = split_part(autori[i], '_', 2);
15       IF NOT EXISTS(SELECT * FROM b.autore WHERE nome =
16         autoreNome AND cognome = autoreCognome) THEN
17         RAISE NOTICE 'Autore non presente, verr\'a inserito';
18         INSERT INTO b.autore (nome, cognome) VALUES
19           (autoreNome, autoreCognome);
20       END IF;
21       idAutore = (SELECT id_autore FROM b.autore WHERE nome =
22         autoreNome AND cognome = autoreCognome);
```

```

19         IF(tipoRisorsa = 1) THEN
20             INSERT INTO b.autorelibro (id_autore, id_libro) VALUES
                (idAutore, idRisorsa);
21         ELSEIF(tipoRisorsa = 0) THEN
22             INSERT INTO b.autorearticolo (id_autore, id_articolo)
                VALUES (idAutore, idRisorsa);
23         END IF;
24     END LOOP;
25 END
26 $$ LANGUAGE plpgsql;

```

4.2.2 Funzione Disponibilità Libro

```

1 CREATE OR REPLACE FUNCTION b.getDisponibilitaLibro(inputLibro
    b.libri.id_libro%TYPE) RETURNS boolean AS
2 $$
3 DECLARE
4 BEGIN
5     IF EXISTS(SELECT * FROM b.stock s WHERE s.id_libro = inputLibro)
        THEN
6         return true;
7     ELSE
8         return false;
9     END IF;
10 END;
11 $$ language plpgsql;

```

4.2.3 Funzione Disponibilità Serie

```

1 CREATE OR REPLACE FUNCTION b.getDisponibilitaSerie(inputSerie
    b.Serie.id_serie%TYPE) RETURNS boolean AS
2 $$
3 DECLARE
4     scorribli b.libri.id_libro%TYPE;
5     cursoreLibri CURSOR FOR (SELECT id_libro
6                                FROM b.libriinserie
7                                WHERE id_serie = inputSerie);
8 BEGIN
9     OPEN cursorelibri;
10    LOOP
11        FETCH cursoreLibri INTO scorribli;
12        EXIT WHEN NOT FOUND;
13        IF NOT b.getDisponibilitaLibro(scorribli) THEN
14            CLOSE cursoreLibri;
15            return false;
16        END IF;
17    END LOOP;

```

```

18 |         CLOSE cursoreLibri;
19 |         return true;
20 |     END;
21 | $$ 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.

Per la gestione l'inserimento degli articoli ci sono due appositi trigger: `trig_ArticoliRivista` e `trig_ArticoliConferenze`, questi trigger si attivano nel momento in cui vengono inseriti dei dati nelle view `ins_ArticoliRivista` e `ins_ArticoliConferenza`.

Questi trigger provvedono a:

- Verificare che la conferenza o la rivista siano gi presenti nel database. In caso contrario, provvede all'inserimento e all'aggiornamento della relativa tabella associativa.
- Verificare che gli autori siano gi presenti nel database. In caso contrario provvede all'inserimento e all'aggiornamento della relativa tabella associativa.

4.3.1 Inserimento Articolo e Rivista

```

1 | --View da dove viene inserito un articolo scientifico e la rivista
2 |   dove \e stato presentato
3 | CREATE OR REPLACE VIEW b.ins_ArticoliRivista AS
4 | SELECT a.doi,
5 |        a.titolo,
6 |        TEXT          as AutoriNome_Cognome, --'nome1 cognome1,
           nome2 cognome2'
           a.datapubblicazione,

