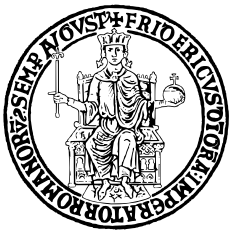
Università degli Studi di Napoli Federico II



Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione

Corso di Laurea Magistrale in Informatica

Progetto d'esame di Basi di Dati

Progettazione ed implementazione di una base di dati relazionale per la gestione di conferenze scientifiche

Relatrice
Professoressa Mara Sangiovanni

Candidati
Antonio Caporaso
matr: N86003458
Giorgio Di Fusco
matr: N86004389

Anno Accademico 2022-2023

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei listati

Capitolo 1

Traccia

Si sviluppi un sistema informativo, composto da una base di dati relazionale e da un applicativo Java dotato di GUI (Swing o JavaFX), per la gestione di **conferenze scientifiche**.

Ogni conferenza ha una data di inizio e di fine, una collocazione (sede, indirizzo), uno o più enti che la organizzano, degli sponsor (che coprono in parte le spese), una descrizione, ed un gruppo di organizzatori, che può essere distinto in comitato scientifico e comitato locale (che si occupa cioè della logistica). Di ognuno degli organizzatori, così come di tutti i partecipanti, si riportano titolo, nome, cognome, email ed istituzione di afferenza.

Ogni conferenza può avere una o più sessioni, anche in parallelo fra loro. Ogni sessione ha una locazione all'interno della sede. Per ogni sessione c'è un programma, che prevede la presenza di un coordinatore (chair) che gestisce la sessione, ed eventualmente di un keynote speaker (un partecipante di particolare rilievo invitato dagli organizzatori). Ogni sessione avrà quindi una successione di interventi ad orari predefiniti e di specifici partecipanti. Per ogni intervento si conserva un abstract (un breve testo in cui viene spiegato il contenuto del lavoro presentato).

Si deve poter considerare la presenza di spazi di intervallo (coffee breaks, pranzo) ma anche la presenza di eventi sociali (cene, gite, etc).

1.1 Output attesi dal committente

1. Documento di Design della base di dati:
   (a) Class Diagram della base di dati.
   (b) Dizionario delle Classi, delle Associazioni e dei Vincoli.
   (c) Schema Logico con descrizione di Trigger e Procedure individuate.
2. File SQL contenenti:
   (a) Creazione della struttura della base di dati.
   (b) Popolamento del DB.
   (c) (Facoltativo, ma apprezzato) README contenente i commenti all'SQL.

Capitolo 2

Progettazione

2.1 Analisi dei dati

Le entità che possono essere individuate nel problema sono elencate all'interno della Tabella 2.1.

2.2 Schema concettuale

Nella Figura 2.1 è presente lo schema concettuale della base di dati descritta nella sezione 1.

2.3 Ristrutturazione dello schema concettuale

2.3.1 Rimozione degli attributi multivalore

All'interno del diagramma delle classi mostrato in Figura 2.1 sono presenti vari attributi multivalore. Per ciascuno di essi sono state fatte le seguenti valutazioni:

1. Si partiziona l'attributo *Indirizzo* presente in SEDE suddividendolo in vari campi *Via*, *Civico*, *Cap*, *City*, *Provincia* e *Nazione* e creando una nuova entità chiamata INDIRIZZO.
2. Si è deciso di partizionare l'attributo *Valuta* presente nella classe di associazione SPONSORIZZAZIONE creando una nuova classe chiamata VALUTA.

2.3.2 Rimozione classi di associazione

All'interno dello schema concettuale è presente la classe di associazione SPONSORIZZAZIONE all'interno dell'associazione [*...*] tra CONFERENZA e SPONSOR. Nello schema ristrutturato questa è stata rimossa reificandola e scindendo l'associazione in due associazioni di tipo [1..*].

2.3.3 Rimozione generalizzazioni

Per quanto riguarda la rimozione delle generalizzazioni presenti nello schema concetuale:

1. Nel caso delle entità COMITATO SCIENTIFICO e COMITATO LOCALE che specializzano la classe COMITATO si è optato per l'accorpamento delle classi figlie all'interno della superclasse attraverso la specifica di una enumerazione chiamata COMITATO_ST composta dai campi *Scientifico* e *Locale*;
2. Nel caso delle entità PRANZO e COFFEE BREAK che specializzano la classe INTERVALLO si è adottato la stessa politica.

Entità	Descrizione
Conferenza	Per le conferenze delle quali si vuole poter gestire le informazioni. Di ogni conferenza si conservano il *nome*, l'*inizio* e la *fine* e una *descrizione*.
Ente	Per gli enti che organizzano le conferenze scientifiche. Di ogni ente si conserva il *nome* e la *sigla*.
Sponsor	Per gli sponsor che coprono le spese della conferenza. Di ogni sponsor si conserva il *nome*.
Comitato	Per i gruppi di organizzatori che si occupano della gestione della conferenza. Si distinguono in comitati *scientifici* e *locali*.
Organizzatore	Per i membri dei comitati. Di ogni organizzatore si riportano *titolo, nome, cognome, email* ed *istituzione di afferenza*.
Sede	Per descrivere il luogo dove si tengono le varie conferenze. Di ogni sede si conservano il *nome*, l'*indirizzo* e la *città*.
Sala	Per tenere traccia dell'ubicazione delle varie sessioni. Di ogni sala si conserva il *nome della sala* e la sua *capacità*.
Sessione	Per rappresentare le sessioni di una conferenza. Per ogni sessione si riporta il *titolo*, un *coordinatore*, data e orario d'*inizio* e di *fine*.
Programma	Per il programma di ciascuna sessione. Ogni programma specifica la presenza di un *keynote speaker*, ovvero un partecipante di rilievo.
Intervento	Per i vari interventi di una sessione. Per ogni intervento si conserva un *abstract*, il partecipante (*speaker*) che effettua l'intervento e l'*orario* dello stesso.
Partecipante	Per i partecipanti delle varie sessioni. Ogni partecipante ha gli stessi attributi degli organizzatori.
Intervallo	Per descrivere i vari intervalli presenti all'interno di una sessione. Questi possono essere di tue tipologie: *coffee break* oppure dei *pranzi*. Per ogni intervallo si riporta l'*orario*.
Evento sociale	Per i vari eventi sociali previsti all'interno di una sessione. Questi possono essere di varia natura. Come per gli intervalli se ne riporta l'*orario*.
Utente	Per i vari utenti che creano le conferenze all'interno di un applicativo.

Tabella 2.1: *Entità del problema*

2.3.4 Scelta degli identificatori principali

Risulta conveniente ai fini di una migliore traduzione delle associazioni l'introduzione di chiavi surrogate per ogni entità. Tali chiavi altro non saranno che identificativi numerici interi del tipo *Id_ NomeEntità*, eccezion fatta per l'entità VALUTA la quale viene identificata univocamente da una stringa di tre caratteri stando allo standard ISO 4217[1].

[1]ISO 4217 è uno standard internazionale che descrive codici di tre lettere per definire i nomi delle valute, stabilito dall'Organizzazione internazionale per la normazione (ISO), che viene usato comunemente nel sistema bancario e nel mondo economico, nonché nella stampa specializzata.

2.4 Progettazione logica

Una volta aver ristrutturato lo schema concettuale mostrato in Figura 2.1 si procede traducendo le varie associazioni descritte in Figura 2.2. Iniziamo col tradurre direttamente tutte le classi. Man mano che si andranno a tradurre le varie associazioni andremo a modificare la struttura dei vari schemi relazionali laddove necessario.

2.4.1 Traduzione delle classi

Si ha quindi:

Indirizzo

Id_Indirizzo	Via	Civico	CAP	City	Provincia	Nazione

Ente

id_ente	nome	sigla

Sede

id_sede	nome

Sponsor

id_Sponsor	Nome

Comitato

id_Comitato	Tipologia

Organizzatore

id_Organizzatore	Nome	Cognome	Titolo	Email

Sala

id_Sala	Nome	Capienza

Conferenza

id_Conferenza	Nome	Descrizione	Inizio	Fine

Partecipante

id_Partecipante	Nome	Cognome	Titolo	Email

Sessione

id_Sessione	Nome	Inizio	Fine

Valuta

Iso	Nome	Simbolo

Speaker

id_Speaker	Nome	Cognome	Titolo	Email

Programma

Id_Programma

Intervallo

id_Intervallo	Tipologia	Inizio	Fine

Intervento

id_Intervento	Titolo	Abstract	Inizio	Fine

Evento

id_Evento	Tipologia	Inizio	Fine

2.4.2 Traduzione delle associazioni

Traduzione delle associazioni molti a molti

Traduciamo le associazioni *..* mediante la realizzazioni di apposite tabelle ponte. Si ha allora:

1. L'associazione EnteConferenza tra Ente e Conferenza:

   EnteConferenza

id_ente	id_conferenza

2. L'associazione OrganizzatoreComitato tra Organizzatore e Comitato:

   OrganizzatoreComitato

id_organizzatore	id_comitato

3. L'associazione PartecipanteSessione tra Partecipante e Sessione:

   PartecipanteSessione

id_Partecipante	id_Sessione

Traduzione delle associazioni uno a molti

Per ciascuna delle associazioni binarie di tipo uno a molti si identificano le entità deboli e quelle forti che partecipano all'associazione. Per tradurre l'associazione in relazioni basterà includere la chiave surrogata dell'entità forte all'interno della relazione dell'entità debole. Avremo quindi:

1. Associazioni di composizione:

   (a) Una sede è composta da più sale quindi:

   Sala

id_Sala	Nome	Capienza	id_sede

   (b) Una conferenza è composta da più sessioni:

   Sessione

id_Sessione	Nome	Inizio	Fine	id_conferenza

   (c) Un programma è composto da interventi, intervalli ed eventi:

   Intervallo

id_Intervallo	Tipologia	Inizio	Fine	id_programma

   Intervento

id_Intervento	Titolo	Abstract	Inizio	Fine	id_programma

   Evento

id_Evento	Tipologia	Inizio	Fine	id_programma

2. Un partecipante, uno speaker ed un organizzatore appartengono ad una istituzione, ovvero un Ente:

   Speaker

id_Speaker	Nome	Cognome	Titolo	Email	id_ente

   Partecipante

id_Speaker	Nome	Cognome	Titolo	Email	id_ente

   Organizzatore

id_Speaker	Nome	Cognome	Titolo	Email	id_ente

3. Ogni intervento ha uno speaker che lo effettua:

   Intervento

id_Intervento	id_speaker	Titolo	Abstract	Inizio	Fine	id_programma

4. Una sala può ospitare più sessioni:

   Sessione

id_Sessione	Nome	Inizio	Fine	id_sala	id_conferenza

5. Una sede può ospitare più conferenze:

   Conferenza

id_Conferenza	Nome	Descrizione	Inizio	Fine	id_sede

6. Una conferenza ha due comitati, uno scientifico ed uno locale;

7. Ogni conferenza ha un utente che la crea;

Conferenza

id_Conferenza	Nome	Descrizione	Inizio	Fine	id_sede	comitato_s	comitato_l	id_utente

Traduzione delle associazioni uno a uno

Si ha:

1. Ogni sessione ha un coordinatore:

   Sessione

id_Sessione	Nome	Inizio	Fine	id_sala	id_conferenza	id_coordinatore

2. Ogni programma si riferisce ad una sessione e ad un keynote speaker:

   Programma

Id_Programma	id_Sessione	id_keynote

3. Ogni sede ha un indirizzo:

   Sede

id_sede	nome	indirizzo

2.4.3 Schema logico

Nella Figura 2.3 è raffigurato lo schema logico risultante.

