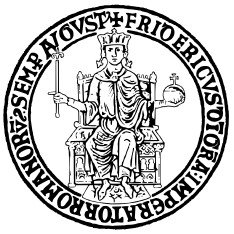
Università degli Studi di Napoli Federico II



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.

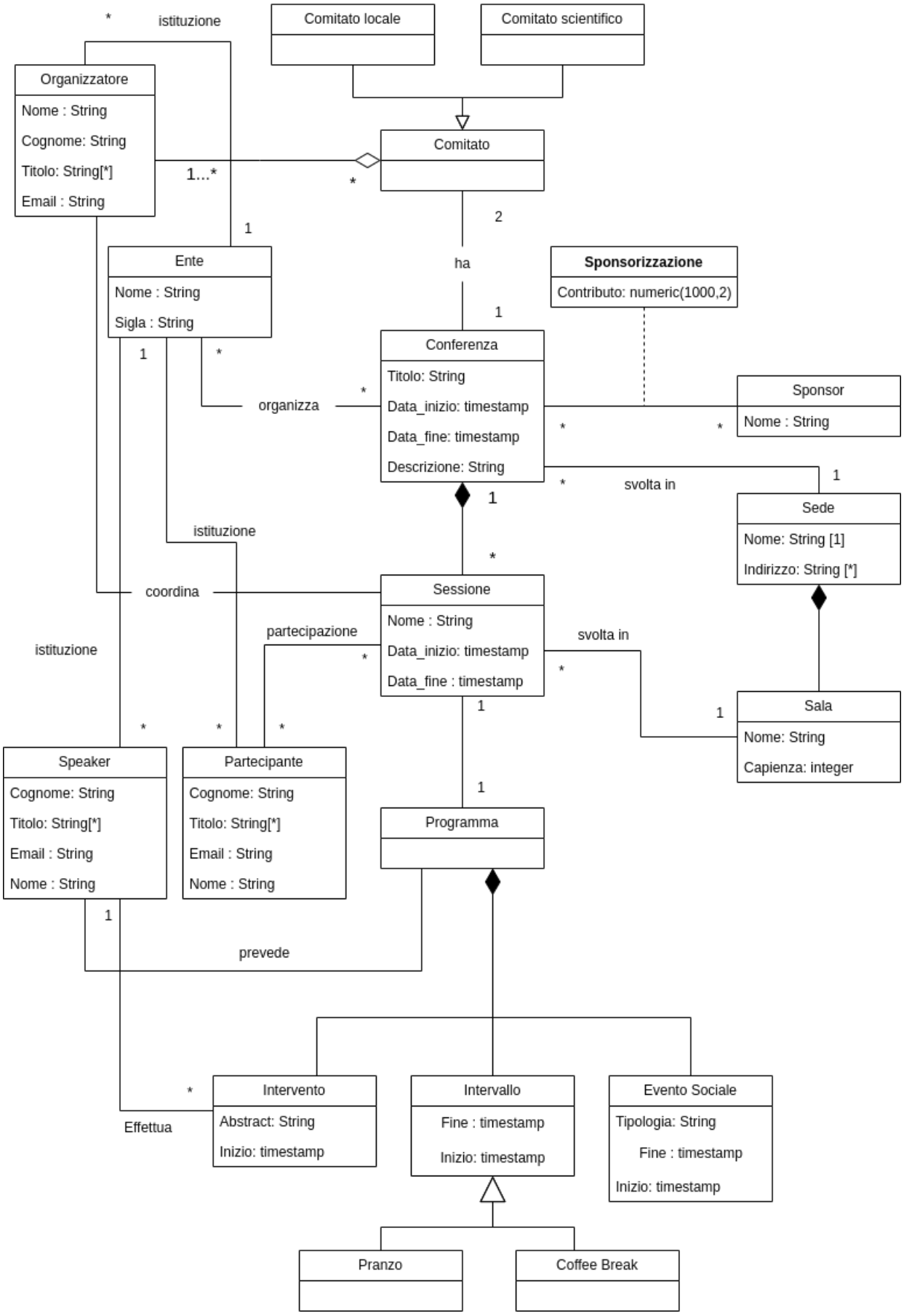


Figura 2.1: *Schema