```

```

7      a.disciplina,
8      a.editore,
9      a.lingua,
10     a.formato,
11     r.nome      as nomeRivista,
12     r.issn      as issnRivista,
13     r.argomento as argomentoRivista,
14     r.responsabile as responsabileRivista,
15     r.prezzo    as prezzoRivista
16 FROM b.Articoli a,
17      b.jolly,
18      b.riviste r;
19
20 --Funzione del trigger
21 CREATE OR REPLACE FUNCTION b.ftrig_ArticoliRivista() RETURNS
22     trigger AS
23 $$
24 DECLARE
25     idRivista b.riviste.id_rivista%TYPE;
26     idArticolo INTEGER;
27 BEGIN
28     --Controllo che l'articolo non sia gi\ 'a presente nel DataBase
29     IF EXISTS(SELECT * FROM b.articoli WHERE doi = NEW.doi) THEN
30         RAISE NOTICE 'Articolo gi\ 'a presente, non verr\'a
31             inserito';
32     ELSE
33         --Controllo che la rivista non sia gi\ 'a presente nel
34         DataBase in tal caso la inserisco
35         IF NOT EXISTS(SELECT * FROM b.riviste WHERE issn =
36             NEW.issnRivista) THEN
37             RAISE NOTICE 'Rivista non presente, verr\'a inserita';
38             INSERT INTO b.riviste (nome, issn, argomento,
39                 datapubblicazione, responsabile, prezzo)
40                 VALUES (NEW.nomeRivista, NEW.issnRivista,
41                     NEW.argomentoRivista, NEW.datapubblicazione,
42                     NEW.responsabileRivista, NEW.prezzoRivista);
43         --Controllo che la rivista presente nel database abbia
44         la stessa data di pubblicazione
45         ELSEIF NOT EXISTS(SELECT datapubblicazione
46             FROM b.riviste
47             WHERE issn = NEW.issnRivista
48                 AND datapubblicazione =
49                     NEW.datapubblicazione) THEN
50             RAISE NOTICE 'Rivista gi\ 'a presente ma con data di
51                 pubblicazione diversa, l\'articolo non verr\'a
52                 inserito';
53             RETURN NEW;
54         END IF;
55         --Inserisco l'articolo
56         INSERT INTO b.articoli (doi, titolo, datapubblicazione,

```

```

46         disciplina, editore, lingua, formato)
VALUES (NEW.doi, NEW.titolo, NEW.datapubblicazione,
47         NEW.disciplina, NEW.editore, NEW.lingua, NEW.formato);
48
49     --Recupero l'id dell'articolo appena inserito
idArticolo = (SELECT id_articolo FROM b.articoli WHERE doi
50         = NEW.doi);
51
52     --Inserisco gli autori richiamando la procedura insAutori
CALL b.insAutori(NEW.AutoriNome_Cognome, idArticolo, 0);
53
54     --Inserisco l'articolo nella rivista
idRivista = (SELECT id_rivista FROM b.riviste WHERE issn =
55         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
63 rivista dove \e stato presentato
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
2  dove \e stato presentato
CREATE OR REPLACE VIEW b.ins_articoliConferenze AS
3  SELECT a.doi,
4         a.titolo,
5         TEXT          as AutoriNome_Cognome, --'nome1 cognome1
6         nome2 cognome2'
7         a.datapubblicazione,
8         a.disciplina,
9         a.editore,
10        a.lingua,
11        a.formato,
12        e.nome         as nomeConferenza,
13        e.indirizzo    as indirizzoConferenza,
14        e.strutturaospitante as strutturaospitanteConferenza,
15        e.datainizio    as datainizioConferenza,
16        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
23     TRIGGER AS
24 $$
25 DECLARE
26     idArticolo INTEGER;
27     idConferenza b.evento.id_evento%TYPE;
28 BEGIN
29     --Controllo che l'articolo non sia gi\ 'a presente nel DataBase
30     IF EXISTS(SELECT * FROM b.articoli WHERE doi = NEW.doi) THEN
31         RAISE NOTICE 'Articolo gi\ 'a presente, non verr\ 'a inserito';
32         --Controllo se la data di pubblicazione dell'articolo \ 'e
33         compresa tra la data di inizio e la data di fine della
34         conferenza
35     ELSEIF (NEW.datapubblicazione < NEW.datainizioConferenza OR
36             NEW.datapubblicazione > NEW.datafineConferenza) THEN
37         RAISE NOTICE 'La data di pubblicazione dell''articolo non \ 'e
38         compresa tra la data di inizio e la data di fine della
39         conferenza, l''articolo non verr\ 'a inserito';
40     ELSE
41         --Controllo che la conferenza non sia gi\ 'a presente nel
42         DataBase in tal caso la inserisco
43         IF NOT EXISTS(SELECT *
44                        FROM b.evento
45                        WHERE nome = NEW.nomeConferenza
46                        AND indirizzo = NEW.indirizzoConferenza
47                        AND datainizio = NEW.dataInizioConferenza) THEN
48             RAISE NOTICE 'Conferenza non presente, verr\ 'a inserita';
49             INSERT INTO b.evento (nome, indirizzo, strutturaospitante,
50                                  datainizio, datafine, responsabile)
51             VALUES (NEW.nomeConferenza, NEW.indirizzoConferenza,
52                     NEW.strutturaospitanteConferenza,
53                     NEW.datainizioConferenza, NEW.datafineConferenza,
54                     NEW.responsabileConferenza);
55         END IF;
56         --Inserisco l'articolo
57         INSERT INTO b.articoli (doi, titolo, datapubblicazione,
58                                 disciplina, editore, lingua, formato)
59         VALUES (NEW.doi, NEW.titolo, NEW.datapubblicazione,
60                 NEW.disciplina, NEW.editore, NEW.lingua, NEW.formato);
61
62         --Recupero l'id dell'articolo appena inserito
63         idArticolo = (SELECT id_articolo FROM b.articoli WHERE doi =
64                        NEW.doi);
65