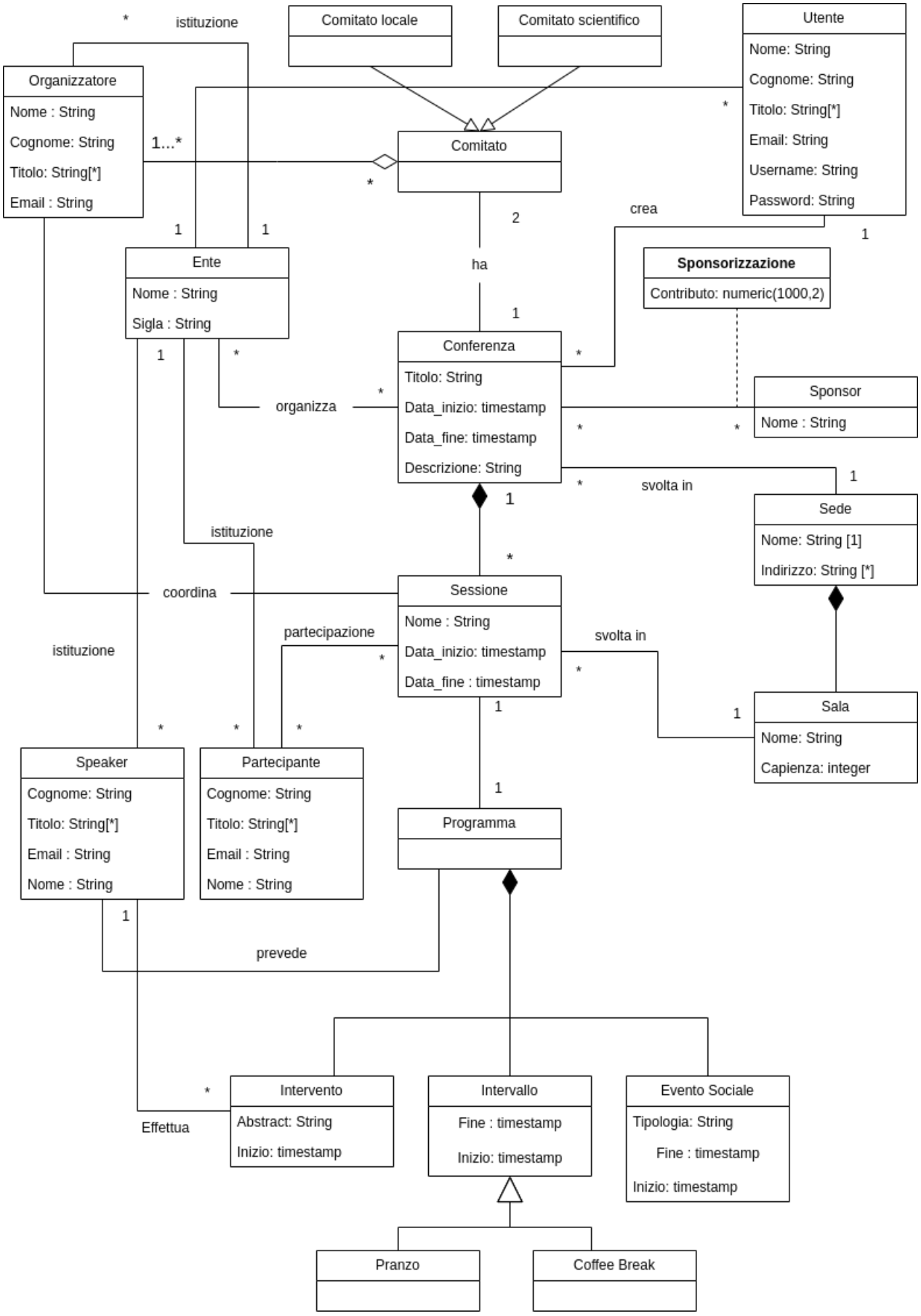


Figura 2.1: *Schema concettuale del problema*

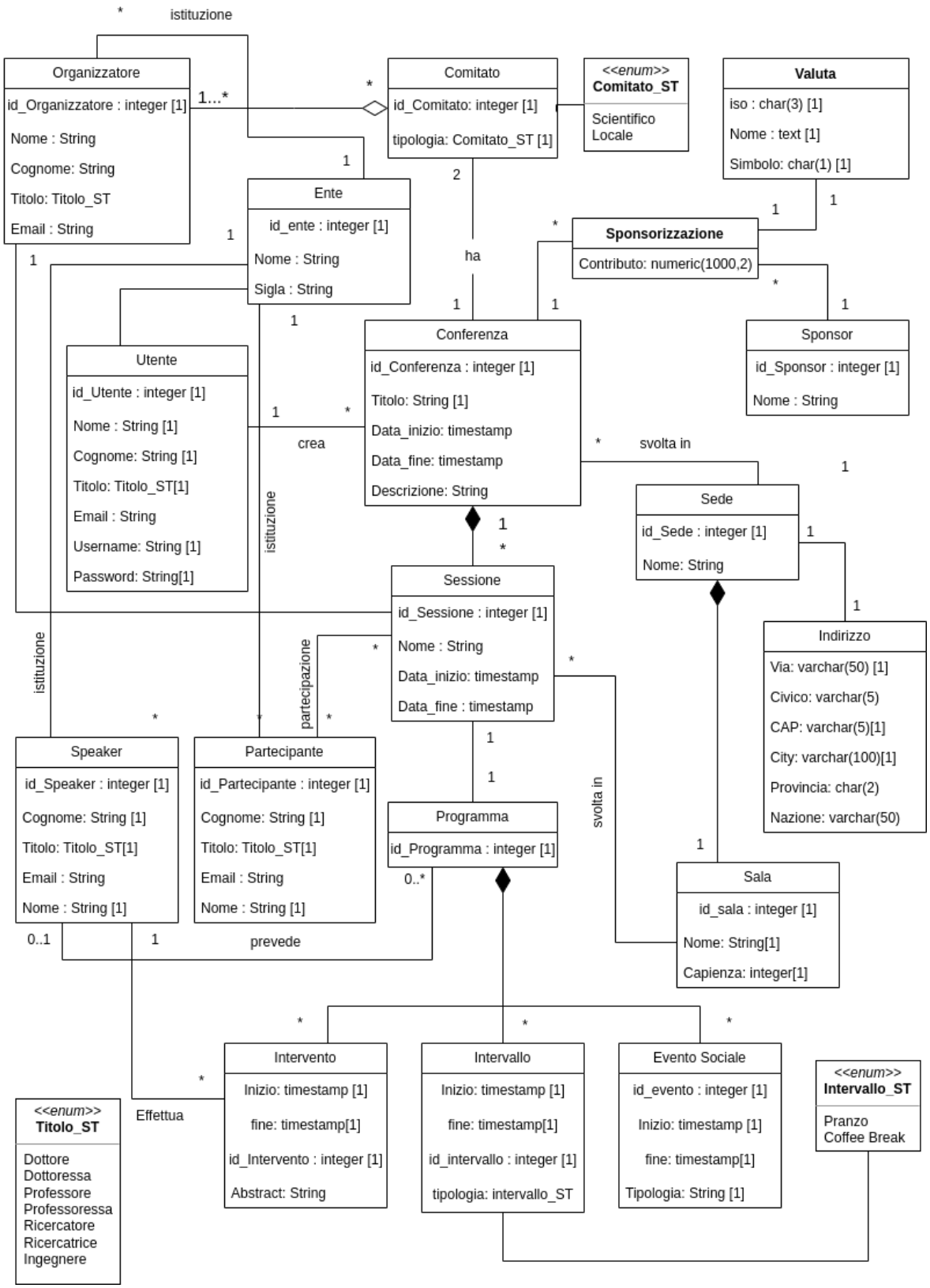
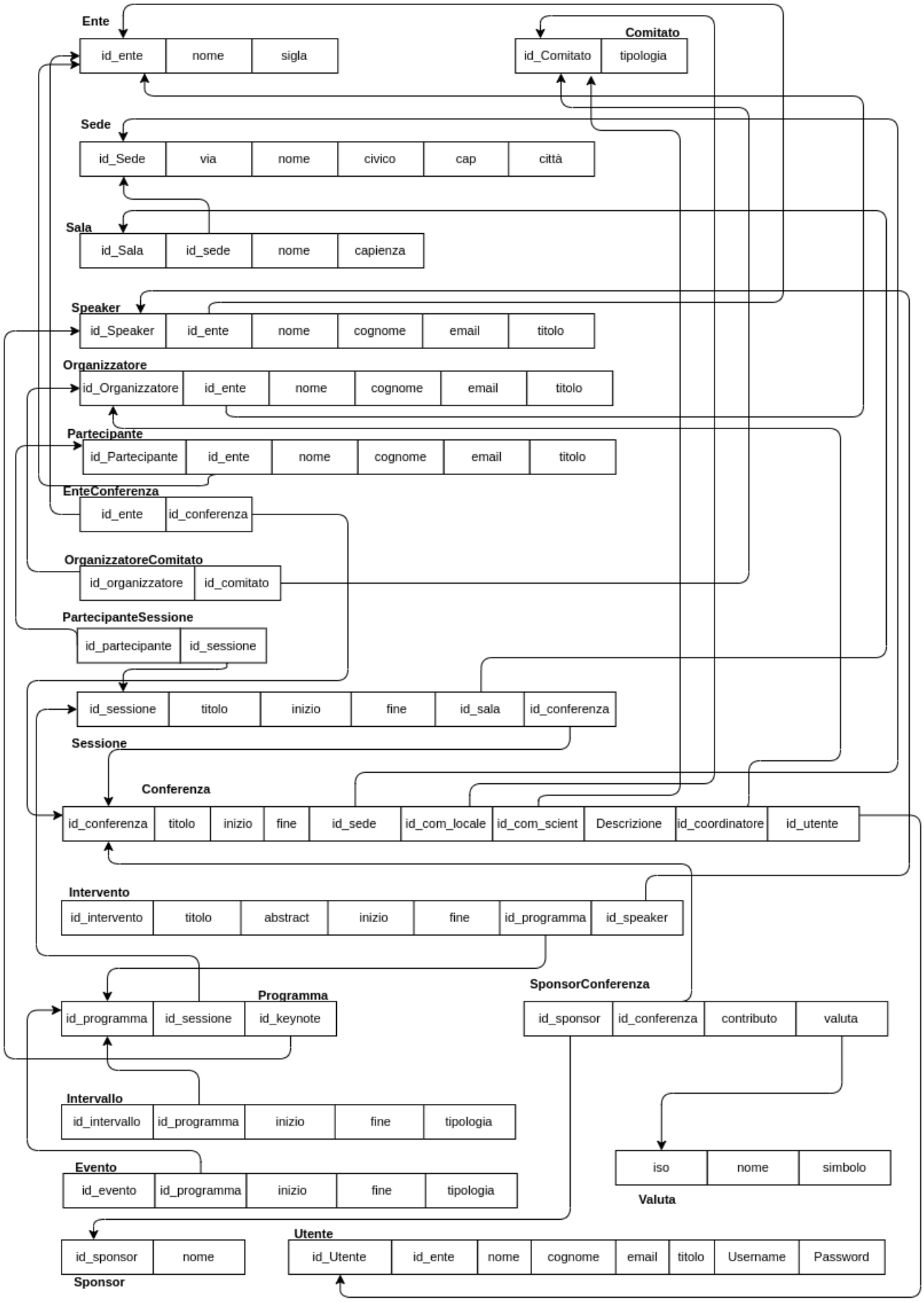


