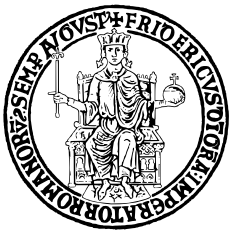
Università degli Studi di Napoli Federico II



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

Relatore
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*.

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:

Conferenza

id_Conferenza	Nome	Descrizione	Inizio	Fine	id_sede	id_com_locale	id_com_scientifico

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.

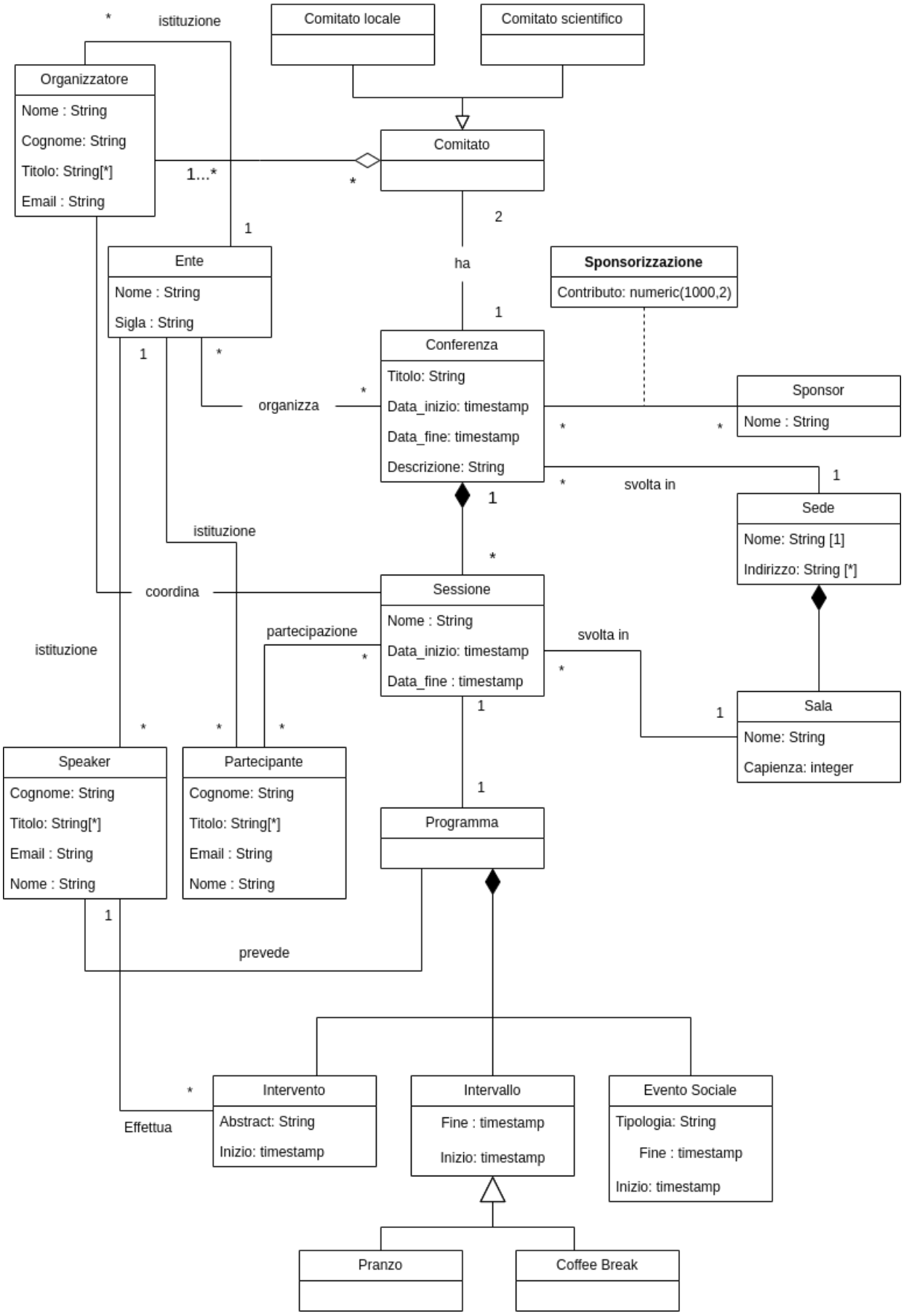


Figura 2.1: *Schema concettuale del problema*

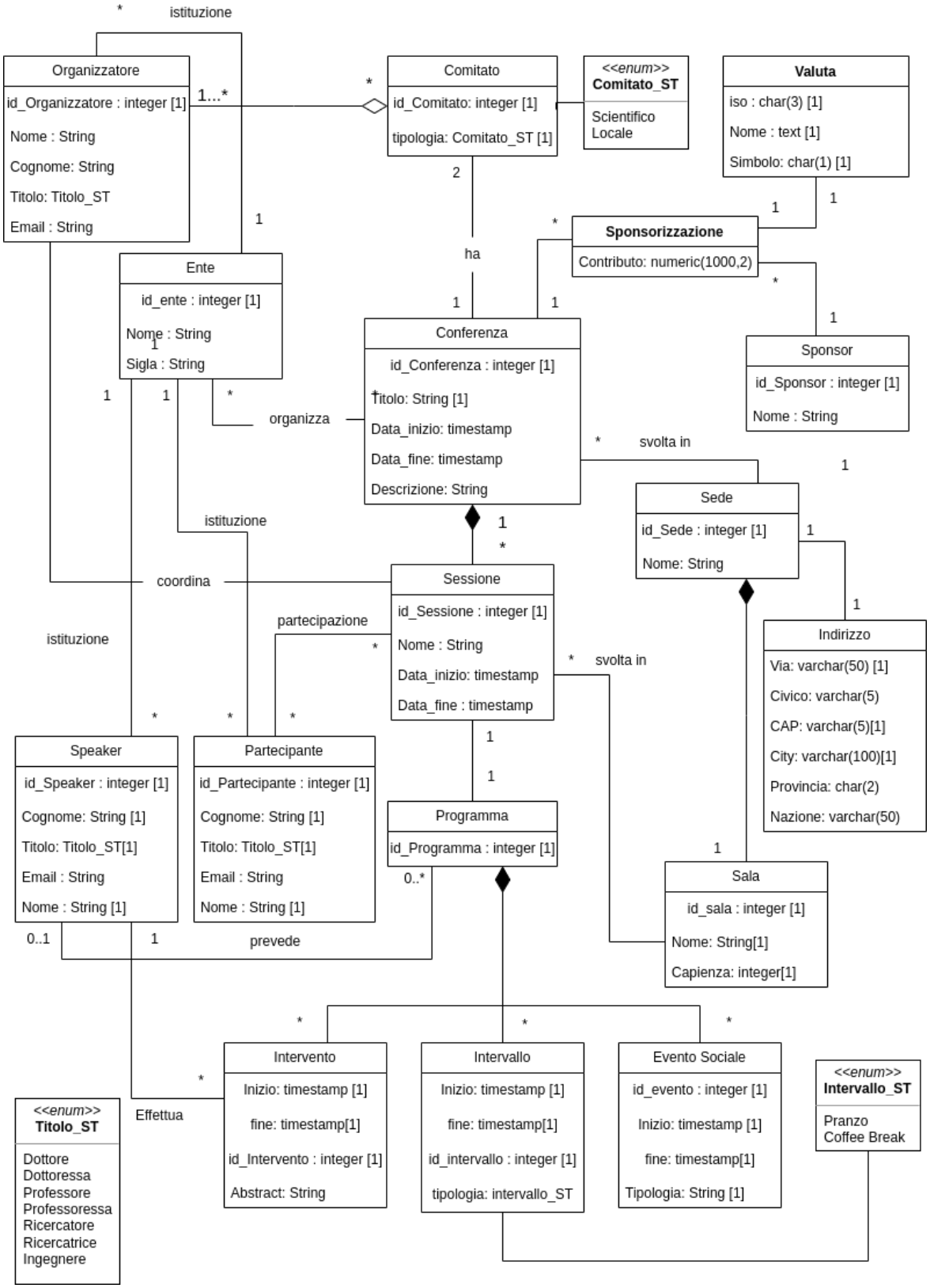
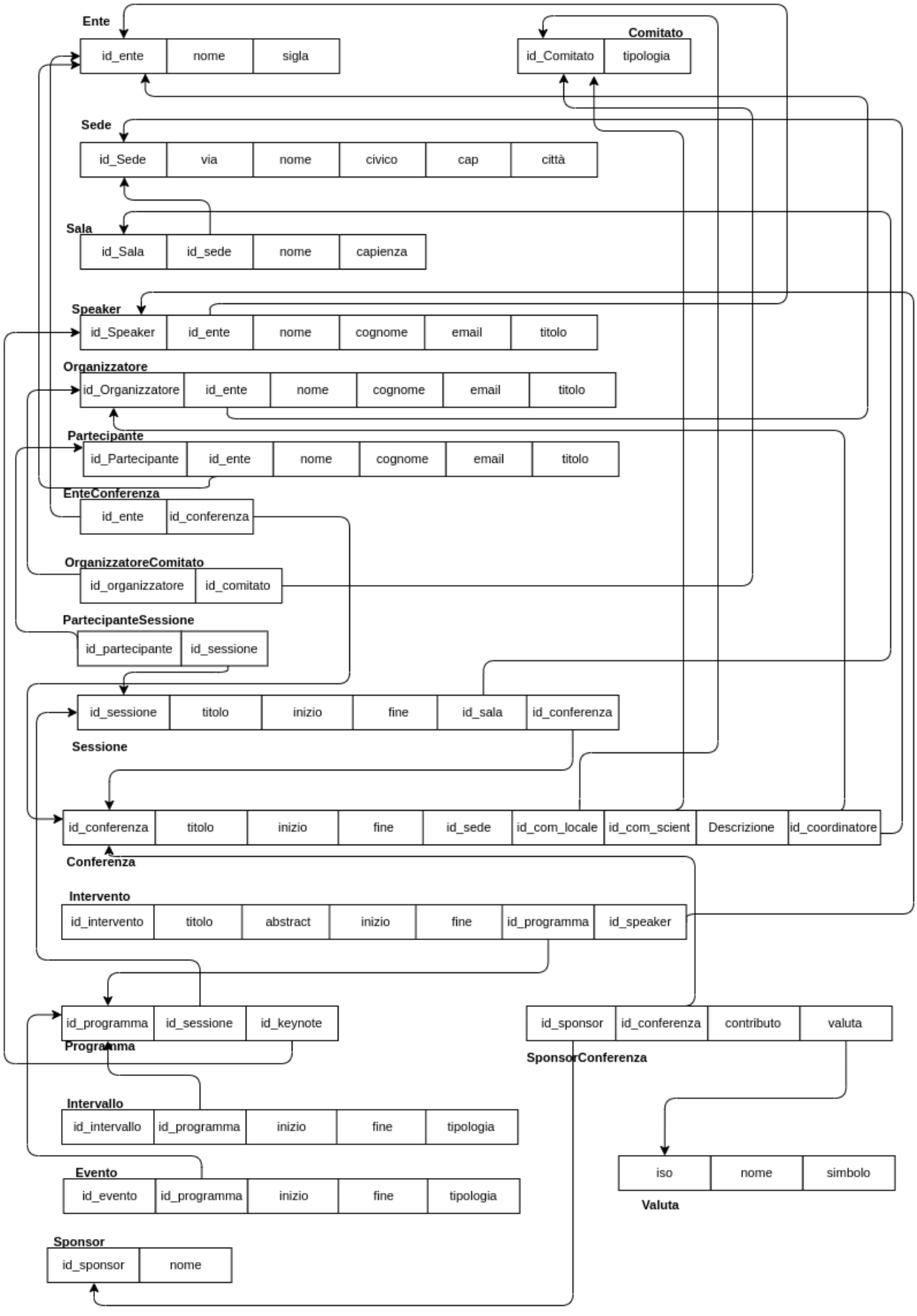


