

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II



SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INFORMATICA

PROGETTO D'ESAME DI BASI DI DATI

PROGETTAZIONE ED IMPLEMENTAZIONE DI UNA  
BASE DI DATI RELAZIONALE PER LA GESTIONE  
DI CONFERENZE SCIENTIFICHE

**Relatrice**

Professoressa Mara SANGIOVANNI

**Candidati**

Antonio CAPORASO

matr: N86003458

Giorgio DI FUSCO

matr: N86004389

Anno Accademico 2022-2023

# Indice

<b>1</b>	<b>Traccia</b>	<b>6</b>
1.1	Output attesi dal committente . . . . .	6
<b>2</b>	<b>Progettazione</b>	<b>7</b>
2.1	Analisi dei dati . . . . .	7
2.2	Schema concettuale . . . . .	7
2.3	Ristrutturazione dello schema concettuale . . . . .	7
2.3.1	Rimozione degli attributi multivalore . . . . .	7
2.3.2	Rimozione classi di associazione . . . . .	7
2.3.3	Rimozione generalizzazioni . . . . .	7
2.3.4	Scelta degli identificatori principali . . . . .	8
2.4	Progettazione logica . . . . .	9
2.4.1	Traduzione delle classi . . . . .	9
2.4.2	Traduzione delle associazioni . . . . .	10
2.4.3	Schema logico . . . . .	12
<b>3</b>	<b>Implementazione fisica</b>	<b>16</b>
3.1	Definizione delle tabelle . . . . .	16
3.1.1	UTENTE . . . . .	16
3.1.2	ENTE . . . . .	16
3.1.3	INDIRIZZO . . . . .	16
3.1.4	SEDE . . . . .	17
3.1.5	SPONSOR . . . . .	17
3.1.6	COMITATO . . . . .	17
3.1.7	ORGANIZZATORE, SPEAKER, PARTECIPANTE . . . . .	17
3.1.8	SALA . . . . .	18
3.1.9	CONFERENZA . . . . .	18
3.1.10	SESSIONE . . . . .	18
3.1.11	PARTECIPAZIONE . . . . .	19
3.1.12	ENTE_CONFERENZA . . . . .	19
3.1.13	VALUTA . . . . .	19
3.1.14	SPONSOR_CONFERENZA . . . . .	19
3.1.15	PROGRAMMA . . . . .	19
3.1.16	INTERVENTO . . . . .	20
3.1.17	INTERVALLO . . . . .	20
3.1.18	EVENTO . . . . .	20
3.1.19	ORGANIZZATORE_COMITATO . . . . .	20
3.2	Definizione dei trigger . . . . .	21
3.2.1	Check_Programma . . . . .	21

3.2.2	Check_Data_Intervento, Check_Data_Intervallo, Check_Data_Evento	22
3.2.3	Create_Programma_Sessione	23
3.2.4	Check_Sala_Sessione	23
3.2.5	Check_Data_Sessione	24
3.2.6	Check_Coordinatore_Sessione	24
3.2.7	Create_Comitati_Conferenza	25
3.2.8	Check_Comitati_Conferenza	25
3.2.9	Check_Sala_Sessione_Unica	26
3.2.10	Check_Organizzatore_Comitato	27
3.2.11	Delete_Sessioni_Conferenza	27
3.2.12	Check_Capienza	28
3.3	Funzioni e procedure	29
3.3.1	Show_Conferenze_By_Date(DATE,DATE)	29
3.3.2	Show_Conferenze_By_Sede(integer)	29
3.3.3	Show_comitato_scientifico(integer)	29
3.3.4	Show_comitato_locale(integer)	29
3.3.5	Show_Partecipanti(INTEGER)	30
3.3.6	Show_Sessioni(integer)	30
3.3.7	Show_interventi_sessione(integer)	31
3.3.8	Show_intervalli_sessione(integer)	31
3.3.9	Show_eventi_sociali_sessione(integer)	31
3.3.10	Show_keynote_sessione(integer)	32
3.3.11	Show_Programma(integer)	32
3.3.12	Add_Intervento(text,text,text,INTEGER,interval)	33
3.3.13	Add_Intervallo(text,INTEGER,interval)	34
3.3.14	Add_Evento(text,INTEGER,interval)	35
3.3.15	Add_Conferenza_Details(text,TIMESTAMP,TIMESTAMP,integer,text)	35
3.3.16	Add_ente(integer, integer)	36
3.3.17	Add_Sponsorizzazione(integer,numeric,char(3),integer)	36
3.3.18	Add_Sessione(text,TIMESTAMP,TIMESTAMP, integer,integer)	36
3.3.19	Add_Partecipante(integer, integer)	37
3.3.20	Add_Enti(integer,text)	37
3.3.21	Add_Conferenza(text,TIMESTAMP,TIMESTAMP,integer, text, text)	37
3.3.22	Slitta_Conferenza(interval)	38
3.3.23	Show_members()	39
3.3.24	Show_percentage_interventi(INTEGER,INTEGER)	40
3.3.25	Show_percentage(INTEGER)	40
3.4	Definizione delle viste	41
3.4.1	SediView	41
3.4.2	Conferenze_Sede	41
3.4.3	Interventi_Speaker	41
3.4.4	Partecipanti_Sessione	41
3.4.5	Partecipanti_Conferenze	42
3.4.6	Sessioni	42
<b>A</b>	<b>Dizionari</b>	<b>43</b>
A.1	Dizionario dei dati	43
A.2	Dizionario delle associazioni	46
A.3	Dizionario dei vincoli	48

# Elenco delle figure

2.1	Schema concettuale del problema . . . . .	13
2.2	Ristrutturazione dello schema concettuale . . . . .	14
2.3	Schema logico . . . . .	15

# Elenco delle tabelle

2.1	Entità del problema . . . . .	8
-----	-------------------------------	---

# Elenco dei listati

3.1	Tabella: Utente . . . . .	16
3.2	Tabella: Ente . . . . .	16
3.3	Tabella: Indirizzo . . . . .	16
3.4	Tabella: Sede . . . . .	17
3.5	Tabella: Sponsor . . . . .	17
3.6	Tabella: Comitato . . . . .	17
3.7	Tabella: Organizzatore . . . . .	17
3.8	Tabella: Partecipante . . . . .	17
3.9	Tabella: Speaker . . . . .	18
3.10	Tabella: Sala . . . . .	18
3.11	Tabella: Conferenza . . . . .	18
3.12	Tabella: Sessione . . . . .	18
3.13	Tabella: Partecipazione . . . . .	19
3.14	Tabella: Ente_Conferenza . . . . .	19
3.15	Tabella: Valuta . . . . .	19
3.16	Tabella: Sponsor_Conferenza . . . . .	19
3.17	Tabella: Programma . . . . .	19
3.18	Tabella: Intervento . . . . .	20
3.19	Tabella: Programma . . . . .	20
3.20	Tabella: Evento . . . . .	20
3.21	Tabella: Organizzatore_Comitato . . . . .	20
3.22	check_programma_entry . . . . .	21
3.23	check_data_intervento . . . . .	22
3.24	create_programma_sessione . . . . .	23
3.25	check_data_sessione . . . . .	24
3.26	check_coordinatore_sessione . . . . .	24
3.27	create_comitati_conferenza . . . . .	25
3.28	check_comitati_conferenza . . . . .	25
3.29	Check_sala_sessione_unica . . . . .	26
3.30	Check_organizzatori_comitato . . . . .	27
3.31	delete_sessioni_conferenza . . . . .	27

# 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 appartenenza.

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 concettuale:

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

**Tabella 2.1:** *Entità del problema*

### 2.3.4 Scelta degli identificatori principali

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

<sup>1</sup>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

<u>Id_Indirizzo</u>	Via	Civico	CAP	City	Provincia	Nazione
---------------------	-----	--------	-----	------	-----------	---------

ENTE

<u>id_ente</u>	nome	sigla
----------------	------	-------

SEDE

<u>id_sede</u>	nome
----------------	------

SPONSOR

<u>id_Sponsor</u>	Nome
-------------------	------

COMITATO

<u>id_Comitato</u>	Tipologia
--------------------	-----------

ORGANIZZATORE

<u>id_Organizzatore</u>	Nome	Cognome	Titolo	Email
-------------------------	------	---------	--------	-------

SALA

<u>id_Sala</u>	Nome	Capienza
----------------	------	----------

CONFERENZA

<u>id_Conferenza</u>	Nome	Descrizione	Inizio	Fine
----------------------	------	-------------	--------	------