```



```

53      --Inserisco gli autori richiamando la procedura insAutori
54      CALL b.insAutori(NEW.AutoriNome_Cognome, idArticolo, 0);
55
56      --Inserisco l'articolo nella conferenza
57      idConferenza = (SELECT id_evento
58                      FROM b.evento
59                      WHERE nome = NEW.nomeConferenza AND indirizzo =
                        NEW.indirizzoConferenza);
60      INSERT INTO b.Conferenza (id_articolo, id_evento) VALUES
        (idArticolo, idConferenza);
61      END IF;
62      RETURN NEW;
63  END ;
64  $$ LANGUAGE plpgsql;
65
66  --Trigger per l'inserimento di un articolo scientifico e la conferenza
67  --dove \e stato presentato
68  CREATE OR REPLACE TRIGGER trig_ArticoliConferenze
69  INSTEAD OF INSERT
70  ON b.ins_ArticoliConferenze
71  FOR EACH ROW
  EXECUTE FUNCTION b.ftrig_ArticoliConferenze();

```

4.3.3 Rimozione Articolo

```

1      CREATE OR REPLACE FUNCTION b.ftrig_rimozineArticoli() RETURNS
      trigger AS
2  $$
3  DECLARE
4      idAutoreArticolo b.autore.id_autore%TYPE;
5      idAutoreArticoli CURSOR FOR SELECT id_autore
6                                  FROM b.autorearticolo
7                                  WHERE id_articolo = OLD.id_articolo;
8      idRivista        b.riviste.id_rivista%TYPE = (SELECT id_rivista
9                                                    FROM b.articoliiinriviste
10                                                    WHERE id_articolo =
11                                                       OLD.id_articolo);
12      IdConferenza     b.evento.id_evento%TYPE = (SELECT id_evento
13                                                    FROM b.conferenza
14                                                    WHERE id_articolo =
15                                                       OLD.id_articolo);
16  BEGIN
17      --Rimuovo gli autori se non hanno scritto altri articoli o libri
18      OPEN idAutoreArticoli;
19      LOOP
20          FETCH idAutoreArticoli INTO idAutoreArticolo;
          EXIT WHEN NOT FOUND;
          IF NOT EXISTS(SELECT id_autore