Figura 2.2: *Ristrutturazione dello schema concettuale*

Figura 2.3: *Schema logico*



Capitolo 3

Implementazione fisica

3.1 Definizione delle tabelle

3.1.1 Utente

```
1  CREATE TABLE utente(
2  id_utente SERIAL PRIMARY KEY,
3  username TEXT NOT NULL UNIQUE,
4  nome TEXT NOT NULL,
5  cognome TEXT NOT NULL,
6  titolo titolo_st,
7  email TEXT NOT NULL UNIQUE,
8  password TEXT NOT NULL,
9  id_ente INTEGER REFERENCES ente(id_ente) ON DELETE CASCADE
10 );
```

Listato 3.1: *Tabella: Utente*

3.1.2 Ente

```
1  CREATE TABLE ente(
2  id_ente SERIAL PRIMARY KEY,
3  nome TEXT NOT NULL UNIQUE,
4  sigla varchar(7) NOT NULL,
5  UNIQUE (nome,sigla)
6  );
```

Listato 3.2: *Tabella: Ente*

3.1.3 Indirizzo

```
1 CREATE TABLE indirizzo(
2 id_indirizzo SERIAL PRIMARY KEY,
3 via TEXT NOT NULL,
4 civico varchar(5) NOT NULL,
5 cap varchar(5) ,
6 city TEXT NOT NULL,
7 provincia varchar(2) NOT NULL,
8 nazione TEXT
9 );
```

Listato 3.3: *Tabella: Indirizzo*

3.1.4 Sede

```
CREATE TABLE sede(
id_sede SERIAL PRIMARY KEY,
nome TEXT ,
id_indirizzo INTEGER REFERENCES indirizzo(id_indirizzo) ON DELETE SET NULL
);
```

Listato 3.4: *Tabella: Sede*

3.1.5 Sponsor

```
CREATE TABLE sponsor(
id_sponsor SERIAL PRIMARY KEY,
nome TEXT NOT NULL
);
```

Listato 3.5: *Tabella: Sponsor*

3.1.6 Comitato

Ogni comitato ha una tipologia che varia tra i valori *scientifico* e *locale*. Definiamo quindi il tipo `comitato_st` che useremo per specificare la tipologia del comitato:

```
CREATE TYPE comitato_st AS enum ('locale','scientifico');
CREATE TABLE comitato(
id_comitato SERIAL PRIMARY KEY,
tipologia comitato_st NOT NULL
);
```

Listato 3.6: *Tabella: Comitato*

3.1.7 Organizzatore, Speaker, Partecipante

```
CREATE TYPE titolo_st AS enum ('Dottore','Dottoressa','Professore','
    Professoressa','Assistente','Ricercatore','Ricercatrice','Ingegnere');

CREATE TABLE organizzatore(
id_organizzatore SERIAL PRIMARY KEY,
nome TEXT NOT NULL,
cognome TEXT NOT NULL,
titolo titolo_st,
email TEXT NOT NULL UNIQUE,
id_ente INTEGER REFERENCES ente(id_ente) ON DELETE CASCADE
);
```

Listato 3.7: *Tabella: Organizzatore*

```
CREATE TABLE partecipante(
id_partecipante SERIAL PRIMARY KEY,
nome TEXT NOT NULL,
cognome TEXT NOT NULL,
titolo titolo_st,
email TEXT NOT NULL UNIQUE,
id_ente INTEGER REFERENCES ente(id_ente) ON DELETE SET NULL
);
```

Listato 3.8: *Tabella: Partecipante*

```
CREATE TABLE speaker(
id_speaker SERIAL PRIMARY KEY,
nome TEXT NOT NULL,
cognome TEXT NOT NULL,
titolo titolo_st,
email TEXT NOT NULL UNIQUE,
id_ente INTEGER REFERENCES ente(id_ente) ON DELETE CASCADE NOT NULL
);
```

Listato 3.9: *Tabella: Speaker*

3.1.8 Sala

```
  CREATE TABLE sala(
  id_sala SERIAL PRIMARY KEY,
  nome TEXT NOT NULL,
  capienza INTEGER NOT NULL,
  id_sede INTEGER REFERENCES sede(id_sede) ON DELETE CASCADE
  );
```

Listato 3.10: *Tabella: Sala*

3.1.9 Conferenza

```
CREATE TABLE conferenza(
id_conferenza SERIAL PRIMARY KEY,
titolo TEXT NOT NULL,
descrizione TEXT NOT NULL,
inizio TIMESTAMP NOT NULL,
fine TIMESTAMP NOT NULL,
id_sede INTEGER REFERENCES sede(id_sede) ON DELETE SET NULL,
comitato_s INTEGER REFERENCES comitato(id_comitato) ON DELETE SET NULL,
comitato_l INTEGER REFERENCES comitato(id_comitato) ON DELETE SET NULL,
id_utente INTEGER REFERENCES utente(id_utente) ON DELETE CASCADE,
CHECK (inizio <= fine),
CHECK (inizio >= now())
);
```

Listato 3.11: *Tabella: Conferenza*

3.1.10 Sessione

```
CREATE TABLE sessione(
id_sessione SERIAL PRIMARY KEY,
titolo TEXT NOT NULL,
inizio TIMESTAMP NOT NULL,
fine TIMESTAMP NOT NULL,
id_coordinatore INTEGER REFERENCES organizzatore(id_organizzatore) ON DELETE
    SET NULL,
id_conferenza INTEGER REFERENCES conferenza(id_conferenza) ON DELETE CASCADE,
id_sala INTEGER REFERENCES sala(id_sala) ON DELETE SET NULL,
CHECK (inizio <= fine)
);
```

Listato 3.12: *Tabella: Sessione*

3.1.11 Partecipazione

```
CREATE TABLE partecipazione(
id_partecipante INTEGER REFERENCES partecipante(id_partecipante) ON DELETE
    CASCADE,
id_sessione INTEGER REFERENCES sessione(id_sessione) ON DELETE CASCADE,
UNIQUE (id_partecipante,id_sessione)
);
```

Listato 3.13: *Tabella: Partecipazione*

3.1.12 Ente_Conferenza

```
CREATE TABLE ente_conferenza(
id_ente INTEGER REFERENCES ente(id_ente) ON DELETE CASCADE,
id_conferenza INTEGER REFERENCES conferenza(id_conferenza) ON DELETE CASCADE,
UNIQUE (id_ente,id_conferenza)
);
```

Listato 3.14: *Tabella: Ente_Conferenza*

3.1.13 Valuta

```
CREATE TABLE valuta(
iso CHAR(3) PRIMARY KEY,
nome TEXT NOT NULL,
simbolo TEXT NOT NULL
);
```

Listato 3.15: *Tabella: Valuta*

3.1.14 Sponsor_Conferenza

```
CREATE TABLE sponsor_conferenza(
id_sponsor INTEGER REFERENCES sponsor(id_sponsor) ON DELETE CASCADE NOT NULL,
contributo NUMERIC(1000,2) NOT NULL,
valuta CHAR(3) REFERENCES valuta(iso) NOT NULL,
id_conferenza INTEGER REFERENCES conferenza(id_conferenza) ON DELETE CASCADE
    NOT NULL,
UNIQUE (id_sponsor,id_conferenza)
);
```

Listato 3.16: *Tabella: Sponsor_Conferenza*

3.1.15 Programma

```
CREATE TABLE programma(
id_programma SERIAL PRIMARY KEY,
id_sessione INTEGER REFERENCES sessione(id_sessione) ON DELETE CASCADE NOT NULL
    ,
id_keynote INTEGER REFERENCES speaker(id_speaker) ON DELETE SET NULL,
UNIQUE (id_programma, id_sessione)
);
```

Listato 3.17: *Tabella: Programma*

3.1.16 Intervento

```
CREATE TABLE intervento(
id_intervento SERIAL PRIMARY KEY,
titolo TEXT NOT NULL,
abstract TEXT NOT NULL,
inizio TIMESTAMP NOT NULL,
fine TIMESTAMP NOT NULL,
id_speaker INTEGER REFERENCES speaker(id_speaker) ON DELETE CASCADE,
id_programma INTEGER REFERENCES programma(id_programma) ON DELETE CASCADE NOT
    NULL,
UNIQUE (id_speaker,id_programma),
CHECK (inizio <= fine)
);
```

Listato 3.18: *Tabella: Intervento*

3.1.17 Intervallo

```
CREATE TYPE intervallo_st AS enum ('pranzo','coffee break');
CREATE TABLE intervallo(
id_intervallo SERIAL PRIMARY KEY,
tipologia intervallo_st NOT NULL,
inizio TIMESTAMP NOT NULL,
fine TIMESTAMP NOT NULL,
CHECK (inizio <= fine),
id_programma INTEGER REFERENCES programma(id_programma) ON DELETE CASCADE NOT
    NULL
);
```