concettuale del problema*

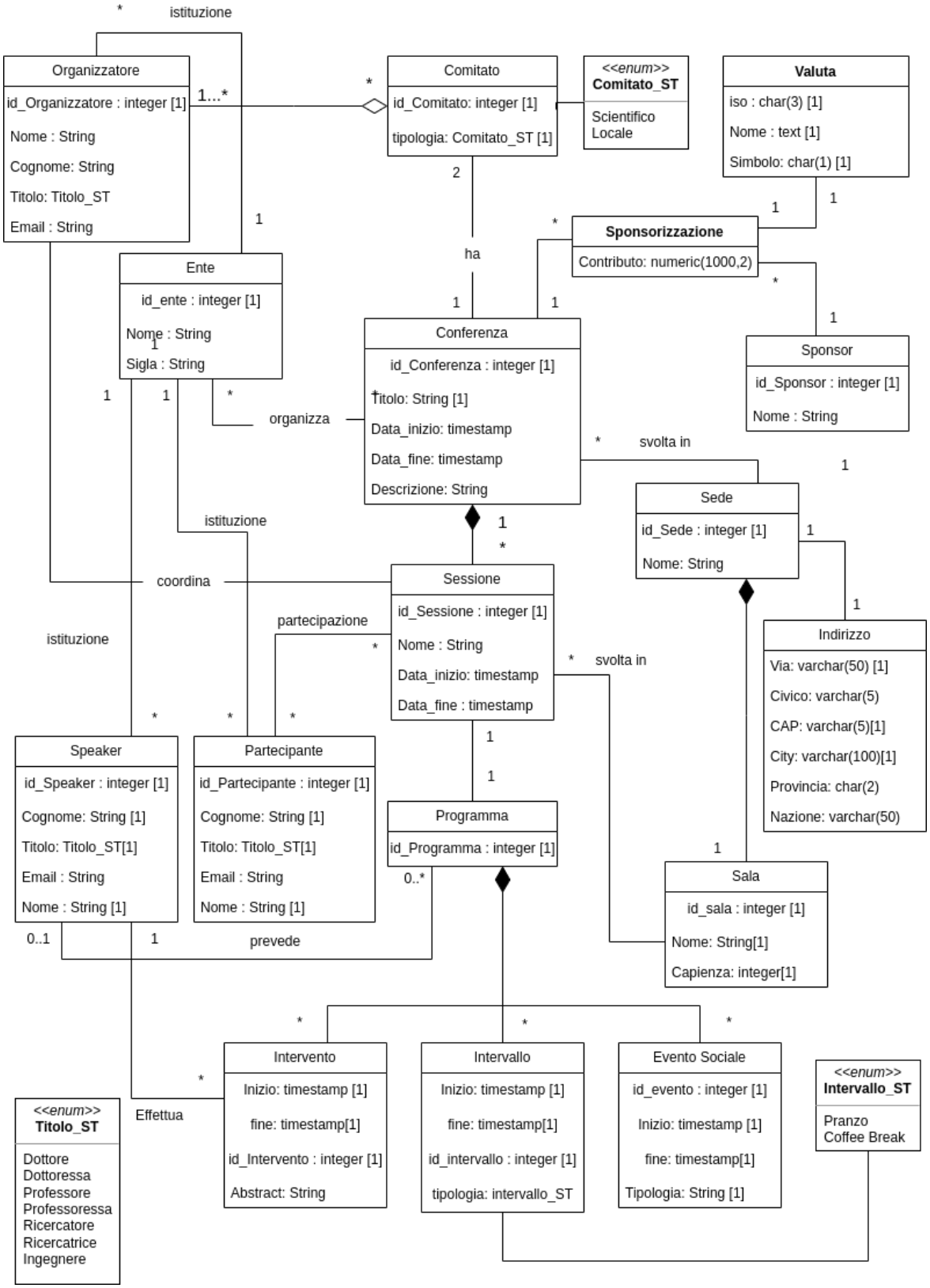
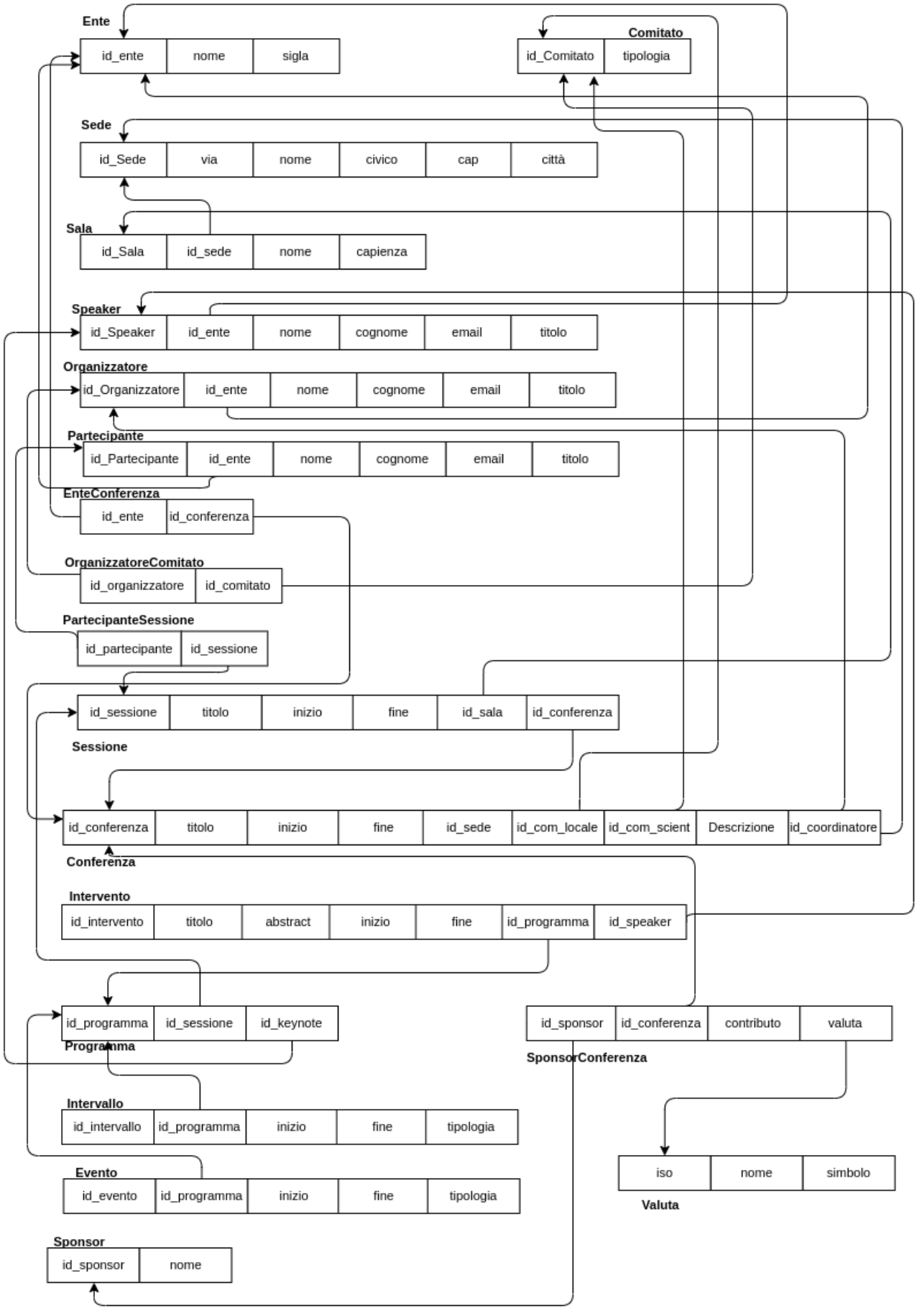