```

```

21         FROM b.autorearticolo
22         WHERE id_autore = idAutoreArticolo
23         AND id_articolo <> OLD.id_articolo) THEN
24     IF NOT EXISTS(SELECT * FROM b.autorelibro WHERE id_autore =
25         idAutoreArticolo) THEN
26         DELETE FROM b.autore WHERE id_autore = idAutoreArticolo;
27     END IF;
28 END IF;
29
30 --Rimuovo la Rivista se non ha altri articoli
31 IF NOT EXISTS(SELECT *
32     FROM b.articoliiinriviste
33     WHERE id_articolo <> old.id_articolo
34     AND id_rivista = idRivista) THEN
35     DELETE FROM b.riviste WHERE id_rivista = idRivista;
36 END IF;
37
38 --Rimuovo Conferenza se non ha altri articoli
39 IF NOT EXISTS(SELECT *
40     FROM b.conferenza
41     WHERE id_articolo <> old.id_articolo
42     AND id_evento = IdConferenza) THEN
43     DELETE FROM b.evento WHERE id_evento = IdConferenza;
44 END IF;
45
46 CLOSE idAutoreArticoli;
47 RETURN NEW;
48 END;
49 $$ LANGUAGE plpgsql;
50
51 --Trigger per la rimozione di un articolo scientifico
52 CREATE TRIGGER trig_rimozioneArticoli
53 BEFORE DELETE
54 ON b.articoli
55 FOR EACH ROW
56 EXECUTE PROCEDURE b.ftrig_rimozioneArticoli();

```

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 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

```

1  --View da dove viene inserito un libro
2  CREATE OR REPLACE VIEW b.ins_Libri AS
3  SELECT l.titolo,
4         l.ISBN,
5         j.TEXT as AutoriNome_Cognome, --'Nome1_Cognome1
           Nome2_Cognome2'
6         l.datapubblicazione,
7         l.Editore,
8         l.Genere,
9         l.Lingua,
10        l.Formato,
11        l.Prezzo,
12        s.nome as NOME_Serie_di_Appartenenza,
13        s.ISSN as ISSN_Serie_di_Appartenenza
14  FROM b.libri as l,
15        b.serie as s,
16        b.jolly as j;
17
18  --Funzione del trigger per l'inserimento di un libro
19  CREATE OR REPLACE FUNCTION b.ftrig_Libri() RETURNS TRIGGER AS
20  $$
21  DECLARE
22      idLibro b.libri.ID_Libro%TYPE;
23      idSerie b.serie.ID_Serie%TYPE;
24  BEGIN
25      --Controllo che il libro non sia gi\`a presente nel DataBase
26      IF EXISTS(SELECT * FROM b.libri WHERE isbn = NEW.isbn) THEN
27          RAISE NOTICE 'Libro gi\`a presente';
28      ELSE
29          --Controllo che la serie di appartenenza del libro non sia
           gi\`a presente nel DataBase in tal caso la inserisco
30          IF NOT EXISTS(SELECT * FROM b.riviste WHERE issn =
                NEW.issn_serie_di_appartenenza) THEN
31              RAISE NOTICE 'Serie non presente';
32              IF NEW.nome_serie_di_appartenenza IS NOT NULL THEN

```

```

33         INSERT INTO b.serie(nome, issn) values
           (NEW.nome_serie_di_appartenenza,
            NEW.issn_serie_di_appartenenza);
34     END IF;
35     --Controllo che il formato del libro sia compatibile
           con la serie gi\`a presente nel DataBase
36 ELSEIF NOT EXISTS(SELECT *
37                     FROM (b.serie s NATURAL JOIN b.libriinserie
                           ls)
38                     JOIN b.libri l ON ls.id_libro =
                           l.id_libro
39                     WHERE l.formato = NEW.formato) THEN
40         RAISE NOTICE 'Il formato del libro non \`e compatibile
           con la serie, libro non inserito';
41     RETURN NEW;
42 END IF;
43 --Inserisco il libro
44 INSERT INTO b.libri (titolo, isbn, datapubblicazione,
           editore, genere, lingua, formato, prezzo)
45 VALUES (NEW.titolo, NEW.isbn, NEW.datapubblicazione,
           NEW.editore, NEW.genere, NEW.lingua, NEW.formato,
46           NEW.prezzo);
47 --Recupero l'id del libro appena inserito
48 idLibro = (SELECT id_libro FROM b.libri WHERE isbn =
           NEW.isbn);
49
50 --Inserisco gli autori richiamando la procedura insAutori
51 CALL b.insAutori(NEW.autoriNome_cognome, idLibro, 1);
52
53 --Inserisco il libro nella serie
54 idSerie = (SELECT id_serie FROM b.serie WHERE issn =
           NEW.issn_serie_di_appartenenza);
55 RAISE NOTICE 'idSerie: %', idSerie;
56 IF idSerie IS NOT NULL THEN
57     INSERT INTO b.libriinserie (id_libro, id_serie) VALUES
           (idLibro, idSerie);
58 END IF;
59 END IF;
60 RETURN NEW;
61 END
62 $$ LANGUAGE plpgsql;
63
64 --Trigger per l'inserimento di un libro
65 CREATE OR REPLACE TRIGGER trig_Libri
66     INSTEAD OF INSERT
67     ON b.ins_libri
68     FOR EACH ROW
69     EXECUTE FUNCTION b.ftrig_Libri();