PARTECIPANTE

<u>id_Partecipante</u>	Nome	Cognome	Titolo	Email
------------------------	------	---------	--------	-------

SESSIONE

<u>id_Sessione</u>	Nome	Inizio	Fine
--------------------	------	--------	------

VALUTA

<u>Iso</u>	Nome	Simbolo
------------	------	---------

SPEAKER

<u>id_Speaker</u>	Nome	Cognome	Titolo	Email
-------------------	------	---------	--------	-------

PROGRAMMA

<u>Id_Programma</u>
---------------------

INTERVALLO

<u>id_Intervallo</u>	Tipologia	Inizio	Fine
----------------------	-----------	--------	------

INTERVENTO

<u>id_Intervento</u>	Titolo	Abstract	Inizio	Fine
----------------------	--------	----------	--------	------

EVENTO

<u>id_Evento</u>	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

<u>id_ente</u>	<u>id_conferenza</u>
----------------	----------------------

2. L'associazione ORGANIZZATORECOMITATO tra ORGANIZZATORE e COMITATO:

ORGANIZZATORECOMITATO

<u>id_organizzatore</u>	<u>id_comitato</u>
-------------------------	--------------------

3. L'associazione PARTECIPANTESESSIONE tra PARTECIPANTE e SESSIONE:

PARTECIPANTESESSIONE

<u>id_Partecipante</u>	<u>id_Sessione</u>
------------------------	--------------------

### 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

<u>id_Sala</u>	Nome	Capienza	<u>id_sede</u>
----------------	------	----------	----------------

- (b) Una conferenza è composta da più sessioni:

SESSIONE

<u>id_Sessione</u>	Nome	Inizio	Fine	<u>id_conferenza</u>
--------------------	------	--------	------	----------------------

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

INTERVALLO

<u>id_Intervallo</u>	Tipologia	Inizio	Fine	<u>id_programma</u>
----------------------	-----------	--------	------	---------------------

INTERVENTO

<u>id_Intervento</u>	Titolo	Abstract	Inizio	Fine	<u>id_programma</u>
----------------------	--------	----------	--------	------	---------------------

EVENTO

<u>id_Evento</u>	Tipologia	Inizio	Fine	<u>id_programma</u>
------------------	-----------	--------	------	---------------------

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

SPEAKER

<u>id_Speaker</u>	Nome	Cognome	Titolo	Email	<u>id_ente</u>
-------------------	------	---------	--------	-------	----------------

PARTECIPANTE

<u>id_Speaker</u>	Nome	Cognome	Titolo	Email	<u>id_ente</u>
-------------------	------	---------	--------	-------	----------------

ORGANIZZATORE

<u>id_Speaker</u>	Nome	Cognome	Titolo	Email	<u>id_ente</u>
-------------------	------	---------	--------	-------	----------------

3. Ogni intervento ha uno speaker che lo effettua:

INTERVENTO

<u>id_Intervento</u>	<u>id_speaker</u>	Titolo	Abstract	Inizio	Fine	<u>id_programma</u>
----------------------	-------------------	--------	----------	--------	------	---------------------

4. Una sala può ospitare più sessioni:

SESSIONE

<u>id_Sessione</u>	Nome	Inizio	Fine	<u>id_sala</u>	<u>id_conferenza</u>
--------------------	------	--------	------	----------------	----------------------

5. Una sede può ospitare più conferenze:

CONFERENZA

<u>id_Conferenza</u>	Nome	Descrizione	Inizio	Fine	<u>id_sede</u>
----------------------	------	-------------	--------	------	----------------

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

7. Ogni conferenza ha un utente che la crea;

CONFERENZA

<u>id_Conferenza</u>	Nome	Descrizione	Inizio	Fine	<u>id_sede</u>	<u>comitato_s</u>	<u>comitato_l</u>	<u>id_utente</u>
----------------------	------	-------------	--------	------	----------------	-------------------	-------------------	------------------

### Traduzione delle associazioni uno a uno

Si ha:

1. Ogni sessione ha un coordinatore:

SESSIONE

<u>id_Sessione</u>	Nome	Inizio	Fine	<u>id_sala</u>	<u>id_conferenza</u>	<u>id_coordinatore</u>
--------------------	------	--------	------	----------------	----------------------	------------------------

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

PROGRAMMA

<u>Id_Programma</u>	<u>id_Sessione</u>	<u>id_keynote</u>
---------------------	--------------------	-------------------

3. Ogni sede ha un indirizzo:

SEDE

<u>id_sede</u>	nome	indirizzo
----------------	------	-----------

### **2.4.3 Schema logico**

Nella Figura 2.3 è raffigurato lo schema logico risultante.