Listato 3.19: *Tabella: Programma*

3.1.18 Evento

```
CREATE TABLE evento(
id_evento SERIAL PRIMARY KEY,
tipologia TEXT NOT NULL,
inizio TIMESTAMP NOT NULL,
fine TIMESTAMP NOT NULL,
CHECK (inizio <= fine),
id_programma INTEGER REFERENCES programma(id_programma) ON DELETE CASCADE NOT
    NULL
);
```

Listato 3.20: *Tabella: Evento*

3.1.19 Organizzatore_Comitato

```
CREATE TABLE organizzatore_comitato(
id_organizzatore INTEGER REFERENCES organizzatore(id_organizzatore) ON DELETE
    CASCADE,
id_comitato INTEGER REFERENCES comitato(id_comitato) ON DELETE CASCADE,
UNIQUE (id_organizzatore,id_comitato)
);
```

Listato 3.21: *Tabella: Organizzatore_Comitato*

3.2 Definizione dei trigger

3.2.1 Check_Programma

In un programma non devono esserci eventi, intervalli o interventi che si sovrappongono. Per questo motivo definiamo il trigger `check_programma_entry` che viene eseguito per ogni inserimento o aggiornamento nelle tabelle Intervento, Intervallo ed Evento.

```
CREATE OR REPLACE FUNCTION check_programma()
RETURNS TRIGGER AS $$
DECLARE
  inizio_evento TIMESTAMP;
  fine_evento TIMESTAMP;
  inizio_intervallo TIMESTAMP;
  fine_intervallo TIMESTAMP;
  inizio_intervento TIMESTAMP;
  fine_intervento TIMESTAMP;
  intervento_id INTEGER;
  intervallo_id INTEGER;
  evento_id INTEGER;
  interventi_cur cursor FOR
    SELECT id_intervento
    FROM intervento
    WHERE id_programma = new.id_programma;
  intervalli_cur cursor FOR
    SELECT id_intervallo
    FROM intervallo
    WHERE id_programma = new.id_programma;
  eventi_cur cursor FOR
    SELECT id_evento
    FROM evento
    WHERE id_programma = new.id_programma;
BEGIN
  OPEN interventi_cur;
  LOOP
    FETCH interventi_cur INTO intervento_id;
    EXIT WHEN NOT FOUND;
    SELECT inizio,fine INTO inizio_intervento,fine_intervento
    FROM intervento
    WHERE id_intervento = intervento_id;
    IF (new.inizio>=inizio_intervento AND new.fine<=fine_intervento) THEN
      RAISE EXCEPTION 'Impossibile inserire intervento';
    END IF;
  END LOOP;
  CLOSE interventi_cur;

  OPEN intervalli_cur;
  LOOP
    FETCH intervalli_cur INTO intervallo_id;
    EXIT WHEN NOT FOUND;
    SELECT inizio,fine INTO inizio_intervallo,fine_intervallo
    FROM intervallo
    WHERE id_intervallo = intervallo_id;
    IF (new.inizio>=inizio_intervallo AND new.fine<=fine_intervallo) THEN
      RAISE EXCEPTION 'Impossibile inserire intervallo';
    END IF;
    END LOOP;
  CLOSE intervalli_cur;

  OPEN eventi_cur;
  LOOP
```

```
54      FETCH eventi_cur INTO evento_id;
55      EXIT WHEN NOT FOUND;
56      SELECT inizio,fine INTO inizio_evento,fine_evento
57      FROM evento
58      WHERE id_evento = evento_id;
59      IF (new.inizio>=inizio_evento AND new.fine<=fine_evento) THEN
60        RAISE EXCEPTION 'Impossibile inserire evento';
61      END IF;
62    END LOOP;
63    CLOSE eventi_cur;
64    RETURN NEW;
65  END;
66  $$
67  LANGUAGE PLPGSQL;
68
69  CREATE TRIGGER check_programma
70  BEFORE INSERT OR UPDATE ON intervento
71  FOR EACH ROW
72  EXECUTE FUNCTION check_programma();
73
74  CREATE TRIGGER check_programma
75  BEFORE INSERT OR UPDATE ON intervallo
76  FOR EACH ROW
77  EXECUTE FUNCTION check_programma();
78
79  CREATE TRIGGER check_programma
80  BEFORE INSERT OR UPDATE ON evento
81  FOR EACH ROW
82  EXECUTE FUNCTION check_programma();
```

Listato 3.22: *check_programma_entry*

3.2.2 Check_Data_Intervento, Check_Data_Intervallo, Check_Data_Evento

Ogni volta che viene inserito o aggiornato un intervento, un intervallo o un evento bisogna controllare sempre che la data di inizio e di fine sia coerente con quella della sessione cui appartengono:

```
1  CREATE OR REPLACE FUNCTION check_data() RETURNS TRIGGER AS $$
2  DECLARE
3  inizio_sessione TIMESTAMP;
4  fine_sessione TIMESTAMP;
5  BEGIN
6  SELECT inizio,fine INTO inizio_sessione,fine_sessione
7  FROM sessione
8  WHERE id_sessione = (SELECT id_sessione FROM programma WHERE id_programma = new
       .id_programma);
9
10 IF (new.inizio < inizio_sessione OR new.fine > fine_sessione) THEN
11 RAISE EXCEPTION 'L''intervento non e'' compreso nella sessione';
12 END IF;
13 RETURN NEW;
14 END;
15 $$ LANGUAGE PLPGSQL;
16
17 CREATE TRIGGER check_data_evento
18 BEFORE INSERT OR UPDATE ON evento
19 FOR EACH ROW
20 EXECUTE FUNCTION check_data();
21
```

```
CREATE TRIGGER check_data_intervento
BEFORE INSERT OR UPDATE ON intervento
FOR EACH ROW
EXECUTE FUNCTION check_data();

CREATE TRIGGER check_data_intervallo
BEFORE INSERT OR UPDATE ON intervallo
FOR EACH ROW
EXECUTE FUNCTION check_data();
```

Listato 3.23: *check_data_intervento*

3.2.3 Create_Programma_Sessione

Il trigger `Create_Programma_Sessione` viene attivato subito dopo aver inserito una nuova sessione ed effettua l'inserimento di un programma vuoto associato alla sessione.

```
CREATE OR REPLACE FUNCTION create_programma_sessione() RETURNS TRIGGER AS $$
BEGIN
INSERT INTO programma(id_sessione) VALUES (new.id_sessione);
RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER create_programma_sessione
AFTER INSERT ON sessione
FOR EACH ROW
EXECUTE FUNCTION create_programma_sessione();
```

Listato 3.24: *create_programma_sessione*

3.2.4 Check_Sala_Sessione

Quando inseriamo una sessione bisogna stare attenti che la chiave esterna della sala sia effettivamente una sala appartenente alla sede che ospita la conferenza della sessione IN questione. Il trigger `check_sala_sessione` effettua quindi questo controllo prima di ciascun inserimento nella tabella SESSIONE:

```
CREATE OR REPLACE FUNCTION check_sala_sessione() RETURNS TRIGGER AS $$
DECLARE
sede INTEGER;
sala INTEGER;
BEGIN
SELECT id_sede INTO sede
FROM conferenza
WHERE id_conferenza = new.id_conferenza;

SELECT id_sala INTO sala
FROM sala
WHERE id_sala = new.id_sala;

IF sala IS NULL THEN
Return new;
END IF;

IF sala NOT IN (
SELECT id_sala
FROM sala
WHERE id_sede = sede
```

```
) THEN
RAISE EXCEPTION 'La sala selezionata non appartiene alla sede della conferenza'
    ;
END IF;

RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;


CREATE TRIGGER check_sala_sessione
BEFORE INSERT OR UPDATE ON sessione
FOR EACH ROW
EXECUTE FUNCTION check_sala_sessione();
```

3.2.5 Check_Data_Sessione

Analogamente ai trigger 3.2.2 si definisce il trigger `check_data_sessione` che controlla che le DATE di inizio e di fine di ciascuna sessione siano coerenti con quelle della relativa conferenza:

```
CREATE OR REPLACE FUNCTION check_data_sessione() RETURNS TRIGGER AS $$
DECLARE
inizio_conferenza TIMESTAMP;
fine_conferenza TIMESTAMP;
BEGIN
SELECT inizio, fine INTO inizio_conferenza,fine_conferenza
FROM conferenza
WHERE id_conferenza = new.id_conferenza;

IF (new.inizio < inizio_conferenza OR new.fine > fine_conferenza) THEN
RAISE EXCEPTION 'La sessione non e'' compresa nella conferenza';
END IF;
RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER check_data_sessione
BEFORE INSERT OR UPDATE ON sessione
FOR EACH ROW
EXECUTE FUNCTION check_data_sessione();
```

Listato 3.25: *check_data_sessione*

3.2.6 Check_Coordinatore_Sessione

Quando si specifica il coordinatore della sessione bisogna controllare che questi appartenga al comitato scientifico che è il gruppo di organizzatori che si occupano della gestione delle conferenze e delle sessioni:

```
CREATE OR REPLACE FUNCTION check_coordinatore_sessione() RETURNS TRIGGER AS $$
DECLARE
id_comitato_scientifico_conferenza INTEGER;
BEGIN

SELECT comitato_s INTO id_comitato_scientifico_conferenza
FROM conferenza c
WHERE c.id_conferenza = new.id_conferenza;

IF (new.id_coordinatore is NOT NULL) THEN
```

```
IF (id_comitato_scientifico_conferenza NOT IN (SELECT id_comitato FROM
    organizzatore_comitato WHERE id_organizzatore = new.id_coordinatore))  THEN
RAISE EXCEPTION 'Il coordinatore della sessione deve appartenere al comitato
    scientifico della conferenza';
END IF;
END IF;
RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER check_coordinatore_sessione
BEFORE INSERT OR UPDATE ON sessione
FOR EACH ROW
EXECUTE FUNCTION check_coordinatore_sessione();
```

Listato 3.26: *check_ coordinatore_ sessione*

3.2.7 Create_Comitati_Conferenza

Gli enti che organizzano le conferenze nominano due comitati per ogni conferenza che organizzano. Per questo motivo, ogni volta che viene inserita una nuova conferenza viene attivato il trigger `create_comitati_conferenza` che si occupa di creare due nuovi comitati di tipologica *scientifica* e *locale* e associarli alla nuova conferenza appena create:

```
CREATE OR REPLACE FUNCTION create_comitati_conferenza() RETURNS TRIGGER AS $$
DECLARE
id_comitatoscientifico INTEGER;
id_comitatolocale INTEGER;
BEGIN
INSERT INTO comitato(tipologia) VALUES ('scientifico') returning id_comitato
    INTO id_comitatoscientifico;
INSERT INTO comitato(tipologia) VALUES ('locale') returning id_comitato INTO
    id_comitatolocale;
UPDATE conferenza SET comitato_s = id_comitatoscientifico, comitato_l =
    id_comitatolocale WHERE id_conferenza = new.id_conferenza;
RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER create_comitati_conferenza
AFTER INSERT ON conferenza
FOR EACH ROW
EXECUTE FUNCTION create_comitati_conferenza();
```