Figura 2.2: *Ristrutturazione dello schema concettuale*

Figura 2.3: *Schema logico*



Capitolo 3

Implementazione fisica

3.1 Definizione delle tabelle

```
create schema conference;
set search_path to conference;

create sequence ente_id_ente_seq;
create or replace function nextval_ente() returns text as $$
begin
    return 'IST'||to_char(nextval('ente_id_ente_seq'),'FM0000');
end;
$$
language plpgsql;

create table ente(
id_ente text primary key default nextval_ente(),
nome text not null unique,
sigla varchar(7) not null,
unique (nome,sigla)
);

create table indirizzo(
    id_indirizzo serial primary key,
    via text not null,
    civico varchar(5) not null,
    cap varchar(5) ,
    city text not null,
    provincia varchar(2) not null,
    nazione text
);

create table sede(
    id_sede serial primary key,
    nome text ,
    id_indirizzo integer references indirizzo(id_indirizzo) on
   delete set null
);
```

```
34
35
36  create table sponsor(
37      id_sponsor serial primary key,
38      nome text not null
39  );
40
41
42  create type comitato_st as enum ('locale','scientifico');
43  create table comitato(
44      id_comitato serial primary key,
45      tipologia comitato_st not null
46  );
47
48
49  create type titolo_st as enum ('Dottore','Dottoressa','Professore'
        ,'Professoressa','Assistente','Ricercatore','Ricercatrice','
        Ingegnere');
50  create table organizzatore(
51      id_organizzatore serial primary key,
52      nome text not null,
53      cognome text not null,
54      titolo titolo_st,
55      email text not null unique,
56      id_ente text references ente(id_ente) on delete cascade
57  );
58
59
60  create table sala(
61      id_sala serial primary key,
62      nome text not null,
63      capienza integer not null,
64      id_sede integer references sede(id_sede) on delete cascade
65  );
66
67
68  create table conferenza(
69      id_conferenza serial primary key,
70      titolo text not null,
71      descrizione text not null,
72      inizio timestamp not null,
73      fine timestamp not null,
74      id_sede integer references sede(id_sede) on delete set null,
75      comitato_s integer references comitato(id_comitato) on delete
        set null,
76      comitato_l integer references comitato(id_comitato) on delete
        set null,
77      -- Vincolo di integrita': la conferenza deve iniziare prima di
        finire
```

```
78      check (inizio <= fine),
79      -- Vincolo di integrita': la conferenza deve iniziare in
        futuro
80      check (inizio >= now())
81  );
82
83
84  create table partecipante(
85      id_partecipante serial primary key,
86      nome text not null,
87      cognome text not null,
88      titolo titolo_st,
89      email text not null unique,
90      id_ente text references ente(id_ente) on delete set null
91  );
92
93
94  create table sessione(
95      id_sessione serial primary key,
96      titolo text not null,
97      inizio timestamp not null,
98      fine timestamp not null,
99      id_coordinatore integer references organizzatore(
        id_organizzatore) on delete set null,
100     id_conferenza integer references conferenza(id_conferenza) on
        delete cascade,
101     id_sala integer references sala(id_sala) on delete set null,
102     -- Vincolo di integrita': la sessione deve iniziare prima di
        finire
103     check (inizio <= fine)
104 );
105
106 create table partecipazione(
107     id_partecipante integer references partecipante(
        id_partecipante) on delete cascade,
108     id_sessione integer references sessione(id_sessione) on delete
        cascade,
109     -- Vincolo di unicita': un partecipante puo' partecipare ad
        una sessione una sola volta
110     unique (id_partecipante,id_sessione)
111 );
112
113 create table ente_conferenza(
114     id_ente text references ente(id_ente) on delete cascade,
115     id_conferenza integer references conferenza(id_conferenza) on
        delete cascade,