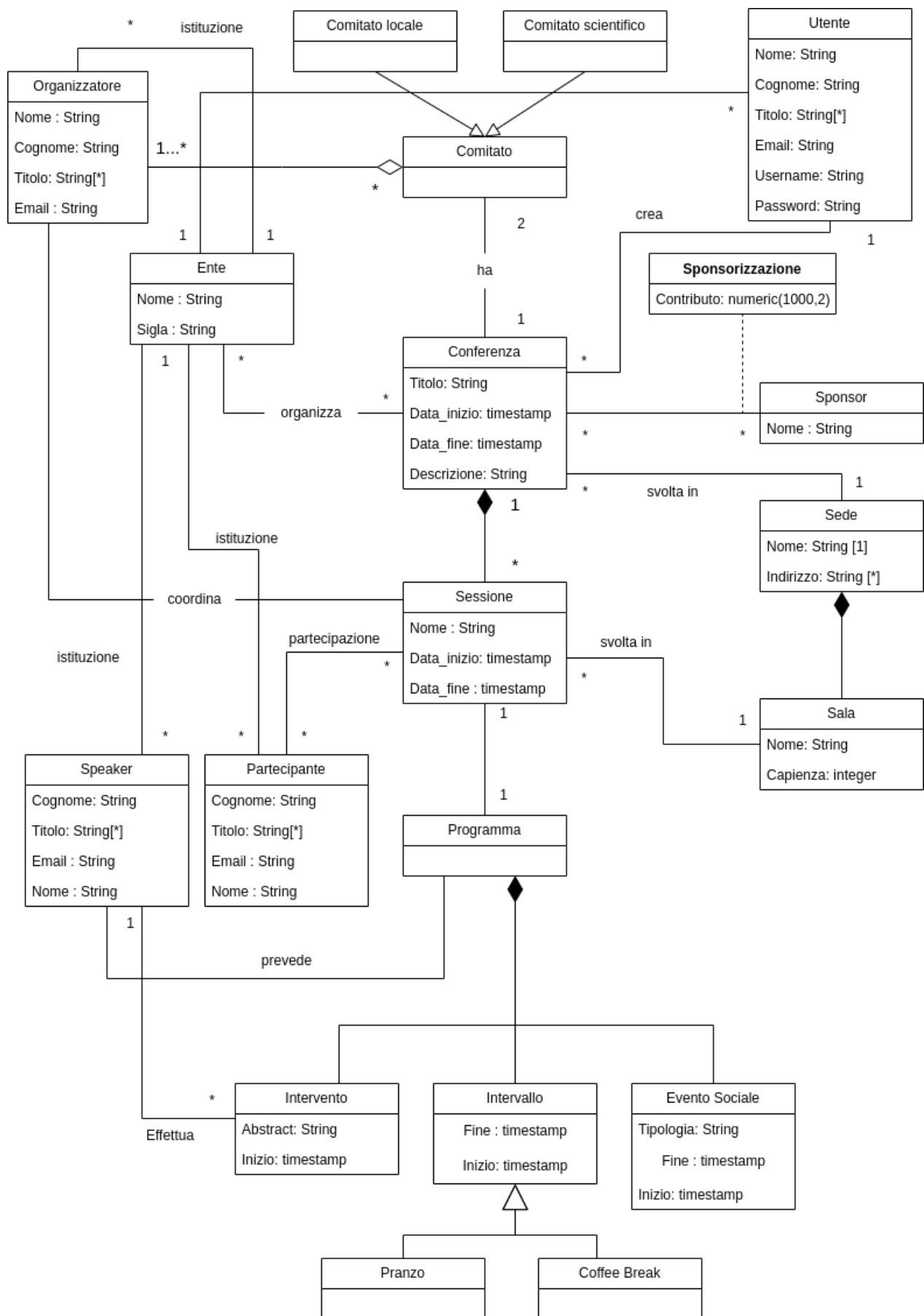


Figura 2.1: Schema concettuale del problema

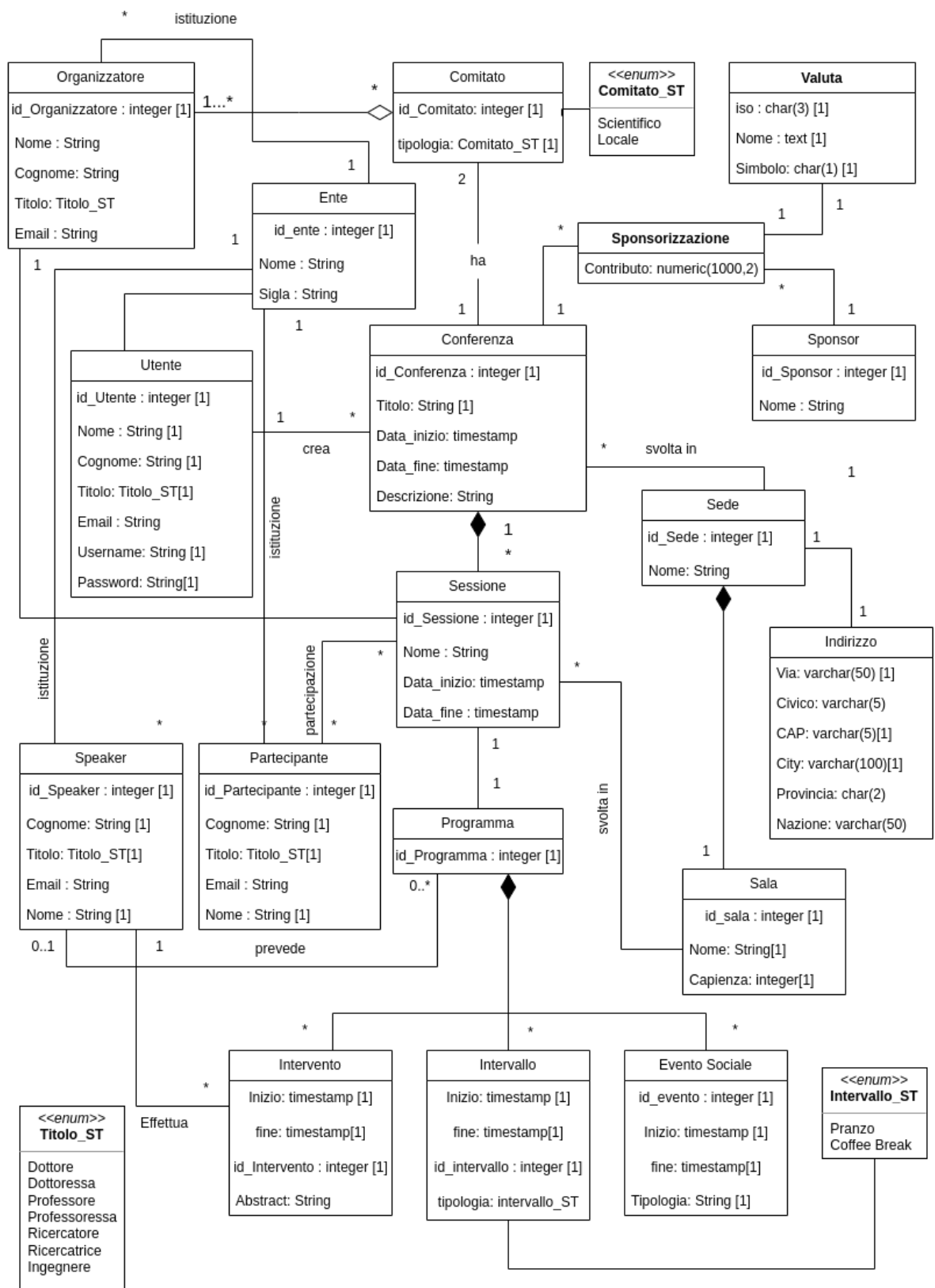
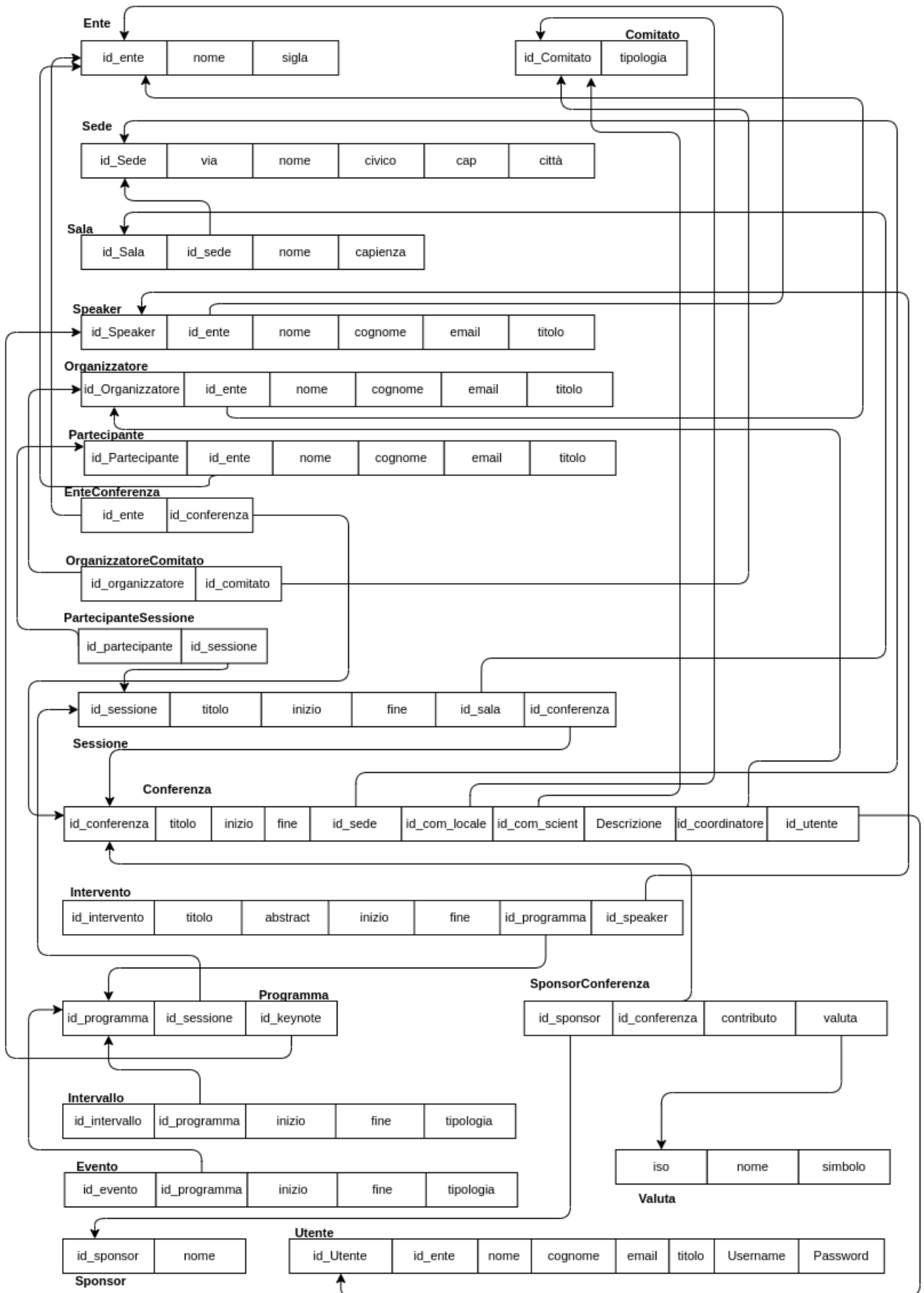


Figura 2.2: *Ristrutturazione dello schema concettuale*

Figura 2.3: Schema logico





## Capitolo 3

# Implementazione fisica

### 3.1 Definizione delle tabelle

#### 3.1.1 UTENTE

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

Listato 3.1: Tabella: Utente

#### 3.1.2 ENTE

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

Listato 3.2: Tabella: Ente

#### 3.1.3 INDIRIZZO

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

Listato 3.3: Tabella: Indirizzo

### 3.1.4 SEDE

```
1 CREATE TABLE sede(  
2 id_sede serial PRIMARY KEY,  
3 nome text ,  
4 id_indirizzo INTEGER REFERENCES indirizzo(id_indirizzo) ON delete SET null  
5 );
```