Listato 3.27: *create_comitati_conferenza*

3.2.8 Check_Comitati_Conferenza

Ogni volta che si aggiorna una conferenza bisogna controllare che le chiavi esterne dei due comitati si riferiscano sempre a comitati della tipologia richiesta:

```
CREATE OR REPLACE FUNCTION check_comitati_conferenza() RETURNS TRIGGER AS $$
DECLARE
id_comitato_scientifico INTEGER;
id_comitato_locale INTEGER;
BEGIN
SELECT id_comitato INTO id_comitato_scientifico
FROM comitato
WHERE id_comitato = new.comitato_s;
```

```
9
10 SELECT id_comitato INTO id_comitato_locale
11 FROM comitato
12 WHERE id_comitato = new.comitato_l;
13
14 IF id_comitato_scientifico IS NULL THEN
15 RETURN NEW;
16 END IF;
17
18 IF id_comitato_locale IS NULL THEN
19 RETURN NEW;
20 END IF;
21
22 IF (SELECT tipologia FROM comitato WHERE id_comitato = id_comitato_scientifico)
       <> 'scientifico' THEN
23 RAISE EXCEPTION 'Il comitato scientifico deve essere scientifico';
24 END IF;
25
26 IF (SELECT tipologia FROM comitato WHERE id_comitato = id_comitato_locale) <> '
     locale' THEN
27 RAISE EXCEPTION 'Il comitato locale deve essere locale';
28 END IF;
29
30 RETURN NEW;
31 END;
32 $$ LANGUAGE PLPGSQL;
33
34 CREATE TRIGGER check_comitati_conferenza
35 BEFORE UPDATE ON conferenza
36 FOR EACH ROW
37 EXECUTE FUNCTION check_comitati_conferenza();
```

Listato 3.28: *check_comitati_conferenza*

3.2.9 Check_Sala_Sessione_Unica

Una sala non può ospitare più di una sessione alla volta.

```
1 CREATE OR REPLACE FUNCTION check_sala_sessione_unica() RETURNS TRIGGER AS $$
2 DECLARE
3 inizio_sessione TIMESTAMP;
4 fine_sessione TIMESTAMP;
5 sessioni cursor FOR SELECT id_sessione FROM sessione WHERE id_sala = new.
     id_sala;
6 sessione_id INTEGER;
7 BEGIN
8 OPEN sessioni;
9 LOOP
10 FETCH sessioni INTO sessione_id;
11 EXIT WHEN NOT FOUND;
12 SELECT inizio,fine INTO inizio_sessione,fine_sessione
13 FROM sessione
14 WHERE id_sessione = sessione_id;
15 IF (new.inizio >= inizio_sessione AND new.inizio <= fine_sessione) OR (new.fine
       >= inizio_sessione AND new.fine <= fine_sessione) THEN
16 RAISE EXCEPTION 'La sala non puo'' ospitare piu'' di una sessione alla volta';
17 END IF;
18 END LOOP;
19 CLOSE sessioni;
20 RETURN NEW;
21 END;
```

```
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER check_sala_sessione_unica
BEFORE INSERT OR UPDATE ON sessione
FOR EACH ROW
EXECUTE FUNCTION check_sala_sessione_unica();
```

Listato 3.29: *Check_sala_sessione_unica*

3.2.10 Check_Organizzatore_Comitato

Ogni volta che si inserisce un nuovo organizzatore all'interno di un comitato bisogna controllare che questo appartenga ad uno degli enti che organizzano la conferenza.

```
CREATE OR REPLACE FUNCTION check_organizzatore_comitato() RETURNS TRIGGER AS $$
DECLARE
ente_id INTEGER;
BEGIN

SELECT id_ente INTO ente_id
FROM organizzatore o
WHERE o.id_organizzatore = new.id_organizzatore;

IF ente_id IS NULL THEN
RAISE EXCEPTION 'L''organizzatore non esiste';
END IF;

IF ente_id NOT IN (
SELECT id_ente
FROM ente_conferenza
WHERE id_conferenza IN (
SELECT id_conferenza
FROM conferenza
WHERE NEW.id_comitato IN (id_comitato_scientifico, id_comitato_locale)
)
) THEN
RAISE EXCEPTION 'L''organizzatore deve appartenere ad un ente che ha
    organizzato la conferenza';
END IF;
RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER check_organizzatore_comitato
BEFORE INSERT OR UPDATE ON organizzatore_comitato
FOR EACH ROW
EXECUTE FUNCTION check_organizzatore_comitato();
```

Listato 3.30: *Check_organizzatori_comitato*

3.2.11 Delete_Sessioni_Conferenza

Nel caso IN cui si volesse modificare la data di inizio o di fine di una conferenza vengono automaticamente cancellate le sessioni che si trovano escluse dal nuovo intervallo di DATE.

```
CREATE OR REPLACE FUNCTION delete_sessioni_conferenza() RETURNS TRIGGER AS $$
DECLARE
sessioni_cur cursor FOR
SELECT id_sessione
FROM sessione
```

```
WHERE id_conferenza = old.id_conferenza;
sessione_id INTEGER;
BEGIN
OPEN sessioni_cur;
LOOP
FETCH sessioni_cur INTO sessione_id;
EXIT WHEN NOT FOUND;
IF (SELECT inizio FROM sessione WHERE id_sessione = sessione_id) < new.inizio
OR (SELECT fine FROM sessione WHERE id_sessione = sessione_id) > new.fine THEN
DELETE FROM sessione WHERE id_sessione = sessione_id;
END IF;
END LOOP;
CLOSE sessioni_cur;
RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER delete_sessioni_conferenza
BEFORE UPDATE ON conferenza
FOR EACH ROW
EXECUTE FUNCTION delete_sessioni_conferenza();
```

Listato 3.31: *DELETE_sessioni_conferenza*

3.2.12 Check_Capienza

Ogni volta che si aggiunge un nuovo partecipante della sessione bisogna controllare prima che la capienza della sala dove si svolge la sessione non sia stata raggiunta:

```
CREATE OR REPLACE FUNCTION check_capienza_sala() RETURNS TRIGGER AS $$
DECLARE
capienza_s INTEGER;
partecipanti INTEGER;
BEGIN
SELECT capienza INTO capienza_s
FROM sala
WHERE id_sala = new.id_sala;

SELECT count(*) INTO partecipanti
FROM partecipazione
WHERE id_sessione = new.id_sessione;

IF (partecipanti >= capienza_s) THEN
RAISE EXCEPTION 'La capienza della sala e'' stata raggiunta';
END IF;
RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER check_capienza_sala
before insert ON partecipazione
FOR EACH ROW
EXECUTE FUNCTION check_capienza_sala();
```

3.3 Funzioni e procedure

3.3.1 Show_Conferenze_By_Date(DATE,DATE)

La funzione `Show_Conferenze_By_Date` prende IN ingresso due DATE e restituisce l'insieme di tutte le conferenze comprese tra queste:

```
CREATE OR REPLACE FUNCTION show_conference_by_date(dataI DATE, dataF DATE)
RETURNS SETOF conferenza AS $$
BEGIN
RETURN QUERY
SELECT * FROM conferenza
WHERE inizio >= dataI AND fine <= dataF;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.2 Show_Conferenze_By_Sede(integer)

La funzione `Show_Conferenze_By_Sede` prende IN ingresso la chiave primaria di una sede e restituisce l'insieme di tutte le conferenze ospitate IN quella determinata sede:

```
CREATE OR REPLACE FUNCTION show_conferences_by_sede(sede INTEGER)
RETURNS SETOF conferenza AS $$
BEGIN
RETURN QUERY
SELECT * FROM conferenza
WHERE id_sede = sede;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.3 Show_comitato_scientifico(integer)

La funzione `Show_comitato_scientifico` prende IN ingresso la chiave primaria di una conferenza e restituisce la lista di tutti i membri organizzatori appartenenti al comitato scientifico della conferenza:

```
CREATE OR REPLACE FUNCTION show_comitato_scientifico(conferenza INTEGER)
RETURNS SETOF organizzatore AS $$
BEGIN
RETURN QUERY
-- Select dei dettagli dell'organizzatore
SELECT * FROM organizzatore
WHERE id_organizzatore IN (
-- Select degli id degli organizzatori appartenenti al comitato scientifico
SELECT id_organizzatore FROM organizzatore_comitato
WHERE id_comitato = (
-- Select dell'id del comitato scientifico della conferenza
SELECT id_comitato_scientifico FROM conferenza
WHERE id_conferenza = conferenza
)
);
END;
$$ LANGUAGE PLPGSQL;
```

3.3.4 Show_comitato_locale(integer)

La funzione `Show_comitato_locale` prende IN ingresso la chiave primaria di una conferenza e restituisce la lista di tutti i membri organizzatori appartenenti al comitato locale della conferenza:

```sql
CREATE OR REPLACE FUNCTION show_comitato_locale(conferenza INTEGER)
RETURNS SETOF organizzatore AS $$
BEGIN
RETURN QUERY
-- Select dei dettagli dell'organizzatore
SELECT * FROM organizzatore
WHERE id_organizzatore IN (
-- Select degli id degli organizzatori appartenenti al comitato locale
SELECT id_organizzatore FROM organizzatore_comitato
WHERE id_comitato = (
-- Select dell'id del comitato locale della conferenza
SELECT id_comitato_locale FROM conferenza
WHERE id_conferenza = conferenza
)
);
END;
$$ LANGUAGE PLPGSQL;
```

3.3.5 Show_Partecipanti(INTEGER)

La funzione `Show_Partecipanti` prende IN ingresso la chiave primaria di una conferenza e restituisce tutti i dettagli dei partecipanti di *tutte le sessioni* della conferenza.

```sql
CREATE OR REPLACE FUNCTION show_partecipanti(conferenza INTEGER)
RETURNS SETOF partecipante AS $$
BEGIN
RETURN QUERY
-- Select dei dettagli del partecipante
SELECT * FROM partecipante
WHERE id_partecipante IN (
-- Select degli id dei partecipanti
SELECT id_partecipante FROM partecipazione
WHERE id_sessione IN (
-- Select degli id delle sessioni della conferenza
SELECT id_sessione FROM sessione
WHERE id_conferenza = conferenza
)
);
END;
$$ LANGUAGE PLPGSQL;
```