```

4.4.2 Inserimento Presentazione di un libro

```
1  --View da dove viene inserito una presentazione
2  CREATE OR REPLACE VIEW b.ins_presentazione AS
3  SELECT l.ISBN,
4         e.nome,
5         e.Indirizzo,
6         e.StrutturaOspitante,
7         e.DataInizio,
8         e.DataFine,
9         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)
20     THEN --Controllo se il libro esiste
21         RAISE NOTICE 'Il libro non esiste!! Presentazione non
22             inserita';
23     ELSEIF EXISTS(SELECT *
24                   FROM (b.evento as e NATURAL JOIN b.presentazione
25                        as p) --Controllo se esiste già una
26                        presentazione per quel libro
27                   JOIN b.libri as l ON p.id_libro =
28                        l.id_libro
29                   WHERE ISBN = NEW.ISBN) THEN
30         RAISE NOTICE 'Esista già una presentazione per questo
31             libro!! Presentazione non inserita';
32     ELSE --Inserisco la presentazione
33         INSERT INTO b.evento (nome, indirizzo, strutturaospitante,
34                               datainizio, datafine,
35                               responsabile) --Inserisco l'evento
36         VALUES (NEW.nome, NEW.Indirizzo, NEW.StrutturaOspitante,
37                 NEW.DataInizio, NEW.DataFine, NEW.Responsabile);
38         INSERT INTO b.presentazione (id_evento, id_libro)
39         --Inserisco la presentazione
40         SELECT e.ID_evento, l.ID_libro --Trasformo l'ISBN in un ID
41             e recupero l>ID dell'evento
42         FROM b.evento e,
43              b.libri l
44         WHERE l.ISBN = NEW.ISBN
45              AND e.nome = NEW.nome
46              AND e.indirizzo = NEW.Indirizzo
47              AND e.strutturaospitante = NEW.StrutturaOspitante
48              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
50     ON b.ins_presentazione
51     FOR EACH ROW
52 EXECUTE FUNCTION b.ftrig_presentazione();

```

4.4.3 Rimozione Libro

```

1     CREATE OR REPLACE FUNCTION b.ftrig_rimozineLibri() RETURNS trigger
2     AS
3     $$
4     DECLARE
5         idAutoreLibro b.autore.id_autore%TYPE;
6         idAutoriLibri CURSOR FOR (SELECT id_autore
7                                     FROM b.autorelibro
8                                     WHERE id_libro = OLD.id_libro);
9         idEvento      b.evento.id_evento%TYPE = (SELECT id_evento
10                                                    FROM b.presentazione
11                                                    WHERE id_libro =
12                                                         OLD.id_libro);
13         idSerie       b.serie.id_serie%TYPE = (SELECT id_serie
14                                                  FROM b.libriinserie
15                                                  WHERE id_libro =
16                                                         OLD.id_libro);
17 BEGIN
18     --Rimuovo gli autori se non hanno scritto altri articoli o libri
19     OPEN idAutoriLibri;
20     LOOP
21         FETCH idAutoriLibri INTO idAutoreLibro;
22         EXIT WHEN NOT FOUND;
23         IF NOT EXISTS(SELECT * FROM b.autorelibro WHERE id_autore =
24                       idAutoreLibro AND id_libro <> OLD.id_libro) THEN
25             IF NOT EXISTS(SELECT * FROM b.autorearticolo WHERE
26                           id_autore = idAutoreLibro) THEN
27                 DELETE FROM b.autore WHERE id_autore = idAutoreLibro;
28             END IF;
29         END IF;
30     END LOOP;
31 END LOOP;
32
33