Listato 3.4: Tabella: Sede

### 3.1.5 SPONSOR

```
1 CREATE TABLE sponsor(  
2 id_sponsor serial PRIMARY KEY,  
3 nome text NOT NULL  
4 );
```

Listato 3.5: Tabella: Sponsor

### 3.1.6 COMITATO

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

```
1 CREATE TYPE comitato_st AS enum ('locale','scientifico');  
2 CREATE TABLE comitato(  
3 id_comitato serial PRIMARY KEY,  
4 tipologia comitato_st NOT NULL  
5 );
```

Listato 3.6: Tabella: Comitato

### 3.1.7 ORGANIZZATORE, SPEAKER, PARTECIPANTE

```
1 CREATE TYPE titolo_st AS enum ('Dottore','Dottorressa','Professore','  
    Professoressa','Assistente','Ricercatore','Ricercatrice','Ingegnere');  
2  
3 CREATE TABLE organizzatore(  
4 id_organizzatore serial PRIMARY KEY,  
5 nome text NOT NULL,  
6 cognome text NOT NULL,  
7 titolo titolo_st,  
8 email text NOT NULL UNIQUE,  
9 id_ente INTEGER REFERENCES ente(id_ente) ON delete cascade  
10 );
```

Listato 3.7: Tabella: Organizzatore

```
1 CREATE TABLE partecipante(  
2 id_partecipante serial PRIMARY KEY,  
3 nome text NOT NULL,  
4 cognome text NOT NULL,  
5 titolo titolo_st,  
6 email text NOT NULL UNIQUE,  
7 id_ente INTEGER REFERENCES ente(id_ente) ON delete SET null  
8 );
```

Listato 3.8: Tabella: Partecipante

```

1 CREATE TABLE speaker(
2 id_speaker serial PRIMARY KEY,
3 nome text NOT NULL,
4 cognome text NOT NULL,
5 titolo titolo_st,
6 email text NOT NULL UNIQUE,
7 id_ente INTEGER REFERENCES ente(id_ente) ON delete cascade NOT NULL
8 );

```

**Listato 3.9:** *Tabella: Speaker*

### 3.1.8 SALA

```

1 CREATE TABLE sala(
2 id_sala serial PRIMARY KEY,
3 nome text NOT NULL,
4 capienza INTEGER NOT NULL,
5 id_sede INTEGER REFERENCES sede(id_sede) ON delete cascade
6 );

```

**Listato 3.10:** *Tabella: Sala*

### 3.1.9 CONFERENZA

```

1 CREATE TABLE conferenza(
2 id_conferenza serial PRIMARY KEY,
3 titolo text NOT NULL,
4 descrizione text NOT NULL,
5 inizio TIMESTAMP NOT NULL,
6 fine TIMESTAMP NOT NULL,
7 id_sede INTEGER REFERENCES sede(id_sede) ON delete SET null,
8 comitato_s INTEGER REFERENCES comitato(id_comitato) ON delete SET null,
9 comitato_l INTEGER REFERENCES comitato(id_comitato) ON delete SET null,
10 id_utente INTEGER REFERENCES utente(id_utente) ON delete cascade,
11 check (inizio <= fine),
12 check (inizio >= now())
13 );

```

**Listato 3.11:** *Tabella: Conferenza*

### 3.1.10 SESSIONE

```

1 CREATE TABLE sessione(
2 id_sessione serial PRIMARY KEY,
3 titolo text NOT NULL,
4 inizio TIMESTAMP NOT NULL,
5 fine TIMESTAMP NOT NULL,
6 id_coordinatore INTEGER REFERENCES organizzatore(id_organizzatore) ON delete
  SET null,
7 id_conferenza INTEGER REFERENCES conferenza(id_conferenza) ON delete cascade,
8 id_sala INTEGER REFERENCES sala(id_sala) ON delete SET null,
9 check (inizio <= fine)
10 );

```

**Listato 3.12:** *Tabella: Sessione*

### 3.1.11 PARTECIPAZIONE

```
1 CREATE TABLE partecipazione(  
2 id_partecipante INTEGER REFERENCES partecipante(id_partecipante) ON delete  
  cascade,  
3 id_sessione INTEGER REFERENCES sessione(id_sessione) ON delete cascade,  
4 UNIQUE (id_partecipante,id_sessione)  
5 );
```

Listato 3.13: Tabella: Partecipazione

### 3.1.12 ENTE\_CONFERENZA

```
1 CREATE TABLE ente_conferenza(  
2 id_ente INTEGER REFERENCES ente(id_ente) ON delete cascade,  
3 id_conferenza INTEGER REFERENCES conferenza(id_conferenza) ON delete cascade,  
4 UNIQUE (id_ente,id_conferenza)  
5 );
```

Listato 3.14: Tabella: Ente\_Conferenza

### 3.1.13 VALUTA

```
1 CREATE TABLE valuta(  
2 iso char(3) PRIMARY KEY,  
3 nome text NOT NULL,  
4 simbolo text NOT NULL  
5 );
```

Listato 3.15: Tabella: Valuta

### 3.1.14 SPONSOR\_CONFERENZA

```
1 CREATE TABLE sponsor_conferenza(  
2 id_sponsor INTEGER REFERENCES sponsor(id_sponsor) ON delete cascade NOT NULL,  
3 contributo numeric(1000,2) NOT NULL,  
4 valuta char(3) REFERENCES valuta(iso) NOT NULL,  
5 id_conferenza INTEGER REFERENCES conferenza(id_conferenza) ON delete cascade  
  NOT NULL,  
6 UNIQUE (id_sponsor,id_conferenza)  
7 );
```

Listato 3.16: Tabella: Sponsor\_Conferenza

### 3.1.15 PROGRAMMA

```
1 CREATE TABLE programma(  
2 id_programma serial PRIMARY KEY,  
3 id_sessione INTEGER REFERENCES sessione(id_sessione) ON delete cascade NOT NULL  
  ,  
4 id_keynote INTEGER REFERENCES speaker(id_speaker) ON delete SET null,  
5 UNIQUE (id_programma, id_sessione)  
6 );
```

Listato 3.17: Tabella: Programma

### 3.1.16 INTERVENTO

```
1 CREATE TABLE intervento(  
2 id_intervento serial PRIMARY KEY,  
3 titolo text NOT NULL,  
4 abstract text NOT NULL,  
5 inizio TIMESTAMP NOT NULL,  
6 fine TIMESTAMP NOT NULL,  
7 id_speaker INTEGER REFERENCES speaker(id_speaker) ON delete cascade,  
8 id_programma INTEGER REFERENCES programma(id_programma) ON delete cascade NOT  
  NULL,  
9 UNIQUE (id_speaker,id_programma),  
10 check (inizio <= fine)  
11 );
```

Listato 3.18: Tabella: Intervento

### 3.1.17 INTERVALLO

```
1 CREATE TYPE intervallo_st AS enum ('pranzo','coffee break');  
2 CREATE TABLE intervallo(  
3 id_intervallo serial PRIMARY KEY,  
4 tipologia intervallo_st NOT NULL,  
5 inizio TIMESTAMP NOT NULL,  
6 fine TIMESTAMP NOT NULL,  
7 check (inizio <= fine),  
8 id_programma INTEGER REFERENCES programma(id_programma) ON delete cascade NOT  
  NULL  
9 );
```

Listato 3.19: Tabella: Programma

### 3.1.18 EVENTO

```
1 CREATE TABLE evento(  
2 id_evento serial PRIMARY KEY,  
3 tipologia text NOT NULL,  
4 inizio TIMESTAMP NOT NULL,  
5 fine TIMESTAMP NOT NULL,  
6 check (inizio <= fine),  
7 id_programma INTEGER REFERENCES programma(id_programma) ON delete cascade NOT  
  NULL  
8 );
```

Listato 3.20: Tabella: Evento

### 3.1.19 ORGANIZZATORE\_COMITATO

```
1 CREATE TABLE organizzatore_comitato(  
2 id_organizzatore INTEGER REFERENCES organizzatore(id_organizzatore) ON delete  
  cascade,  
3 id_comitato INTEGER REFERENCES comitato(id_comitato) ON delete cascade,  
4 UNIQUE (id_organizzatore,id_comitato)  
5 );
```