3.3.6 Show_Sessioni(integer)

La funzione `show_sessioni` prende IN ingresso la chiave primaria di una conferenza e restituisce tutti i dettagli delle sessioni.

```sql
CREATE OR REPLACE FUNCTION show_sessioni(conferenza INTEGER)
RETURNS SETOF sessione AS $$
BEGIN
RETURN QUERY
SELECT * FROM sessione
WHERE id_conferenza = conferenza
order by inizio;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.7 Show_interventi_sessione(integer)

La funzione **show_interventi_sessione** prende IN ingresso la chiave primaria di una sessione e mostra tutti gli interventi presenti nel programma di tale sessione:

```
CREATE OR REPLACE FUNCTION show_interventi_sessione(sessione INTEGER)
RETURNS TABLE
(
titolo TEXT,
inizio TIMESTAMP,
fine TIMESTAMP,
abstract TEXT,
speaker TEXT
)  AS $$
DECLARE
programma INTEGER;
BEGIN
SELECT id_programma INTO programma
FROM programma
WHERE id_sessione = sessione;

SELECT titolo,inizio,fine,abstract, s.nome || ' ' || s.cognome AS speaker
FROM intervento i JOIN speaker s ON i.id_speaker = s.id_speaker
WHERE i.id_programma = programma
order by inizio;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.8 Show_intervalli_sessione(integer)

La funzione **show_intervalli_sessione** prende IN ingresso la chiave primaria di una sessione e mostra tutti gli intervalli presenti nel programma di tale sessione:

```
CREATE OR REPLACE FUNCTION show_intervalli_sessione(sessione INTEGER)
RETURNS TABLE
(
id_intervallo INTEGER,
tipologia intervallo_st,
inizio TIMESTAMP,
fine TIMESTAMP
)
AS $$
DECLARE
programma INTEGER;
BEGIN
SELECT id_programma INTO programma
FROM programma
WHERE id_sessione = sessione;

SELECT id_intervallo,tipologia,inizio,fine
FROM intervallo i
WHERE id_programma = programma
order by inizio;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.9 Show_eventi_sociali_sessione(integer)

La funzione **show_eventi_sociali_sessione** prende IN ingresso la chiave primaria di una sessione e mostra tutti gli intervalli presenti nel programma di tale sessione:

```sql
CREATE OR REPLACE FUNCTION show_eventi_sociali_sessione(sessione INTEGER)
RETURNS TABLE
(
id_evento INTEGER,
tipologia TEXT,
inizio TIMESTAMP,
fine TIMESTAMP)
AS $$
DECLARE
programma INTEGER;
BEGIN
SELECT id_programma INTO programma
FROM programma
WHERE id_sessione = sessione;

SELECT id_evento,tipologia,inizio,fine
FROM evento
WHERE id_programma = programma
order by inizio;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.10 Show_keynote_sessione(integer)

La funzione **show_keynote_sessione** prende IN ingresso la chiave primaria di una sessione e mostra i dettagli del keynote speaker, se presente:

```sql
CREATE OR REPLACE FUNCTION show_keynote_sessione(sessione INTEGER)
RETURNS TABLE(
id_speaker INTEGER,
nome TEXT,
cognome TEXT,
titolo TEXT,
email TEXT,
ente TEXT)
AS $$
DECLARE
programma INTEGER;
BEGIN
SELECT id_programma INTO programma
FROM programma
WHERE id_sessione = sessione;

SELECT s.id_speaker,s.nome,s.cognome,s.titolo,s.email,e.nome
FROM speaker s JOIN ente e ON s.id_ente = e.id_ente
WHERE s.id_speaker = (
SELECT id_keynote
FROM programma
WHERE id_programma = programma
);
END;
$$ LANGUAGE PLPGSQL;
```

3.3.11 Show_Programma(integer)

La funzione **Show_Programma** prende IN ingresso la chiave primaria di una sessione e restituisce una tabella che mostra tutti gli appuntamenti IN progamma IN ordine cronologico:

```sql
CREATE OR REPLACE FUNCTION show_programma(sessione INTEGER)
```

```
RETURNS TABLE (
id_entry INTEGER,
appuntamento TEXT,
inizio TIMESTAMP,
fine TIMESTAMP,
descrizione TEXT,
speaker TEXT
)
AS $$
DECLARE
programma INTEGER;
BEGIN
SELECT id_programma INTO programma
FROM programma
WHERE id_sessione = sessione;

RETURN QUERY
SELECT *
FROM (
SELECT distinct i.id_intervento AS id_entry,
'intervento' AS appuntamento,
i.inizio,
i.fine,
i.abstract,
s.nome || ' ' || s.cognome AS speaker
FROM intervento i
JOIN speaker s ON i.id_speaker = s.id_speaker
WHERE i.id_programma = programma

UNION ALL

SELECT i2.id_intervallo AS id_entry,
'intervallo' AS appuntamento,
i2.inizio,
i2.fine,
tipologia::TEXT AS descrizione,
NULL
FROM intervallo i2
WHERE i2.id_programma = programma

UNION ALL

SELECT e.id_evento AS id_entry,
'evento' AS appuntamento,
e.inizio,
e.fine,
e.tipologia::TEXT AS descrizione,
NULL
FROM evento e
WHERE e.id_programma = programma
) AS subquery
ORDER BY inizio;
END;
$$ LANGUAGE plpgsql;
```

3.3.12 `Add_Intervento(TEXT,TEXT,TEXT,INTEGER,interval)`

La procedura `Add_intervento` provvede all'inserimento di un intervento all'interno del programma della sessione. Questa calcola l'orario esatto IN cui inserire il nuovo punto sulla base

dell'ultimo punto IN programma. Se non esistono punti IN programma allora l'ora di inizio è calcolato come l'inizio della sessione:

```
create or replace procedure add_intervento
(titolo TEXT,
abstract TEXT,
speaker TEXT,
sessione_id INTEGER,
durata interval)
AS $$
DECLARE
programma INTEGER;
fine_prev TIMESTAMP;
BEGIN
SELECT id_programma INTO programma
FROM programma
WHERE id_sessione = sessione_id;

SELECT max(fine) INTO fine_prev
FROM show_programma(sessione_id);

IF (fine_prev is NULL) THEN
SELECT inizio INTO fine_prev
FROM sessione
WHERE id_sessione = sessione_id;
END IF;

INSERT INTO intervento(titolo,abstract,id_speaker,id_programma,inizio,fine)
VALUES (titolo,abstract,speaker,programma,fine_prev,fine_prev+durata);
RAISE NOTICE 'Inserimento completato';
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE '%', SQLERRM;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.13 Add_Intervallo(TEXT,INTEGER,interval)

La procedura `Add_Intervallo` provvede all'inserimento di un intervallo all'interno del programma della sessione. Questa calcola l'orario esatto IN cui inserire il nuovo punto sulla base dell'ultimo punto IN programma. Se non esistono punti IN programma allora l'ora di inizio è calcolato come l'inizio della sessione:

```
create or replace procedure
add_intervallo(tipologia TEXT , sessione_id INTEGER, durata interval)
AS $$
DECLARE
programma INTEGER;
fine_prev TIMESTAMP;
BEGIN
SELECT id_programma INTO programma
FROM programma
WHERE id_sessione = sessione_id;

SELECT max(fine) INTO fine_prev
FROM show_programma(sessione_id);

IF (fine_prev is NULL) THEN
SELECT inizio INTO fine_prev
FROM sessione
```

```
WHERE id_sessione = sessione_id;
END IF;

INSERT INTO intervallo(tipologia,id_programma,inizio,fine)
VALUES (tipologia::intervallo_st, programma, fine_prev, fine_prev+durata);
RAISE NOTICE 'Inserimento completato';
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE '%', SQLERRM;
END;
$$
LANGUAGE PLPGSQL;
```

3.3.14 Add_Evento(TEXT,INTEGER,interval)

La procedura **Add_Evento** provvede all'inserimento di un evento all'interno del programma della sessione. Questa calcola l'orario esatto IN cui inserire il nuovo punto sulla base dell'ultimo punto IN programma. Se non esistono punti IN programma allora l'ora di inizio è calcolato come l'inizio della sessione:

```
create or replace procedure
add_evento
(tipologia TEXT,
sessione_id INTEGER,
durata interval)
AS $$
DECLARE
programma_id INTEGER;
fine_prev TIMESTAMP;
BEGIN
-- Recupera l'id del programma della sessione
SELECT id_programma INTO programma_id
FROM programma
WHERE id_sessione = sessione_id;

-- Recupera l'id dell'ultimo punto IN programma, la tipologia e la fine
SELECT max(fine) INTO fine_prev
FROM show_programma(sessione_id);

IF (fine_prev is NULL) THEN
SELECT inizio INTO fine_prev
FROM sessione
WHERE id_sessione = sessione_id;
END IF;

INSERT INTO evento(tipologia, id_programma, inizio, fine)
VALUES (tipologia, programma_id, fine_prev, fine_prev+durata);
RAISE NOTICE 'Inserimento completato';
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE '%', SQLERRM;
END;
$$
LANGUAGE PLPGSQL;
```

3.3.15 Add_Conferenza_Details(TEXT,TIMESTAMP,TIMESTAMP,integer,TEXT)

La funzione **Add_Conferenza_Details** aggiunge una conferenza e restituisce la chiave primaria della nuova conferenza.

```
CREATE OR REPLACE FUNCTION add_conferenza_details
(nome TEXT, inizio TIMESTAMP, fine TIMESTAMP, sede INTEGER, abstract TEXT,
    utente INTEGER)
RETURNS INTEGER AS $$
DECLARE
id INTEGER;
BEGIN
INSERT INTO conferenza(titolo, inizio, fine, id_sede, descrizione, id_utente)
VALUES (nome, inizio, fine, sede, abstract,utente)
RETURNING id_conferenza INTO id;
RAISE NOTICE 'Inserimento completato';
RETURN id;
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE 'Errore nell''inserimento di una conferenza: %', SQLERRM;
RETURN 0;
END;
$$ LANGUAGE plpgsql;
```

3.3.16 Add_ente(integer, integer)

La procedura `Add_ente` provvede all'inserimento di una nuova istituzione tra gli organizzatori di una conferenza.