116     -- Vincolo di unicita': un ente puo' organizzare una
        conferenza una sola volta
117     unique (id_ente,id_conferenza)
```

```
);

create table valuta(
    iso char(3) primary key,
    nome text not null,
    simbolo text not null
);

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,
    -- Vincolo di unicita': uno sponsor puo' sponsorizzare una conferenza una sola volta
    unique (id_sponsor,id_conferenza)
);

create sequence speaker_id_speaker_seq;
create or replace function nextval_speaker() returns text as $$
begin
    return 'SPK'||to_char(nextval('speaker_id_speaker_seq'),'FM0000');
end;
$$
language plpgsql;

create table speaker(
    id_speaker text primary key default nextval_speaker(),
    nome text not null,
    cognome text not null,
    titolo titolo_st,
    email text not null unique,
    id_ente text references ente(id_ente) on delete set null
);

create sequence programma_id_programma_seq;
create or replace function nextval_programma() returns text  as $$
begin
    return 'PRG'||to_char(nextval('programma_id_programma_seq'),'FM0000');
end;
$$
language plpgsql;

create table programma(
    id_programma text primary key default nextval_programma(),
```

```
162     id_sessione integer references sessione(id_sessione) on delete
        cascade not null,
163     id_keynote text references speaker(id_speaker) on delete set
        null,
164     unique (id_programma, id_sessione)
165 );
166
167 create sequence intervento_id_intervento_seq;
168 create or replace function nextval_intervento() returns text  as
        $$
169 begin
170     return 'INT'||to_char(nextval('intervento_id_intervento_seq'),
        'FM0000');
171 end;
172 $$
173 language plpgsql;
174
175 create table intervento(
176     id_intervento text primary key default nextval_intervento(),
177     titolo text not null,
178     abstract text not null,
179     inizio timestamp not null,
180     fine timestamp not null,
181     id_speaker text references speaker(id_speaker) on delete
        cascade,
182     id_programma text references programma(id_programma) on delete
        cascade not null,
183     -- Vincolo di unicita': uno speaker puo' intervenire in una
        sessione una sola volta
184     unique (id_speaker,id_programma),
185     -- Vincolo di integrita': l'intervento deve iniziare prima di
        finire
186     check (inizio <= fine)
187 );
188
189 create type intervallo_st as enum ('pranzo','coffee break');
190 create sequence intervallo_id_intervallo_seq;
191 create or replace function nextval_intervallo() returns text  as
        $$
192 begin
193     return 'BRK'||to_char(nextval('intervallo_id_intervallo_seq'),
        'FM0000');
194 end;
195 $$
196 language plpgsql;
197
198 create table intervallo(
199     id_intervallo text primary key default nextval_intervallo(),
200     tipologia intervallo_st not null,
```

```
    inizio timestamp not null,
    fine timestamp not null,
    -- Vincolo di integrita': l'intervallo deve iniziare prima di finire
    check (inizio <= fine),
    id_programma text references programma(id_programma) on delete cascade not null
);

create sequence evento_id_evento_seq;
create or replace function nextval_evento() returns text  as $$
begin
    return 'EVT'||to_char(nextval('evento_id_evento_seq'),'FM0000');
end;
$$
language plpgsql;

create table evento(
    id_evento text primary key default nextval_evento(),
    tipologia text not null,
    inizio timestamp not null,
    fine timestamp not null,
    -- Vincolo di integrita': l'evento deve iniziare prima di finire
    check (inizio <= fine),
    id_programma text references programma(id_programma) on delete cascade not null
);

create table organizzatore_comitato(
    id_organizzatore integer references organizzatore(id_organizzatore) on delete cascade,
    id_comitato integer references comitato(id_comitato) on delete cascade,
    -- Vincolo di integrita': un organizzatore puo' far parte di un comitato una sola volta
    unique (id_organizzatore,id_comitato)
);
```