Figura 2.2: *Ristrutturazione dello schema concettuale*

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
);

create table sponsor(
    id_sponsor serial primary key,
    nome text not null
);

create type comitato_st as enum ('locale','scientifico');
create table comitato(
    id_comitato serial primary key,
```

```
    tipologia comitato_st not null
);

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 text references ente(id_ente) on delete cascade
);

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
);

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,
    check (inizio <= fine),
    check (inizio >= now())
);

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 text references ente(id_ente) on delete set null
);

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)
);

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)
```

```
);

create table ente_conferenza(
    id_ente text references ente(id_ente) on delete cascade,
    id_conferenza integer references conferenza(id_conferenza) on delete
    cascade,
    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,
    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 cascade NOT 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(),
    id_sessione integer references sessione(id_sessione) on delete cascade not
    null,
    id_keynote text references speaker(id_speaker) on delete set null,
    unique (id_programma, id_sessione)
);

create sequence intervento_id_intervento_seq;
create or replace function nextval_intervento() returns text  as $$
begin
```

```
157     return 'INT'||to_char(nextval('intervento_id_intervento_seq'),'FM0000');
158 end;
159 $$
160 language plpgsql;
161 
162 create table intervento(
163     id_intervento text primary key default nextval_intervento(),
164     titolo text not null,
165     abstract text not null,
166     inizio timestamp not null,
167     fine timestamp not null,
168     id_speaker text references speaker(id_speaker) on delete cascade,
169     id_programma text references programma(id_programma) on delete cascade not
        null,
170     unique (id_speaker,id_programma),
171     check (inizio <= fine)
172 );
173 
174 create type intervallo_st as enum ('pranzo','coffee break');
175 create sequence intervallo_id_intervallo_seq;
176 create or replace function nextval_intervallo() returns text  as $$
177 begin
178     return 'BRK'||to_char(nextval('intervallo_id_intervallo_seq'),'FM0000');
179 end;
180 $$
181 language plpgsql;
182 
183 create table intervallo(
184     id_intervallo text primary key default nextval_intervallo(),
185     tipologia intervallo_st not null,
186     inizio timestamp not null,
187     fine timestamp not null,
188     check (inizio <= fine),
189     id_programma text references programma(id_programma) on delete cascade not
        null
190 );
191 
192 create sequence evento_id_evento_seq;
193 create or replace function nextval_evento() returns text  as $$
194 begin
195     return 'EVT'||to_char(nextval('evento_id_evento_seq'),'FM0000');
196 end;
197 $$
198 language plpgsql;
199 
200 create table evento(
201     id_evento text primary key default nextval_evento(),
202     tipologia text not null,
203     inizio timestamp not null,
204     fine timestamp not null,
205     check (inizio <= fine),
206     id_programma text references programma(id_programma) on delete cascade not
        null
207 );
208 
209 create table organizzatore_comitato(
210     id_organizzatore integer references organizzatore(id_organizzatore) on
        delete cascade,
211     id_comitato integer references comitato(id_comitato) on delete cascade,
212     unique (id_organizzatore,id_comitato)
213 );
```

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 text;
intervallo_id text;
evento_id text;
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 questo intervento 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;
if (new.inizio>=inizio_intervallo AND new.fine<=fine_intervallo) then
raise exception 'Impossibile inserire questo intervallo in questo orario';
end if;
end loop;
close intervalli_cur;
open eventi_cur;
loop
fetch eventi_cur into evento_id;
exit when not found;
select inizio,fine into inizio_evento,fine_evento
from evento
where id_evento = evento_id;
if (new.inizio>=inizio_evento AND new.fine<=fine_evento) then
raise exception 'Impossibile inserire questo evento in questo orario';
end if;
end loop;
```

```
close eventi_cur;
return new;
end;
$$ language plpgsql;
```

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;
```

```
6   end;
7   $$ 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:

```
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 'Gli orari non sono compatibili con quelli della
   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 3.4: *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:

```
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 3.5: *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:

```
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 3.6: *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
```

```
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 3.7: *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()
2   returns trigger as $$
3   declare
4     inizio_sessione timestamp;
5     fine_sessione timestamp;
6     sessioni cursor for
7       select id_sessione
8       from sessione
9       where id_sala = new.id_sala;
10    sessione_id integer;
11  begin
12
13  open sessioni;
14  loop
15    fetch sessioni into sessione_id;
16    exit when not found;
17    select inizio,fine into inizio_sessione,fine_sessione
18    from sessione
19    where id_sessione = sessione_id;
20    if (new.inizio >= inizio_sessione AND new.inizio <= fine_sessione)
21    OR (new.fine >= inizio_sessione AND new.fine <= fine_sessione) then
22      raise exception 'Impossibile ospitare due sessioni';
23    end if;
24  end loop;
25  close sessioni;
26  return new;
27  end;
28  $$ language plpgsql;
29
30  create trigger check_sala_sessione_unica
31  before insert or update on sessione
32  for each row
33  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;
    return new;
  end;
  $$ language plpgsql;

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

Listato 3.10: *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 >= start 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 int)
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 int)
returns setof organizzatore as $$
begin
return query
select * from organizzatore
where id_organizzatore in (
  select id_organizzatore
  from organizzatore_comitato
  where id_comitato = (
    select id_comitato_scientifico
    from conferenza
    where id_conferenza = conferenza));
end;
$$ 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 $$
begin
return query
select * from partecipante
where id_partecipante in (
  select id_partecipante
  from partecipazione
  where id_sessione in (
    select id_sessione
    from sessione
    where id_conferenza = conferenza));
end;
$$ 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.

```
create or replace function show_sessioni(conferenza int)
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 int)
returns table
(
  titolo text,
  inizio timestamp,
  fine timestamp,
  abstract text,
  speaker text
)  as $$
declare
  programma text;
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 int)
  returns table
  (
    tipologia intervallo_st,
    inizio timestamp,
    fine timestamp
  )
  as $$
  declare
  programma text;
  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 text;
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)

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 int, durata interval)
```

```
as $$
declare
programma text;
id text;
query text;
category text;
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;
```

3.3.13 Add_Intervallo(text,int,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 int, durata interval)
as $$
declare
programma text;
id text;
query text;
category text;
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,int,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 int, durata interval)
as $$
declare
programma_id text;
id text;
query text;
category text;
fine_prev timestamp;
begin

select id_programma into programma_id
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 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)
  RETURNS integer AS $$
  DECLARE
```

```
id integer;
BEGIN
INSERT INTO conferenza(titolo, inizio, fine, id_sede, descrizione)
VALUES (nome, inizio, fine, sede, abstract)
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 text, 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
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(conferenza_id integer, durata interval)
as $$
declare
sessione_id integer;
intervento_id text;
evento_id text;
intervallo_id text;
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
where id_conferenza = conferenza_id;

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

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

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

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

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

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

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

update evento
set inizio = inizio + durata, fine = fine + durata
where id_evento = evento_id;
end loop;
close eventi;
end loop;
close sessioni;
alter table conferenza enable trigger all;
alter table sessione enable trigger all;
alter table intervento enable trigger all;
alter table intervallo enable trigger all;
alter table evento enable trigger all;
alter table programma enable trigger all;
raise notice 'Slittamento completato';
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;
```

3.3.24 Show_percentage_interventi(int,int)

```
create or replace function
show_percentage_interventi(mese int, anno int)
returns table
(
  sigla varchar(7),
  percentuale text
) as $$
declare
totale int;
begin
select count(*) into totale
from intervento
where date_part('month',inizio) = mese and date_part('year',inizio) = anno;

return query
select e.sigla, (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.sigla;
end;
$$
language plpgsql;
```

3.3.25 Show_percentage(int)

```
create or replace function
show_percentage_interventi(anno int)
returns table
(
  sigla varchar(7),
  percentuale text
```

```
) as $$
declare
totale int;
begin
select count(*) into totale
from intervento
where date_part('year',inizio) = anno;

return query
select e.sigla, (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.sigla;
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.