```
create or replace procedure add_ente(ente INTEGER, conferenza INTEGER)
AS $$
BEGIN
INSERT INTO ente_conferenza(id_ente,id_conferenza)
VALUES (ente,conferenza);
RAISE NOTICE 'Inserimento completato';
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE '%', SQLERRM;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.17 Add_Sponsorizzazione(integer,NUMERIC,CHAR(3),integer)

La procedura `Add_Sponsorizzazione` inserisce una nuova sponsorizzazione per la conferenza:

```
create or replace procedure add_sponsorizzazione(sponsor INTEGER, contributo
    NUMERIC(1000,2), valuta CHAR(3), conferenza INTEGER)
AS $$
BEGIN
INSERT INTO sponsorizzazione(id_sponsor,contributo,valuta,id_conferenza)
VALUES (sponsor,contributo,valuta,conferenza);
RAISE NOTICE 'Inserimento completato';
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE '%', SQLERRM;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.18 Add_Sessione(TEXT,TIMESTAMP,TIMESTAMP, integer,integer)

La procedura `Add_Sessione` aggiunge una nuova sessione per la conferenza:

```
create or replace procedure add_sessione(titolo TEXT, inizio TIMESTAMP, fine
    TIMESTAMP, sala INTEGER, conferenza INTEGER)
```

```
AS $$
BEGIN
INSERT INTO sessione(titolo,inizio,fine,id_sala,id_conferenza)
VALUES (titolo,inizio,fine,sala,conferenza);
RAISE NOTICE 'Inserimento completato';
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE '%', SQLERRM;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.19 Add_Partecipante(integer, integer)

La procedura `Add_Partecipante` inserisce un nuovo partecipante alla sessione:

```
create or replace procedure add_partecipante(partecipante INTEGER, sessione
    INTEGER)
AS $$
BEGIN
INSERT INTO partecipante_sessione(id_partecipante,id_sessione)
VALUES (partecipante,sessione);
RAISE NOTICE 'Inserimento completato';
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE '%', SQLERRM;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.20 Add_Enti(integer,TEXT)

```
CREATE OR REPLACE PROCEDURE add_enti(conferenza INTEGER, sigle TEXT)
AS $$
DECLARE
sigla_ente TEXT;
ente_id INTEGER;
BEGIN
FOR sigla_ente IN SELECT unnest(string_to_array(sigle, ',')) LOOP
-- Cerca l'id dell'ente corrispondente alla sigla
SELECT id_ente INTO ente_id FROM ente WHERE sigla = sigla_ente;

-- Inserisci la tupla (id_ente, conferenza) nella tabella ente_conferenza
INSERT INTO ente_conferenza(id_ente, id_conferenza) VALUES (ente_id, conferenza
    );
END LOOP;
RAISE NOTICE 'Inserimento completato';
EXCEPTION
WHEN OTHERS THEN
RAISE EXCEPTION 'Errore durante l''inserimento delle tuple nella tabella
    ente_conferenza: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

3.3.21 Add_Conferenza(TEXT,TIMESTAMP,TIMESTAMP,integer, TEXT, TEXT)

```
create or replace procedure add_conferenza(nome TEXT, inizio TIMESTAMP, fine
    TIMESTAMP, sede INTEGER, descrizione TEXT, sigle TEXT)
AS $$
DECLARE
```

```
id_conferenza INTEGER;
BEGIN
id_conferenza := add_conferenza_details(nome,inizio,fine,sede,descrizione);
call add_enti(id_conferenza,sigle);
EXCEPTION
WHEN OTHERS THEN
RAISE NOTICE '%', SQLERRM;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.22 Slitta_Conferenza(interval)

```
  create or replace procedure
slitta_conferenza(conferenza_id INTEGER, durata interval)
AS $$
DECLARE
sessione_id INTEGER;
intervento_id INTEGER;
evento_id INTEGER;
intervallo_id INTEGER;
sessioni cursor FOR
SELECT id_sessione
FROM sessione
WHERE id_conferenza = conferenza_id;

interventi cursor FOR
SELECT id_intervento
FROM intervento i JOIN programma p
ON i.id_programma = p.id_programma
WHERE p.id_sessione IN
(SELECT id_sessione
FROM sessione
WHERE id_conferenza = conferenza_id);

intervalli cursor FOR
SELECT id_intervallo
FROM intervallo i JOIN programma p
ON i.id_programma = p.id_programma
WHERE p.id_sessione IN
(SELECT id_sessione
FROM sessione
WHERE id_conferenza = conferenza_id);

eventi cursor FOR
SELECT id_evento
FROM evento e JOIN programma p
ON e.id_programma = p.id_programma
WHERE p.id_sessione IN
(SELECT id_sessione
FROM sessione
WHERE id_conferenza = conferenza_id);
BEGIN
ALTER TABLE conferenza DISABLE TRIGGER ALL;
ALTER TABLE sessione DISABLE TRIGGER ALL;
ALTER TABLE intervento DISABLE TRIGGER ALL;
ALTER TABLE intervallo DISABLE TRIGGER ALL;
ALTER TABLE evento DISABLE TRIGGER ALL;
ALTER TABLE programma DISABLE TRIGGER ALL;
UPDATE conferenza
SET inizio = inizio + durata, fine = fine + durata
```

```
49 WHERE id_conferenza = conferenza_id;
50
51 OPEN sessioni;
52 LOOP
53 FETCH sessioni INTO sessione_id;
54 EXIT WHEN NOT FOUND;
55
56 UPDATE sessione
57 SET inizio = inizio + durata, fine = fine + durata
58 WHERE id_sessione = sessione_id;
59
60 OPEN interventi;
61 LOOP
62 FETCH interventi INTO intervento_id;
63 EXIT WHEN NOT FOUND;
64
65 UPDATE intervento
66 SET inizio = inizio + durata, fine = fine + durata
67 WHERE id_intervento = intervento_id ;
68 END LOOP;
69 CLOSE interventi;
70
71 OPEN intervalli;
72 LOOP
73 FETCH intervalli INTO intervallo_id;
74 EXIT WHEN NOT FOUND;
75
76 UPDATE intervallo
77 SET inizio = inizio + durata, fine = fine + durata
78 WHERE id_intervallo = intervallo_id;
79 END LOOP;
80 CLOSE intervalli;
81
82 OPEN eventi;
83 LOOP
84 FETCH eventi INTO evento_id;
85 EXIT WHEN NOT FOUND;
86
87 UPDATE evento
88 SET inizio = inizio + durata, fine = fine + durata
89 WHERE id_evento = evento_id;
90 END LOOP;
91 CLOSE eventi;
92 END LOOP;
93 CLOSE sessioni;
94 ALTER TABLE conferenza ENABLE TRIGGER ALL;
95 ALTER TABLE sessione ENABLE TRIGGER ALL;
96 ALTER TABLE intervento ENABLE TRIGGER ALL;
97 ALTER TABLE intervallo ENABLE TRIGGER ALL;
98 ALTER TABLE evento ENABLE TRIGGER ALL;
99 ALTER TABLE programma ENABLE TRIGGER ALL;
100 RAISE NOTICE 'Slittamento completato';
101 EXCEPTION
102 WHEN OTHERS THEN
103 RAISE NOTICE '%', SQLERRM;
104 END;
105 $$ LANGUAGE PLPGSQL;
```

3.3.23 Show_members()

```
CREATE OR REPLACE FUNCTION show_members(conferenza INTEGER)
RETURNS TABLE
(
id INTEGER,
nome TEXT,
cognome TEXT,
email TEXT,
titolo titolo_st,
sigla varchar(7)
)
AS $$
BEGIN
RETURN QUERY
SELECT o.id_organizzatore, o.nome, o.cognome, o.email,o.titolo, e.sigla
FROM organizzatore o JOIN ente_conferenza ec natural JOIN ente e
ON o.id_ente = ec.id_ente
WHERE ec.id_conferenza = conferenza
GROUP by e.sigla;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.24 Show_percentage_interventi(INTEGER,INTEGER)

```
CREATE OR REPLACE FUNCTION show_percentage_interventi(mese INTEGER, anno
    INTEGER)
RETURNS TABLE
(
ente TEXT,
percentuale TEXT
) AS $$
DECLARE
totale INTEGER;
BEGIN
SELECT count(*) INTO totale
FROM intervento
WHERE date_part('month',inizio) = mese AND date_part('year',inizio) = anno;

RETURN QUERY
SELECT e.nome, (count(*)*100/totale)::TEXT || '%'
FROM intervento i JOIN speaker s
ON i.id_speaker = s.id_speaker JOIN ente e
ON s.id_ente = e.id_ente
WHERE date_part('month',inizio) = mese AND date_part('year',inizio) = anno
group by e.nome;
END;
$$ LANGUAGE PLPGSQL;
```

3.3.25 Show_percentage(INTEGER)

```
CREATE OR REPLACE FUNCTION show_percentage_interventi(anno INTEGER)
RETURNS TABLE
(
nome varchar(7),
percentuale TEXT
) AS $$
DECLARE
totale INTEGER;
BEGIN
```

```
SELECT count(*) INTO totale
FROM intervento
WHERE date_part('year',inizio) = anno;

RETURN QUERY
SELECT e.nome, (count(*)*100/totale)::TEXT || '%'
FROM intervento i JOIN speaker s
ON i.id_speaker = s.id_speaker JOIN ente e
ON s.id_ente = e.id_ente
WHERE date_part('year',inizio) = anno
group by e.nome;
END;
$$ LANGUAGE PLPGSQL;
```

3.4 Definizione delle viste

3.4.1 SediView

```
create view SediView AS
SELECT s.nome AS Sede,
i.via ||', '
|| i.civico
|| ', '
||  i.cap
||', '
|| i.city
|| ' ('
|| i.provincia
||'), '
|| i.nazione AS Indirizzo
FROM sede s natural JOIN indirizzo i;
```

3.4.2 Conferenze_Sede

```
create view conferenze_sede AS
SELECT s.nome AS Sede, count(id_conferenza) AS Numero_Conferenze
FROM sede s,conferenza c
WHERE s.id_sede = c.id_sede
group by s.nome;
```

3.4.3 Interventi_Speaker

```
create view interventi_speaker AS
SELECT s.nome || ' ' || s.cognome AS Speaker, count(i.id_intervento)
FROM speaker s, intervento i
WHERE s.id_speaker = i.id_speaker
group by s.nome,s.cognome;
```

3.4.4 Partecipanti_Sessione

```
create view partecipanti_sessioni AS
SELECT s.titolo AS Sessione,
  count(p.id_partecipante) AS Numero_partecipanti
FROM sessione s, partecipazione p
WHERE s.id_sessione = p.id_sessione
group by s.titolo;
```

3.4.5 Partecipanti_Conferenze

```
create view partecipanti_conferenze AS
SELECT c.titolo AS Conferenza,
  count(p.id_partecipante) AS Numero_partecipanti
FROM conferenza c, sessione s, partecipazione p
WHERE c.id_conferenza = s.id_conferenza
AND s.id_sessione = p.id_sessione
group by c.titolo;
```

3.4.6 Sessioni

```
create view sessioni AS
SELECT s.titolo AS Sessione,s.inizio,s.fine,c.titolo AS Conferenza,s1.nome
  FROM sessione s, conferenza c,sala s1
WHERE s.id_conferenza=c.id_conferenza AND s.id_sala=s1.id_sala
order by s.id_conferenza, s.inizio;
```

Appendice A

Dizionari

A.1 Dizionario dei dati

Classe	Descrizione	Attributi
Comitato	Tabella che descrive i comitati che si occupano della logistica e della pianificazione delle conferenze scientifiche.	`id_comitato` (*serial*) (*totale*): Identificatore univoco per un comitato. `tipologia`(*comitato_st*)(*totale*): Specifica il tipo di comitato (scientifico o locale).
Conferenza	Tabella che descrive le conferenze scientifiche.	`Id_Conferenza`(*serial*)(*totale*): Chiave primaria per una conferenza. `Titolo`(*Text*) (*totale*): Specifica il titolo della conferenza scientifica. `Descrizione`(*Text*)(*parziale*): Fornisce una descrizione della conferenza scientifica. `Inizio`(*Timestamp*)(*totale*): Indica l'inizio della conferenza. `Fine` (*Timestamp*)(*totale*) : Indica la fine della conferenza.
Ente	Tabella delle istituzioni	`Id_Ente`(*serial*)(*totale*): Identificatore primario di una istituzione. `Nome`(*Text*)(*totale*): Nome dell'istituzione. `Sigla`(*Varchar(7)*)(*totale*) : Sigla dell'istituzione.
Evento	Eventi sociali presenti all'interno di una conferenza.	`Id_Evento`(*Serial*)(*Totale*): Identificatore primario per un evento. `Tipologia`(`text`)(totale): Stringa descrittiva della tipologia dell'evento. `Inizio`(*Timestamp*)(*totale*): Indica l'inizio dell'evento.