3.2 Definizione dei trigger

3.2.1 Check_Programma_Entry

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
    /* Controlliamo che il punto non si sovrapponga con
     * ciascun punto gia' presente nel programma */
    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 (inizio_intervento <= new.inizio OR
          fine_intervento >= new.fine) then
        raise exception 'Impossibile inserire un punto del
 programma in questo orario';
      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;
```

```
52      if (inizio_intervallo <= new.inizio OR fine_intervallo >=
    new.fine) then
53          raise exception 'Impossibile inserire un punto del
    programma in questo orario';
54      end if;
55    end loop;
56    close intervalli_cur;
57
58    open eventi_cur;
59    loop
60    fetch eventi_cur into evento_id;
61    exit when not found;
62    select inizio,fine into inizio_evento,fine_evento
63    from evento
64    where id_evento = evento_id;
65    if (inizio_evento <= new.inizio OR fine_evento >= new.fine)
    then
66    raise exception 'Impossibile inserire un punto del programma
    in questo orario';
67    end if;
68    end loop;
69    close eventi_cur;
70
71  return new;
72  end;
73  $$ language plpgsql;
74
75  create trigger check_programma
76  before insert or update on intervento
77  for each row
78  execute function check_programma();
79
80  create trigger check_programma
81  before insert or update on intervallo
82  for each row
83  execute function check_programma();
84
85  create trigger check_programma
86  before insert or update on evento
87  for each row
88  execute function check_programma();
```

Listato 3.1: *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:

```
create or replace function check_data()
returns trigger as $$
declare
  inizio_sessione timestamp;
  fine_sessione timestamp;
begin
  select inizio,fine into inizio_sessione,fine_sessione
  from sessione
  where id_sessione =
    (select id_sessione
    from programma
    where id_programma = new.id_programma);

  if (new.inizio < inizio_sessione OR new.fine > fine_sessione)
   then
    raise exception 'Gli orari non sono compatibili con quelli
   della sessione';
  end if;
  return new;
end;
$$ language plpgsql;

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();

create trigger check_data_evento
before insert or update on evento
for each row
execute function check_data();
```

Listato 3.2: *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;
```

```
8
9   create trigger create_programma_sessione
10  after insert on sessione
11  for each row
12  execute function create_programma_sessione();
```

Listato 3.3: *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:

```
1   create or replace function check_sala_sessione()
2   returns trigger as $$
3   declare
4     sede integer;
5     sala integer;
6   begin
7     select id_sede into sede
8     from conferenza
9     where id_conferenza = new.id_conferenza;
10
11    select id_sala into sala
12    from sala
13    where id_sala = new.id_sala;
14
15    if sala is null then -- Nulla da controllare
16      return new;
17    end if;
18
19    IF sala NOT IN (
20      SELECT id_sala
21      FROM sala
22      WHERE id_sede = sede
23    ) THEN
24      RAISE EXCEPTION 'La sala selezionata non appartiene alla
     sede della conferenza';
25    END IF;
26
27    return new;
28  end;
29  $$ language plpgsql;
30
31
32  create trigger check_sala_sessione
33  before insert or update on sessione
34  for each row
35  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:

```
1  create or replace function check_data_sessione()
2  returns trigger as $$
3  declare
4    inizio_conferenza timestamp;
5    fine_conferenza timestamp;
6  begin
7    select inizio, fine into inizio_conferenza,fine_conferenza
8    from conferenza
9    where id_conferenza = new.id_conferenza;
10
11   if (new.inizio < inizio_conferenza OR new.fine >
     fine_conferenza) then
12     raise exception 'Gli orari non sono compatibili con quelli
     della conferenza';
13   end if;
14   return new;
15 end;
16 $$ language plpgsql;
17
18 create trigger check_data_sessione
19 before insert or update on sessione
20 for each row
21 execute function check_data_sessione();
```

Listato 3.4: *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:

```
1  create or replace function check_coordinatore_sessione()
2  returns trigger as $$
3  declare
4  id_comitato_scientifico_conferenza integer;
5  begin
6
7  select comitato_s into id_comitato_scientifico_conferenza
8  from conferenza c
9  where c.id_conferenza = new.id_conferenza;
10
11 if (new.id_coordinatore is not null) then
12   if (id_comitato_scientifico_conferenza not in
13     (select id_comitato from organizzatore_comitato
14       where id_organizzatore = new.id_coordinatore))  then
```

```
15        raise exception 'Il coordinatore della sessione deve
      appartenere al comitato scientifico della conferenza';
16    end if;
17  end if;
18  return new;
19  end;
20  $$ language plpgsql;
21
22  create trigger check_coordinatore_sessione
23  before insert or update on sessione
24  for each row
25  execute function check_coordinatore_sessione();
```

Listato 3.5: *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:

```
1  create or replace function create_comitati_conferenza()
2  returns trigger as $$
3  declare
4    id_comitatoscientifico integer;
5    id_comitatolocale integer;
6  begin
7    insert into comitato(tipologia) values ('scientifico') returning
       id_comitato into id_comitatoscientifico;
8    insert into comitato(tipologia) values ('locale') returning
      id_comitato into id_comitatolocale;
9    update conferenza
10   set comitato_s = id_comitatoscientifico,
11   comitato_l = id_comitatolocale
12   where id_conferenza = new.id_conferenza;
13 return new;
14 end;
15 $$ language plpgsql;
16
17 create trigger create_comitati_conferenza
18 after insert on conferenza
19 for each row
20 execute function create_comitati_conferenza();
```

Listato 3.6: *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;

select id_comitato into id_comitato_locale
from comitato
where id_comitato = new.comitato_l;

IF id_comitato_scientifico IS NULL THEN
return new;
END IF;

IF id_comitato_locale IS NULL THEN
Return new;
END IF;

IF (select tipologia from comitato where id_comitato =
 id_comitato_scientifico) <> 'scientifico' THEN
RAISE EXCEPTION 'Il comitato scientifico deve essere scientifico
 ';
END IF;

IF (select tipologia from comitato where id_comitato =
 id_comitato_locale) <> 'locale' THEN
RAISE EXCEPTION 'Il comitato locale deve essere locale';
END IF;

return new;
end;
$$ language plpgsql;

create trigger check_comitati_conferenza
before update on conferenza
for each row
execute function check_comitati_conferenza();
```

Listato 3.7: *check_comitati_conferenza*

3.2.9 Check_Sala_Sessione_Unica

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

```
create or replace function check_sala_sessione_unica()
returns trigger as $$
declare
```

```
    inizio_sessione timestamp;
    fine_sessione timestamp;
    sessioni cursor for
      select id_sessione
      from sessione
      where id_sala = new.id_sala;
    sessione_id integer;
  begin

  open sessioni;
  loop
    fetch sessioni into sessione_id;
    exit when not found;
    select inizio,fine into inizio_sessione,fine_sessione
    from sessione
    where id_sessione = sessione_id;
    if (new.inizio >= inizio_sessione AND new.inizio <=
   fine_sessione)
    OR (new.fine >= inizio_sessione AND new.fine <= fine_sessione)
    then
      raise exception 'Impossibile ospitare due sessioni';
    end if;
  end loop;
  close sessioni;
  return new;
  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 3.8: *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 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 3.9: *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;
```

```
24      return new;
25    end;
26    $$ language plpgsql;
27
28    create trigger delete_sessioni_conferenza
29    before update on conferenza
30    for each row
31    execute function delete_sessioni_conferenza();
```

Listato 3.10: *delete_sessioni_conferenza*

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:

```
1   create or replace function show_conference_by_date(dataI date,
     dataF date)
2   returns setof conferenza as $$
3   begin
4   return query
5   select * from conferenza
6   where inizio >= start and fine <= dataF;
7   end;
8   $$ 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:

```
1 create or replace function show_conferences_by_sede(sede int)
2 returns setof conferenza as $$
3 begin
4 return query
5 select * from conferenza
6 where id_sede = sede;
7 end;
8 $$ 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:

```
1 create or replace function
2 show_comitato_scientifico(conferenza int)
3 returns setof organizzatore as $$
```

```
begin
return query
select * from organizzatore
where id_organizzatore in (
  select id_organizzatore
  from organizzatore_comitato
  where id_comitato = (
    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:

```
create or replace function show_comitato_locale(conferenza int)
returns setof organizzatore as $$
begin
return query
select * from organizzatore
where id_organizzatore in (
  select id_organizzatore
  from organizzatore_comitato
  where id_comitato = (
    select id_comitato_locale
    from conferenza
    where id_conferenza = conferenza));
end;
$$ 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.

```
  create or replace function show_partecipanti(conferenza int)
  returns setof partecipante as $$
  begin
  return query
  select * from partecipante
  where id_partecipante in (
    select id_partecipante
    from partecipazione
    where id_sessione in (
      select id_sessione
      from sessione
      where id_conferenza = conferenza));
```

```
13  end;
14  $$ 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.

```
1   create or replace function show_sessioni(conferenza int)
2   returns setof sessione as $$
3   begin
4   return query
5   select * from sessione
6   where id_conferenza = conferenza
7   order by inizio;
8   end;
9   $$ 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:

```
1   create or replace function
2   show_interventi_sessione(sessione int)
3   returns table
4   (
5     titolo text,
6     inizio timestamp,
7     fine timestamp,
8     abstract text,
9     speaker text
10  )  as $$
11  declare
12    programma int;
13  begin
14    select id_programma into programma
15    from programma
16    where id_sessione = sessione;
17
18    select titolo,inizio,fine,abstract, s.nome || ' ' || s.cognome
      as speaker
19    from intervento i join speaker s on i.id_speaker = s.
     id_speaker
20    where i.id_programma = programma
21    order by inizio;
22  end;
23  $$ 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 int)
returns table
(
  tipologia intervallo_st,
  inizio timestamp,
  fine timestamp
)
as $$
declare
programma int;
begin
select id_programma into programma
from programma
where id_sessione = sessione;

select 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:

```
create or replace function
show_eventi_sociali_sessione(sessione int)
returns table
(id_evento text,
tipologia text,
inizio timestamp,
fine timestamp)
as $$
declare
programma int;
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:

```
create or replace function show_keynote_sessione(sessione int)
returns table(
id_speaker text,
nome text,
cognome text,
titolo text,
email text,
ente text)
as $$
declare
speaker_id text;
begin
select id_programma into programma
from programma
where id_sessione = sessione;

select id_keynote into speaker_id
from programma
where id_sessione = sessione;

if not found then
  raise notice 'Keynote non presente';
else
  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 = speaker_id;
end if;
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:

```
  CREATE OR REPLACE FUNCTION
  show_programma(sessione int)
  RETURNS TABLE
  (
    id_entry text,
    appuntamento text,
```

```
    inizio timestamp,
    fine timestamp,
    descrizione text,
    speaker text
)
AS $$
DECLARE
    programma text;
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,int,interval)

```
create or replace procedure
add_intervento(titolo text, abstract text, speaker text,
 sessione int,durata interval)
as $$
declare
programma text;
id_entry text;
query text;
category text;
fine_prev timestamp;
begin
select id_programma into programma
from programma
where id_sessione = sessione;

select id,appuntamento,max(fine) into id_entry,category,
 fine_prev
from show_programma(sessione)
GROUP BY id, appuntamento;

if(fine_prev is null) then
fine_prev := (select inizio from sessione where id_sessione =
 session);
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,int,interval)

```
create or replace procedure
add_intervallo(tipologia text,sessioneP int,durata interval)
as $$
declare
programma text;
```

```
id_entry text;
query text;
category text;
fine_prev timestamp;
begin
select id_programma into programma
from programma
where id_sessione = sessioneP;

select id,appuntamento, max(fine) into id_entry,category,
 fine_prev
from show_programma(sessioneP)
GROUP BY id, appuntamento;

if(fine_prev is null) then
fine_prev := (select inizio from sessione where id_sessione =
 sessioneP);
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,int,interval)

```
create or replace procedure
add_evento(tipologia text, session int, durata interval)
as $$
declare
programma text;
id_entry text;
query text;
category text;
fine_prev timestamp;
begin
select id_programma into programma
from programma
where id_sessione = session;

select id, appuntamento, max(fine) into id_entry,category,
 fine_prev
from show_programma(session)
GROUP BY id, appuntamento;
```

```
18
19  if(fine_prev is null) then
20  fine_prev := (select inizio from sessione where id_sessione =
     session);
21  end if;
22
23  insert into evento(tipologia,id_programma,inizio,fine)
24  values (tipologia,programma,fine_prev,fine_prev+durata);
25  raise notice 'Inserimento completato';
26  exception
27  when others then
28  raise notice '%', sqlerrm;
29  end;
30  $$ language plpgsql;
```