Listato 3.21: Tabella: Organizzatore\_Comitato

## 3.2 Definizione dei trigger

### 3.2.1 Check\_Programma

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

```
1 CREATE OR REPLACE FUNCTION check_programma()
2 RETURNS TRIGGER AS $$
3 DECLARE
4     inizio_evento TIMESTAMP;
5     fine_evento TIMESTAMP;
6     inizio_intervallo TIMESTAMP;
7     fine_intervallo TIMESTAMP;
8     inizio_intervento TIMESTAMP;
9     fine_intervento TIMESTAMP;
10    intervento_id INTEGER;
11    intervallo_id INTEGER;
12    evento_id INTEGER;
13    interventi_cur cursor FOR
14        SELECT id_intervento
15        FROM intervento
16        WHERE id_programma = new.id_programma;
17    intervalli_cur cursor FOR
18        SELECT id_intervallo
19        FROM intervallo
20        WHERE id_programma = new.id_programma;
21    eventi_cur cursor FOR
22        SELECT id_evento
23        FROM evento
24        WHERE id_programma = new.id_programma;
25 BEGIN
26     OPEN interventi_cur;
27     LOOP
28         FETCH interventi_cur INTO intervento_id;
29         EXIT WHEN NOT FOUND;
30         SELECT inizio,fine INTO inizio_intervento,fine_intervento
31         FROM intervento
32         WHERE id_intervento = intervento_id;
33         IF (new.inizio>=inizio_intervento AND new.fine<=fine_intervento) THEN
34             RAISE EXCEPTION 'Impossibile inserire intervento';
35         END IF;
36     END LOOP;
37     CLOSE interventi_cur;
38
39     OPEN intervalli_cur;
40     LOOP
41         FETCH intervalli_cur INTO intervallo_id;
42         EXIT WHEN NOT FOUND;
43         SELECT inizio,fine INTO inizio_intervallo,fine_intervallo
44         FROM intervallo
45         WHERE id_intervallo = intervallo_id;
46         IF (new.inizio>=inizio_intervallo AND new.fine<=fine_intervallo) THEN
47             RAISE EXCEPTION 'Impossibile inserire intervallo';
48         END IF;
49     END LOOP;
50     CLOSE intervalli_cur;
51
52     OPEN eventi_cur;
53     LOOP
```

```

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

```

Listato 3.22: *check\_programma\_entry*

### 3.2.2 Check\_Data\_Intervento, Check\_Data\_Intervallo, Check\_Data\_Evento

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

```

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

```

```

22 CREATE TRIGGER check_data_intervento
23 BEFORE INSERT OR UPDATE ON intervento
24 FOR EACH ROW
25 EXECUTE FUNCTION check_data();
26
27 CREATE TRIGGER check_data_intervallo
28 BEFORE INSERT OR UPDATE ON intervallo
29 FOR EACH ROW
30 EXECUTE FUNCTION check_data();

```

**Listato 3.23:** *check\_data\_intervento*

### 3.2.3 Create\_Programma\_Sessione

Il trigger Create\_Programma\_Sessione viene attivato subito dopo aver inserito una nuova sessione ed effettua l'inserimento di un programma vuoto associato alla sessione.

```

1 CREATE OR REPLACE FUNCTION create_programma_sessione() RETURNS TRIGGER AS $$
2 BEGIN
3 INSERT INTO programma(id_sessione) VALUES (new.id_sessione);
4 RETURN NEW;
5 END;
6 $$ LANGUAGE PLPGSQL;
7
8 CREATE TRIGGER create_programma_sessione
9 AFTER INSERT ON sessione
10 FOR EACH ROW
11 EXECUTE FUNCTION create_programma_sessione();

```

**Listato 3.24:** *create\_programma\_sessione*

### 3.2.4 Check\_Sala\_Sessione

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

```

1 CREATE OR REPLACE FUNCTION check_sala_sessione() RETURNS TRIGGER AS $$
2 DECLARE
3 sede INTEGER;
4 sala INTEGER;
5 BEGIN
6 SELECT id_sede INTO sede
7 FROM conferenza
8 WHERE id_conferenza = new.id_conferenza;
9
10 SELECT id_sala INTO sala
11 FROM sala
12 WHERE id_sala = new.id_sala;
13
14 IF sala IS NULL THEN
15 Return new;
16 END IF;
17
18 IF sala NOT IN (
19 SELECT id_sala
20 FROM sala
21 WHERE id_sede = sede

```



```

22 ) THEN
23 RAISE EXCEPTION 'La sala selezionata non appartiene alla sede della conferenza'
24 ;
25 END IF;
26 RETURN NEW;
27 END;
28 $$ LANGUAGE PLPGSQL;
29
30
31 CREATE TRIGGER check_sala_sessione
32 BEFORE INSERT OR UPDATE ON sessione
33 FOR EACH ROW
34 EXECUTE FUNCTION check_sala_sessione();

```

### 3.2.5 Check\_Data\_Session

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() RETURNS TRIGGER AS $$
2 DECLARE
3 inizio_conferenza TIMESTAMP;
4 fine_conferenza TIMESTAMP;
5 BEGIN
6 SELECT inizio, fine INTO inizio_conferenza, fine_conferenza
7 FROM conferenza
8 WHERE id_conferenza = new.id_conferenza;
9
10 IF (new.inizio < inizio_conferenza OR new.fine > fine_conferenza) THEN
11 RAISE EXCEPTION 'La sessione non e'' compresa nella conferenza';
12 END IF;
13 RETURN NEW;
14 END;
15 $$ LANGUAGE PLPGSQL;
16
17 CREATE TRIGGER check_data_sessione
18 BEFORE INSERT OR UPDATE ON sessione
19 FOR EACH ROW
20 EXECUTE FUNCTION check_data_sessione();

```

Listato 3.25: *check\_data\_sessione*

### 3.2.6 Check\_Coordinatore\_Session

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() RETURNS TRIGGER AS $$
2 DECLARE
3 id_comitato_scientifico_conferenza INTEGER;
4 BEGIN
5
6 SELECT comitato_s INTO id_comitato_scientifico_conferenza
7 FROM conferenza c
8 WHERE c.id_conferenza = new.id_conferenza;
9
10 IF (new.id_coordinatore is NOT NULL) THEN

```

```

11 IF (id_comitato_scientifico_conferenza NOT IN (SELECT id_comitato FROM
    organizzatore_comitato WHERE id_organizzatore = new.id_coordinatore)) THEN
12 RAISE EXCEPTION 'Il coordinatore della sessione deve appartenere al comitato
    scientifico della conferenza';
13 END IF;
14 END IF;
15 RETURN NEW;
16 END;
17 $$ LANGUAGE PLPGSQL;
18
19 CREATE TRIGGER check_coordinatore_sessione
20 BEFORE INSERT OR UPDATE ON sessione
21 FOR EACH ROW
22 EXECUTE FUNCTION check_coordinatore_sessione();

```

**Listato 3.26:** *check\_coordinatore\_sessione*

### 3.2.7 Create\_Comitati\_Conferenza

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

```

1 CREATE OR REPLACE FUNCTION create_comitati_conferenza() RETURNS TRIGGER AS $$
2 DECLARE
3 id_comitatoscientifico INTEGER;
4 id_comitatolocale INTEGER;
5 BEGIN
6 INSERT INTO comitato(tipologia) VALUES ('scientifico') returning id_comitato
    INTO id_comitatoscientifico;
7 INSERT INTO comitato(tipologia) VALUES ('locale') returning id_comitato INTO
    id_comitatolocale;
8 UPDATE conferenza SET comitato_s = id_comitatoscientifico, comitato_l =
    id_comitatolocale WHERE id_conferenza = new.id_conferenza;
9 RETURN NEW;
10 END;
11 $$ LANGUAGE PLPGSQL;
12
13 CREATE TRIGGER create_comitati_conferenza
14 AFTER INSERT ON conferenza
15 FOR EACH ROW
16 EXECUTE FUNCTION create_comitati_conferenza();

```

**Listato 3.27:** *create\_comitati\_conferenza*

### 3.2.8 Check\_Comitati\_Conferenza

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

```

1 CREATE OR REPLACE FUNCTION check_comitati_conferenza() RETURNS TRIGGER AS $$
2 DECLARE
3 id_comitato_scientifico INTEGER;
4 id_comitato_locale INTEGER;
5 BEGIN
6 SELECT id_comitato INTO id_comitato_scientifico
7 FROM comitato
8 WHERE id_comitato = new.comitato_s;