Continua nella prossima pagina

Continua dalla pagina precedente

Classe	Descrizione	Attributi
		`Fine` (*Timestamp*)(*totale*) : Indica la fine dell'evento.
Indirizzo	Tabella degli indirizzi per ogni sede	`Id_Indirizzo`(*serial*)(*totale*): Chiave primaria. `Via`(*text*)(*parziale*): nome della via. `Civico`(text)(parziale): civico della sede. `Cap`(char(5))(parziale): codice di avviamento postale `Città`(text)(parziale): città della sede. `Provincia`(varchar(2)): provincia della città. `Stato`(text)(parziale): stato della sede.
Intervallo	Descrittore degli intervalli presenti all'interno di una conferenza scientifica.	`Id_Intervallo`(*Serial*)(*Totale*): Identificatore primario per un evento. `Tipologia`(*Intervallo_ST*)(totale): Specifica il tipo di intervallo (pranzo o coffee break). `Inizio`(*Timestamp*)(*totale*): Indica l'inizio dell'intervallo. `Fine` (*Timestamp*)(*totale*) : Indica la fine dell'intervallo.
Intervento	Descrittore degli interventi che si tengono all'interno delle sessioni.	`Id_Intervento`(*Serial*)(*totale*): Identificatore primario di un intervento. `Titolo`(*Text*) (*totale*): Specifica il titolo dell'intervento. `Abstract`(*Text*)(*parziale*): Fornisce una descrizione dell'intervento. `Inizio`(*Timestamp*)(*totale*): Indica l'inizio dell'intervento. `Fine` (*Timestamp*)(*totale*) : Indica la fine dell'intervento.
Organizzatore	Descrittore dei membri dei comitati.	`Id_Organizzatore`(*serial*)(*Totale*): Identificatore principale di un organizzatore. `Nome`(*text*)(*totale*): nome dell'organizzatore. `Cognome`(*text*)(*totale*): cognome dell'organizzatore.

Continua nella prossima pagina

Continua dalla pagina precedente

Classe	Descrizione	Attributi
		`Titolo`(*Titolo_ST*)(*parziale*): Titolo accademico dell'organizzatore `Email`(*Text*)(*Parziale*): Email dell'organizzatore
Partecipante	Descrittore dei partecipanti delle sessioni.	`Id_Partecipante`(*serial*)(*Totale*): Identificatore principale di un partecipante. `Nome`(*text*)(*totale*): nome dell'organizzatore. `Cognome`(*text*)(*totale*): cognome dell'organizzatore. `Titolo`(*Titolo_ST*)(*parziale*): Titolo accademico del partecipante. `Email`(*Text*)(*Parziale*): Email del partecipante.
Programma	Tabella dei programmi delle sessioni.	`Id_Programma`(*serial*)(*totale*): Identificatore principale dei programmi.
Sala	Tabella delle sale di ciascuna sede.	`Id_sala`(*serial*)(*totale*): identificatore principale di ciascuna sala. `Nome`(*Text*)(*totale*): nome della sala. `Capienza`(*int*)(*totale*): capienza della sala.
Sede	Descrizione delle sedi che ospitano le conferenze	`Id_Sede`(*Serial*)(*totale*) : Identificatore principale delle sedi. `Nome`(*Text*)(*totale*): nome della sede.
Sessione	Tabella delle sessioni di ciascuna conferenza.	`Id_Sessione`(*Serial*)(*total*): Identificatore primario di una sessione. `Titolo`(*Text*) (*totale*): Specifica il titolo della sessione. `Inizio`(*Timestamp*)(*totale*): Indica l'inizio della sessione. `Fine` (*Timestamp*)(*totale*) : Indica la fine della sessione.
Speaker	Descrittore dei vari speaker delle sessioni.	`Id_Speaker`(*serial*)(*Totale*): Identificatore principale di uno speaker. `Nome`(*text*)(*totale*): nome dello speaker. `Cognome`(*text*)(*totale*): cognome dello speaker. `Titolo`(*Titolo_ST*)(*parziale*): Titolo accademico dello speaker.

Continua nella prossima pagina

Continua dalla pagina precedente

Classe	Descrizione	Attributi
		`Email`(*Text*)(*Parziale*): Email dello speaker.
Sponsor	Tabella degli sponsor	`Id_Sponsor`(*serial*)(*totale*): Identificatore primario di uno sponsor. `Nome`(*Text*)(*totale*): Nome dello sponsor.
Valuta	Tabella delle valute	`Iso` (*Char(3)*)(*totale*): codice univoco internazionale delle valute. `Nome`(*text*)(*totale*): nome della valuta. `Simbolo`(*char(1)*)(*totale*): simbolo della valuta.

A.2 Dizionario delle associazioni

Associazione	Descrizione	Classi coinvolte
Appartiene_A	Rappresenta l'appartenenza di un organizzatore ad una precisa istituzione.	**Organizzatore [0..*]**: indica l'organizzatore che appartiene all'ente. **Ente [0..1]** ruolo **in**: indica l'ente al quale appartiene un organizzatore.
Appartiene_A	Rappresenta l'appartenenza di un partecipante ad una precisa istituzione.	**Organizzatore [0..*]**: indica il partecipante che appartiene ad un ente. **Ente [0..1]** ruolo **istituzione**: indica l'ente al quale appartiene un partecipante.
Appartiene_A	Rappresenta l'appartenenza di uno speaker ad una precisa istituzione.	**Organizzatore [0..*]**: indica lo speaker che appartiene all'ente. **Ente [0..1]** ruolo **istituzione**: indica l'ente al quale appartiene uno speaker.
Comitato_Conferenza	Ogni conferenza è legata ai comitati che ne gestiscono l'organizzazione.	**Comitati [2..2]**: indica i due comitati nominati per la conferenza. **Conferenza [1..1]** ruolo **di**: ogni comitato appartiene ad una sola conferenza.

Continua nella pagina successiva

Continua dalla pagina precedente

Associazione	Descrizione	Classi coinvolte
Sponsorizzazione_Conferenza	Ogni conferenza ha varie sponsorizzazioni da parte degli Sponsor che contribuiscono alle spese generali.	**Sponsor [0..*]** **Conferenza [0..*]**
Svolta_In	Specifica l'ubicazione di una conferenza in una sede.	**Conferenza [0..*]** **Sede [1..1]**
Svolta_In	Specifica l'ubicazione di una sessione in una sala.	**Sessione [0..*]** **Sala [1..1]**
Coordina	Ogni sessione ha un coordinatore.	**Sessione [0..1]** **Organizzatore [1..1]**
Sessioni_Conferenza	Ogni conferenza è composta da una o più sessioni.	**Conferenza [1..1]** **Sessioni [0..*]**
Sale_Sede	Ogni sede è composta da una o più sedi.	**Sede [1..1]** **Sala [1..*]**
Programma_Sessione	Ogni sessione ha un programma	**Sessione[1..1]** **Programma [1..1]**
Programma_Intervento	Ogni programma è un composto di vari interventi	**Programma [1..1]** **Intervento [0..*]**
Programma_Intervallo	Ogni programma è un composto di vari intervalli	**Programma [1..1]** **Intervallo [0..*]**
Programma_Evento	Ogni programma è un composto di vari eventi sociali	**Programma [1..*]** **Evento [0..*]**
Partecipante_Sessione	Ogni sessione ha vari partecipanti che partecipano a varie sessioni	**Sessione [0..*]** **Partecipante [0..*]**
Speaker_Intervento	Ogni intervento ha un suo speaker che può effettuare vari interventi	**Intervento [0..*]**

Continua nella pagina successiva

Continua dalla pagina precedente

Associazione	**Descrizione**	**Classi coinvolte**
		Speaker [1..1]
Membro_Comitato	Ogni comitato è composto da vari organizzatori che appartengono a vari comitati	**Organizzatore [0..*]** **Comitato [0..*]**

A.3 Dizionario dei vincoli

Vincolo	**Tipo**	**Descrizione**
CHECK_PROGRAMMA	Interrelazionale	In un programma non devono esserci eventi, intervalli od interventi che si sovrappongono.
CHECK_DATA	Interrelazionale	La data di inizio e di fine di un intervallo, un intervento o un evento devono essere coerenti con quelli della sessione a cui appartengono.
CHECK_SEDE	Interrelazionale	La sala in cui si svolge una sessione deve appartenere alla sede in cui si svolge la conferenza della sessione.
CHECK_DATA_SESSIONE	Interrelazionale	La data di inizio e di fine di ogni sessione deve essere compresa tra l'inizio e la fine della propria conferenza.
CHECK_COORDINATORE	Interrelazionale	Il coordinatore di una sessione deve appartenere al comitato scientifico della conferenza.
CHECK_COMITATI	Intrarelazionale	Ogni volta che si modifica la tabella CONFERENZA bisogna controllare che i valori indicati per i comitati siano coerenti con la tipologia di comitato della colonna.
CHECK_SALA	Interrelazionale	Quando si inserisce una nuova sessione bisogna controllare che la sala indicata sia effettivamente disponibile e non occupata nei giorni indicati.
CHECK_ORGANIZZATORI	Interrelazionale	Gli organizzatori appartenenti ai comitati di una conferenza devono appartenere agli enti che organizzano quella conferenza.
CHECK_CAPIENZA	Interrelazionale	Ogni volta che si aggiunge un nuovo partecipante di una sessione bisogna controllare che non sia stata raggiunta la capienza della sala in cui si svolge la sessione.