```

```

27      --Rimuovo la presentazione del libro
28      DELETE FROM b.evento WHERE id_evento = idEvento;
29
30      --Rimuovo la serie se non ha altri libri
31      IF NOT EXISTS(SELECT * FROM b.libriinserie WHERE id_serie =
          idSerie AND id_libro <> OLD.id_libro) THEN
32          DELETE FROM b.series WHERE id_serie = idSerie;
33      END IF;
34
35      CLOSE idAutoriLibri;
36      RETURN NEW;
37  END;
38  $$ LANGUAGE plpgsql;
39
40  --Trigger per la rimozione di un libro
41  CREATE TRIGGER trig_rimozioneLibri
42      BEFORE DELETE
43      ON b.libri
44      FOR EACH ROW
45  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.

```

1      --View da dove inserisco i dati per aggiungere un libro allo stock
2      CREATE VIEW b.ins_stock AS
3      SELECT id_negozio,
4             isbn,
5             quantita
6      FROM b.libri,
7             b.stock;
8
9      --Funzione del trigger per lo stock di un negozio
10     CREATE OR REPLACE FUNCTION b.ftrig_stock() RETURNS TRIGGER AS
11     $$
12     DECLARE
13         idLibro b.libri.id_libro%TYPE = (SELECT id_libro
14                                           FROM b.libri
15                                           WHERE isbn = NEW.isbn);
16     BEGIN
17         --Controlla se il libro \e presente nel database
18         IF NOT EXISTS(SELECT * FROM b.libri WHERE isbn = NEW.isbn) THEN

```

```

19         RAISE NOTICE 'Libro non presente, inserimento non
20             possibile';
21         --Controllo se il negozio \e presente nel database
22         ELSEIF NOT EXISTS(SELECT * FROM b.negozio WHERE id_negozio =
23             NEW.id_negozio) THEN
24             RAISE NOTICE 'Negozio non presente, inserimento non
25                 possibile';
26         ELSE
27             --Controllo se il libro non \e presente nello stock del
28             negozio ed in tal caso lo inserisco
29             IF NOT EXISTS(SELECT * FROM b.stock WHERE id_negozio =
30                 NEW.id_negozio AND id_libro = idLibro) THEN
31                 INSERT INTO b.stock (id_negozio, id_libro, quantita)
32                     VALUES (NEW.id_negozio, idLibro, NEW.quantita);
33             --Altrimenti aggiorno la quantit\`a del libro nello
34             stock del negozio
35             ELSE
36                 UPDATE b.stock
37                 SET quantita = quantita + NEW.quantita
38                 WHERE id_negozio = NEW.id_negozio AND id_libro =
39                     idLibro;
40             END IF;
41         END IF;
42         RETURN NEW;
43     END;
44 $$ language plpgsql;
45
46 CREATE OR REPLACE TRIGGER trig_Stock
47     INSTEAD OF INSERT
48     ON b.ins_stock
49     FOR EACH ROW
50     EXECUTE FUNCTION b.ftrig_stock();
51
52 --Funzione del trigger per l'aggiornamento dello stock di un
53 negozio
54 CREATE OR REPLACE FUNCTION b.ftrig_RimozioneDaStock() RETURNS
55     trigger AS
56 $$
57 BEGIN
58     --Controllo se la quantit\`a \e 0 e in tal caso rimuovo il
59     libro dallo stock
60     if (NEW.quantita = 0) then
61         DELETE FROM b.stock WHERE id_libro = OLD.id_libro;
62     end if;
63 END;
64 $$ language plpgsql;
65
66 --Trigger per l'aggiornamento dello stock di un negozio
67 CREATE TRIGGER trig_RimozioneDaStock
68     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;

```