3.3.15 Add_Conferenza_Details(text,timestamp,timestamp,integer,text)

```
1   CREATE OR REPLACE FUNCTION add_conferenza_details(nome text,
     inizio timestamp, fine timestamp, sede integer, abstract text)
2   RETURNS integer AS $$
3   DECLARE
4   id integer;
5   BEGIN
6   INSERT INTO conferenza(titolo, inizio, fine, id_sede,
     descrizione)
7   VALUES (nome, inizio, fine, sede, abstract)
8   RETURNING id_conferenza INTO id;
9   raise notice 'Inserimento completato';
10  RETURN id;
11  EXCEPTION
12  WHEN OTHERS THEN
13  RAISE NOTICE 'Errore nell''inserimento di una conferenza: %',
     SQLERRM;
14  RETURN 0;
15  END;
16  $$ LANGUAGE plpgsql;
```

3.3.16 Add_ente(integer, integer)

```
1   create or replace procedure
2   add_ente(ente integer, conferenza integer)
3   as $$
4   begin
5   insert into ente_conferenza(id_ente,id_conferenza)
6   values (ente,conferenza);
7   raise notice 'Inserimento completato';
8   exception
9   when others then
```

```
raise notice '%', sqlerrm;
end;
$$ language plpgsql;
```

3.3.17 Add_Sponsorizzazione(integer,numeric,char(3),integer)

```
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)

```
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)

```
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
SELECT id_ente INTO ente_id FROM ente WHERE sigla = sigla_ente;

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 $$
declare
id_conferenza int;
begin
id_conferenza := add_conferenza_details(nome,inizio,fine,sede,descrizione);
call add_enti(id_conferenza,sigle);
exception
when others then
raise notice '%', sqlerrm;
end;
$$ language plpgsql;
```

3.3.22 Slitta_Conferenza(interval)

```
create or replace procedure
slitta_conferenza(id_conferenza integer, durata interval)
as $$
declare
  id_sessione integer;
  id_intervento text;
  id_evento text;
  id_intervallo text;
  sessioni cursor for
    select id_sessione
    from sessione
    where id_conferenza = id_conferenza;

  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 = id_conferenza);

  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 = id_conferenza);

  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 = id_conferenza);
begin

  alter table conferenza disable trigger check_data_conferenza;
  alter table sessione disable trigger check_data_sessione;
  alter table interventi disable trigger check_data_intervento;
  alter table intervallo disable trigger check_data_intervallo;
  alter table evento disable trigger check_data_evento;
  alter table conferenza disable trigger
```

```
  delete_sessioni_conferenza;

   update conferenza
   set inizio = inizio + durata, fine = fine + durata
   where id_conferenza = id_conferenza;

   open sessioni;
   loop
     fetch sessioni into id_sessione;
     exit when not found;

     update sessione
     set inizio = inizio + durata, fine = fine + durata
     where id_sessione = id_sessione;

     open interventi;
     loop
       fetch interventi into id_intervento;
       exit when not found;

       update intervento
       set inizio = inizio + durata, fine = fine + durata
       where id_intervento = id_intervento;
     end loop;
     close interventi;

     open intervalli;
     loop
       fetch intervalli into id_intervallo;
       exit when not found;

       update intervallo
       set inizio = inizio + durata, fine = fine + durata
       where id_intervallo = id_intervallo;
     end loop;
     close intervalli;

     open eventi;
     loop
       fetch eventi into id_evento;
       exit when not found;

       update evento
       set inizio = inizio + durata, fine = fine + durata
       where id_evento = id_evento;
     end loop;
     close eventi;
   end loop;
   close sessioni;
```

```
    raise notice 'Slittamento completato';

    alter table conferenza enable trigger check_data_conferenza;
    alter table sessione enable trigger check_data_sessione;
    alter table evento enable trigger check_data_evento;
    alter table intervallo enable trigger check_data_intervallo;
    alter table interventi enable trigger check_data_intervento;
  exception
    when others then
      raise notice '%', sqlerrm;
  end;
  $$ 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;
```

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**