```

```

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

```

Listato 3.28: *check\_comitati\_conferenza*

### 3.2.9 Check\_Sala\_Sessione\_Unica

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

```

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

```

```

22 $$ LANGUAGE PLPGSQL;
23
24 CREATE TRIGGER check_sala_sessione_unica
25 BEFORE INSERT OR UPDATE ON sessione
26 FOR EACH ROW
27 EXECUTE FUNCTION check_sala_sessione_unica();

```

**Listato 3.29:** *Check\_sala\_sessione\_unica*

### 3.2.10 Check\_Organizzatore\_Comitato

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

```

1 CREATE OR REPLACE FUNCTION check_organizzatore_comitato() RETURNS TRIGGER AS $$
2 DECLARE
3 ente_id INTEGER;
4 BEGIN
5
6 SELECT id_ente INTO ente_id
7 FROM organizzatore o
8 WHERE o.id_organizzatore = new.id_organizzatore;
9
10 IF ente_id IS NULL THEN
11 RAISE EXCEPTION 'L''organizzatore non esiste';
12 END IF;
13
14 IF ente_id NOT IN (
15 SELECT id_ente
16 FROM ente_conferenza
17 WHERE id_conferenza IN (
18 SELECT id_conferenza
19 FROM conferenza
20 WHERE NEW.id_comitato IN (id_comitato_scientifico, id_comitato_locale)
21 )
22 ) THEN
23 RAISE EXCEPTION 'L''organizzatore deve appartenere ad un ente che ha
    organizzato la conferenza';
24 END IF;
25 RETURN NEW;
26 END;
27 $$ LANGUAGE PLPGSQL;
28
29 CREATE TRIGGER check_organizzatore_comitato
30 BEFORE INSERT OR UPDATE ON organizzatore_comitato
31 FOR EACH ROW
32 EXECUTE FUNCTION check_organizzatore_comitato();

```

**Listato 3.30:** *Check\_organizzatori\_comitato*

### 3.2.11 Delete\_Sessioni\_Conferenza

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

```

1 CREATE OR REPLACE FUNCTION delete_sessioni_conferenza() RETURNS TRIGGER AS $$
2 DECLARE
3 sessioni_cur cursor FOR
4 SELECT id_sessione
5 FROM sessione

```

```

6 WHERE id_conferenza = old.id_conferenza;
7 sessione_id INTEGER;
8 BEGIN
9 OPEN sessioni_cur;
10 LOOP
11 FETCH sessioni_cur INTO sessione_id;
12 EXIT WHEN NOT FOUND;
13 IF (SELECT inizio FROM sessione WHERE id_sessione = sessione_id) < new.inizio
14 OR (SELECT fine FROM sessione WHERE id_sessione = sessione_id) > new.fine THEN
15 delete FROM sessione WHERE id_sessione = sessione_id;
16 END IF;
17 END LOOP;
18 CLOSE sessioni_cur;
19 RETURN NEW;
20 END;
21 $$ LANGUAGE PLPGSQL;
22
23 CREATE TRIGGER delete_sessioni_conferenza
24 BEFORE UPDATE ON conferenza
25 FOR EACH ROW
26 EXECUTE FUNCTION delete_sessioni_conferenza();

```

Listato 3.31: *delete\_sessioni\_conferenza*

### 3.2.12 Check\_Capienza

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

```

1 CREATE OR REPLACE FUNCTION check_capienza_sala() RETURNS TRIGGER AS $$
2 DECLARE
3 capienza_s INTEGER;
4 partecipanti INTEGER;
5 BEGIN
6 SELECT capienza INTO capienza_s
7 FROM sala
8 WHERE id_sala = new.id_sala;
9
10 SELECT count(*) INTO partecipanti
11 FROM partecipazione
12 WHERE id_sessione = new.id_sessione;
13
14 IF (partecipanti >= capienza_s) THEN
15 RAISE EXCEPTION 'La capienza della sala e'' stata raggiunta';
16 END IF;
17 RETURN NEW;
18 END;
19 $$ LANGUAGE PLPGSQL;
20
21 CREATE TRIGGER check_capienza_sala
22 before insert ON partecipazione
23 FOR EACH ROW
24 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:

```
1 CREATE OR REPLACE FUNCTION show_conference_by_date(dataI DATE, dataF DATE)
2 RETURNS SETOF conferenza AS $$
3 BEGIN
4 RETURN QUERY
5 SELECT * FROM conferenza
6 WHERE inizio >= dataI AND fine <= dataF;
7 END;
8 $$ 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 INTEGER)
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 show_comitato_scientifico(conferenza INTEGER)
2 RETURNS SETOF organizzatore AS $$
3 BEGIN
4 RETURN QUERY
5 -- Select dei dettagli dell'organizzatore
6 SELECT * FROM organizzatore
7 WHERE id_organizzatore IN (
8 -- Select degli id degli organizzatori appartenenti al comitato scientifico
9 SELECT id_organizzatore FROM organizzatore_comitato
10 WHERE id_comitato = (
11 -- Select dell'id del comitato scientifico della conferenza
12 SELECT id_comitato_scientifico FROM conferenza
13 WHERE id_conferenza = conferenza
14 )
15 );
16 END;
17 $$ 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:

```

1 CREATE OR REPLACE FUNCTION show_comitato_locale(conferenza INTEGER)
2 RETURNS SETOF organizzatore AS $$
3 BEGIN
4 RETURN QUERY
5 -- Select dei dettagli dell'organizzatore
6 SELECT * FROM organizzatore
7 WHERE id_organizzatore IN (
8 -- Select degli id degli organizzatori appartenenti al comitato locale
9 SELECT id_organizzatore FROM organizzatore_comitato
10 WHERE id_comitato = (
11 -- Select dell'id del comitato locale della conferenza
12 SELECT id_comitato_locale FROM conferenza
13 WHERE id_conferenza = conferenza
14 )
15 );
16 END;
17 $$ 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.

```

1 CREATE OR REPLACE FUNCTION show_partecipanti(conferenza INTEGER)
2 RETURNS SETOF partecipante AS $$
3 BEGIN
4 RETURN QUERY
5 -- Select dei dettagli del partecipante
6 SELECT * FROM partecipante
7 WHERE id_partecipante IN (
8 -- Select degli id dei partecipanti
9 SELECT id_partecipante FROM partecipazione
10 WHERE id_sessione IN (
11 -- Select degli id delle sessioni della conferenza
12 SELECT id_sessione FROM sessione
13 WHERE id_conferenza = conferenza
14 )
15 );
16 END;
17 $$ 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 INTEGER)
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 show_interventi_sessione(sessione INTEGER)
2 RETURNS TABLE
3 (
4 titolo text,
5 inizio TIMESTAMP,
6 fine TIMESTAMP,
7 abstract text,
8 speaker text
9 ) AS $$
10 DECLARE
11 programma INTEGER;
12 BEGIN
13 SELECT id_programma INTO programma
14 FROM programma
15 WHERE id_sessione = sessione;
16
17 SELECT titolo,inizio,fine,abstract, s.nome || ' ' || s.cognome AS speaker
18 FROM intervento i JOIN speaker s ON i.id_speaker = s.id_speaker
19 WHERE i.id_programma = programma
20 order by inizio;
21 END;
22 $$ 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:

```
1 CREATE OR REPLACE FUNCTION show_intervalli_sessione(sessione INTEGER)
2 RETURNS TABLE
3 (
4 id_intervallo INTEGER,
5 tipologia intervallo_st,
6 inizio TIMESTAMP,
7 fine TIMESTAMP
8 )
9 AS $$
10 DECLARE
11 programma INTEGER;
12 BEGIN
13 SELECT id_programma INTO programma
14 FROM programma
15 WHERE id_sessione = sessione;
16
17 SELECT id_intervallo,tipologia,inizio,fine
18 FROM intervallo i
19 WHERE id_programma = programma
20 order by inizio;
21 END;
22 $$ 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:



```

1 CREATE OR REPLACE FUNCTION show_eventi_sociali_sessione(sessione INTEGER)
2 RETURNS TABLE
3 (
4 id_evento INTEGER,
5 tipologia text,
6 inizio TIMESTAMP,
7 fine TIMESTAMP)
8 AS $$
9 DECLARE
10 programma INTEGER;
11 BEGIN
12 SELECT id_programma INTO programma
13 FROM programma
14 WHERE id_sessione = sessione;
15
16 SELECT id_evento,tipologia,inizio,fine
17 FROM evento
18 WHERE id_programma = programma
19 order by inizio;
20 END;
21 $$ 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:

```

1 CREATE OR REPLACE FUNCTION show_keynote_sessione(sessione INTEGER)
2 RETURNS TABLE(
3 id_speaker INTEGER,
4 nome text,
5 cognome text,
6 titolo text,
7 email text,
8 ente text)
9 AS $$
10 DECLARE
11 programma INTEGER;
12 BEGIN
13 SELECT id_programma INTO programma
14 FROM programma
15 WHERE id_sessione = sessione;
16
17 SELECT s.id_speaker,s.nome,s.cognome,s.titolo,s.email,e.nome
18 FROM speaker s JOIN ente e ON s.id_ente = e.id_ente
19 WHERE s.id_speaker = (
20 SELECT id_keynote
21 FROM programma
22 WHERE id_programma = programma
23 );
24 END;
25 $$ 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 programma IN ordine cronologico:

```

1 CREATE OR REPLACE FUNCTION show_programma(sessione INTEGER)

```

```

2 RETURNS TABLE (
3 id_entry INTEGER,
4 appuntamento text,
5 inizio TIMESTAMP,
6 fine TIMESTAMP,
7 descrizione text,
8 speaker text
9 )
10 AS $$
11 DECLARE
12 programma INTEGER;
13 BEGIN
14 SELECT id_programma INTO programma
15 FROM programma
16 WHERE id_sessione = sessione;
17
18 RETURN QUERY
19 SELECT *
20 FROM (
21 SELECT distinct i.id_intervento AS id_entry,
22 'intervento' AS appuntamento,
23 i.inizio,
24 i.fine,
25 i.abstract,
26 s.nome || ' ' || s.cognome AS speaker
27 FROM intervento i
28 JOIN speaker s ON i.id_speaker = s.id_speaker
29 WHERE i.id_programma = programma
30
31 UNION ALL
32
33 SELECT i2.id_intervallo AS id_entry,
34 'intervallo' AS appuntamento,
35 i2.inizio,
36 i2.fine,
37 tipologia::text AS descrizione,
38 NULL
39 FROM intervallo i2
40 WHERE i2.id_programma = programma
41
42 UNION ALL
43
44 SELECT e.id_evento AS id_entry,
45 'evento' AS appuntamento,
46 e.inizio,
47 e.fine,
48 e.tipologia::text AS descrizione,
49 NULL
50 FROM evento e
51 WHERE e.id_programma = programma
52 ) AS subquery
53 ORDER BY inizio;
54 END;
55 $$ LANGUAGE plpgsql;

```

### 3.3.12 Add\_Intervento(text,text,text,INTEGER,interval)

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

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

```
1 create or replace procedure add_intervento
2 (titolo text,
3 abstract text,
4 speaker text,
5 sessione_id INTEGER,
6 durata interval)
7 AS $$
8 DECLARE
9 programma INTEGER;
10 fine_prev TIMESTAMP;
11 BEGIN
12 SELECT id_programma INTO programma
13 FROM programma
14 WHERE id_sessione = sessione_id;
15
16 SELECT max(fine) INTO fine_prev
17 FROM show_programma(sessione_id);
18
19 IF (fine_prev is null) THEN
20 SELECT inizio INTO fine_prev
21 FROM sessione
22 WHERE id_sessione = sessione_id;
23 END IF;
24
25 INSERT INTO intervento(titolo,abstract,id_speaker,id_programma,inizio,fine)
26 VALUES (titolo,abstract,speaker,programma,fine_prev,fine_prev+durata);
27 RAISE NOTICE 'Inserimento completato';
28 EXCEPTION
29 WHEN OTHERS THEN
30 RAISE NOTICE '%', SQLERRM;
31 END;
32 $$ LANGUAGE PLPGSQL;
```

### 3.3.13 Add\_Intervallo(text,INTEGER,interval)

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

```
1 create or replace procedure
2 add_intervallo(tipologia text , sessione_id INTEGER, durata interval)
3 AS $$
4 DECLARE
5 programma INTEGER;
6 fine_prev TIMESTAMP;
7 BEGIN
8 SELECT id_programma INTO programma
9 FROM programma
10 WHERE id_sessione = sessione_id;
11
12 SELECT max(fine) INTO fine_prev
13 FROM show_programma(sessione_id);
14
15 IF (fine_prev is null) THEN
16 SELECT inizio INTO fine_prev
17 FROM sessione
```

```

18 WHERE id_sessione = sessione_id;
19 END IF;
20
21 INSERT INTO intervallo(tipologia,id_programma,inizio,fine)
22 VALUES (tipologia::intervallo_st, programma, fine_prev, fine_prev+durata);
23 RAISE NOTICE 'Inserimento completato';
24 EXCEPTION
25 WHEN OTHERS THEN
26 RAISE NOTICE '%', SQLERRM;
27 END;
28 $$
29 LANGUAGE PLPGSQL;

```

### 3.3.14 Add\_Evento(text,INTEGER,interval)

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

```

1 create or replace procedure
2 add_evento
3 (tipologia text,
4 sessione_id INTEGER,
5 durata interval)
6 AS $$
7 DECLARE
8 programma_id INTEGER;
9 fine_prev TIMESTAMP;
10 BEGIN
11 -- Recupera l'id del programma della sessione
12 SELECT id_programma INTO programma_id
13 FROM programma
14 WHERE id_sessione = sessione_id;
15
16 -- Recupera l'id dell'ultimo punto IN programma, la tipologia e la fine
17 SELECT max(fine) INTO fine_prev
18 FROM show_programma(sessione_id);
19
20 IF (fine_prev is null) THEN
21 SELECT inizio INTO fine_prev
22 FROM sessione
23 WHERE id_sessione = sessione_id;
24 END IF;
25
26 INSERT INTO evento(tipologia, id_programma, inizio, fine)
27 VALUES (tipologia, programma_id, fine_prev, fine_prev+durata);
28 RAISE NOTICE 'Inserimento completato';
29 EXCEPTION
30 WHEN OTHERS THEN
31 RAISE NOTICE '%', SQLERRM;
32 END;
33 $$
34 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.

```

1 CREATE OR REPLACE FUNCTION add_conferenza_details
2 (nome text, inizio TIMESTAMP, fine TIMESTAMP, sede INTEGER, abstract text,
   utente INTEGER)
3 RETURNS INTEGER AS $$
4 DECLARE
5 id INTEGER;
6 BEGIN
7 INSERT INTO conferenza(titolo, inizio, fine, id_sede, descrizione, id_utente)
8 VALUES (nome, inizio, fine, sede, abstract, utente)
9 RETURNING id_conferenza INTO id;
10 RAISE NOTICE 'Inserimento completato';
11 RETURN id;
12 EXCEPTION
13 WHEN OTHERS THEN
14 RAISE NOTICE 'Errore nell''inserimento di una conferenza: %', SQLERRM;
15 RETURN 0;
16 END;
17 $$ 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.

```

1 create or replace procedure add_ente(ente INTEGER, conferenza INTEGER)
2 AS $$
3 BEGIN
4 INSERT INTO ente_conferenza(id_ente, id_conferenza)
5 VALUES (ente, conferenza);
6 RAISE NOTICE 'Inserimento completato';
7 EXCEPTION
8 WHEN OTHERS THEN
9 RAISE NOTICE '%', SQLERRM;
10 END;
11 $$ LANGUAGE PLPGSQL;

```

### 3.3.17 Add\_Sponsorizzazione(integer, numeric, char(3), integer)

La procedura Add\_Sponsorizzazione inserisce una nuova sponsorizzazione per la conferenza:

```

1 create or replace procedure add_sponsorizzazione(sponsor INTEGER, contributo
   numeric(1000,2), valuta char(3), conferenza INTEGER)
2 AS $$
3 BEGIN
4 INSERT INTO sponsorizzazione(id_sponsor, contributo, valuta, id_conferenza)
5 VALUES (sponsor, contributo, valuta, conferenza);
6 RAISE NOTICE 'Inserimento completato';
7 EXCEPTION
8 WHEN OTHERS THEN
9 RAISE NOTICE '%', SQLERRM;
10 END;
11 $$ LANGUAGE PLPGSQL;

```

### 3.3.18 Add\_Sessione(text, TIMESTAMP, TIMESTAMP, integer, integer)

La procedura Add\_Sessione aggiunge una nuova sessione per la conferenza:

```

1 create or replace procedure add_sessione(titolo text, inizio TIMESTAMP, fine
   TIMESTAMP, sala INTEGER, conferenza INTEGER)

```

```

2 AS $$
3 BEGIN
4 INSERT INTO sessione(titolo,inizio,fine,id_sala,id_conferenza)
5 VALUES (titolo,inizio,fine,sala,conferenza);
6 RAISE NOTICE 'Inserimento completato';
7 EXCEPTION
8 WHEN OTHERS THEN
9 RAISE NOTICE '%', SQLERRM;
10 END;
11 $$ LANGUAGE PLPGSQL;

```

### 3.3.19 Add\_Partecipante(integer, integer)

La procedura Add\_Partecipante inserisce un nuovo partecipante alla sessione:

```

1 create or replace procedure add_partecipante(partecipante INTEGER, sessione
    INTEGER)
2 AS $$
3 BEGIN
4 INSERT INTO partecipante_sessione(id_partecipante,id_sessione)
5 VALUES (partecipante,sessione);
6 RAISE NOTICE 'Inserimento completato';
7 EXCEPTION
8 WHEN OTHERS THEN
9 RAISE NOTICE '%', SQLERRM;
10 END;
11 $$ LANGUAGE PLPGSQL;

```

### 3.3.20 Add\_Enti(integer,text)

```

1 CREATE OR REPLACE PROCEDURE add_enti(conferenza INTEGER, sigle text)
2 AS $$
3 DECLARE
4 sigla_ente text;
5 ente_id INTEGER;
6 BEGIN
7 FOR sigla_ente IN SELECT unnest(string_to_array(sigle, ',')) LOOP
8 -- Cerca l'id dell'ente corrispondente alla sigla
9 SELECT id_ente INTO ente_id FROM ente WHERE sigla = sigla_ente;
10
11 -- Inserisci la tupla (id_ente, conferenza) nella tabella ente_conferenza
12 INSERT INTO ente_conferenza(id_ente, id_conferenza) VALUES (ente_id, conferenza
    );
13 END LOOP;
14 RAISE NOTICE 'Inserimento completato';
15 EXCEPTION
16 WHEN OTHERS THEN
17 RAISE EXCEPTION 'Errore durante l''inserimento delle tuple nella tabella
    ente_conferenza: %', SQLERRM;
18 END;
19 $$ LANGUAGE plpgsql;

```

### 3.3.21 Add\_Conferenza(text,TIMESTAMP,TIMESTAMP,integer, text, text)

```

1 create or replace procedure add_conferenza(nome text, inizio TIMESTAMP, fine
    TIMESTAMP, sede INTEGER, descrizione text, sigle text)
2 AS $$
3 DECLARE

```

```

4 id_conferenza INTEGER;
5 BEGIN
6 id_conferenza := add_conferenza_details(nome,inizio,fine,sede,descrizione);
7 call add_enti(id_conferenza,sigle);
8 EXCEPTION
9 WHEN OTHERS THEN
10 RAISE NOTICE '%', SQLERRM;
11 END;
12 $$ LANGUAGE PLPGSQL;

```

### 3.3.22 Slitta\_Conferenza(interval)

```

1  create or replace procedure
2 slitta_conferenza(conferenza_id INTEGER, durata interval)
3 AS $$
4 DECLARE
5 sessione_id INTEGER;
6 intervento_id INTEGER;
7 evento_id INTEGER;
8 intervallo_id INTEGER;
9 sessioni cursor FOR
10 SELECT id_sessione
11 FROM sessione
12 WHERE id_conferenza = conferenza_id;
13
14 interventi cursor FOR
15 SELECT id_intervento
16 FROM intervento i JOIN programma p
17 ON i.id_programma = p.id_programma
18 WHERE p.id_sessione IN
19 (SELECT id_sessione
20 FROM sessione
21 WHERE id_conferenza = conferenza_id);
22
23 intervalli cursor FOR
24 SELECT id_intervallo
25 FROM intervallo i JOIN programma p
26 ON i.id_programma = p.id_programma
27 WHERE p.id_sessione IN
28 (SELECT id_sessione
29 FROM sessione
30 WHERE id_conferenza = conferenza_id);
31
32 eventi cursor FOR
33 SELECT id_evento
34 FROM evento e JOIN programma p
35 ON e.id_programma = p.id_programma
36 WHERE p.id_sessione IN
37 (SELECT id_sessione
38 FROM sessione
39 WHERE id_conferenza = conferenza_id);
40 BEGIN
41 ALTER TABLE conferenza DISABLE TRIGGER ALL;
42 ALTER TABLE sessione DISABLE TRIGGER ALL;
43 ALTER TABLE intervento DISABLE TRIGGER ALL;
44 ALTER TABLE intervallo DISABLE TRIGGER ALL;
45 ALTER TABLE evento DISABLE TRIGGER ALL;
46 ALTER TABLE programma DISABLE TRIGGER ALL;
47 UPDATE conferenza
48 SET inizio = inizio + durata, fine = fine + durata

```

```

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

```

### 3.3.23 Show\_members()



```

1 CREATE OR REPLACE FUNCTION show_members(conferenza INTEGER)
2 RETURNS TABLE
3 (
4 id INTEGER,
5 nome text,
6 cognome text,
7 email text,
8 titolo titolo_st,
9 sigla varchar(7)
10 )
11 AS $$
12 BEGIN
13 RETURN QUERY
14 SELECT o.id_organizzatore, o.nome, o.cognome, o.email,o.titolo, e.sigla
15 FROM organizzatore o JOIN ente_conferenza ec natural JOIN ente e
16 ON o.id_ente = ec.id_ente
17 WHERE ec.id_conferenza = conferenza
18 GROUP by e.sigla;
19 END;
20 $$ LANGUAGE PLPGSQL;

```

### 3.3.24 Show\_percentage\_interventi(INTEGER,INTEGER)

```

1 CREATE OR REPLACE FUNCTION show_percentage_interventi(mese INTEGER, anno
2 INTEGER)
3 RETURNS TABLE
4 (
5 ente text,
6 percentuale text
7 ) AS $$
8 DECLARE
9 totale INTEGER;
10 BEGIN
11 SELECT count(*) INTO totale
12 FROM intervento
13 WHERE date_part('month',inizio) = mese AND date_part('year',inizio) = anno;
14 RETURN QUERY
15 SELECT e.nome, (count(*)*100/totale)::text || '%',
16 FROM intervento i JOIN speaker s
17 ON i.id_speaker = s.id_speaker JOIN ente e
18 ON s.id_ente = e.id_ente
19 WHERE date_part('month',inizio) = mese AND date_part('year',inizio) = anno
20 group by e.nome;
21 END;
22 $$ LANGUAGE PLPGSQL;

```

### 3.3.25 Show\_percentage(INTEGER)

```

1 CREATE OR REPLACE FUNCTION show_percentage_interventi(anno INTEGER)
2 RETURNS TABLE
3 (
4 nome varchar(7),
5 percentuale text
6 ) AS $$
7 DECLARE
8 totale INTEGER;
9 BEGIN

```

```

10 SELECT count(*) INTO totale
11 FROM intervento
12 WHERE date_part('year',inizio) = anno;
13
14 RETURN QUERY
15 SELECT e.nome, (count(*)*100/totale)::text || '%'
16 FROM intervento i JOIN speaker s
17 ON i.id_speaker = s.id_speaker JOIN ente e
18 ON s.id_ente = e.id_ente
19 WHERE date_part('year',inizio) = anno
20 group by e.nome;
21 END;
22 $$ LANGUAGE PLPGSQL;

```

## 3.4 Definizione delle viste

### 3.4.1 SediView

```

1  create view SediView AS
2  SELECT s.nome AS Sede,
3  i.via || ', '
4  || i.civico
5  || ', '
6  || i.cap
7  || ', '
8  || i.city
9  || ' ('
10 || i.provincia
11 || '), '
12 || i.nazione AS Indirizzo
13 FROM sede s natural JOIN indirizzo i;

```

### 3.4.2 Conferenze\_Sede

```

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

```

### 3.4.3 Interventi\_Speaker

```

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

```

### 3.4.4 Partecipanti\_Sessione

```

1  create view partecipanti_sessioni AS
2  SELECT s.titolo AS Sessione,
3  count(p.id_partecipante) AS Numero_partecipanti
4  FROM sessione s, partecipazione p
5  WHERE s.id_sessione = p.id_sessione
6  group by s.titolo;

```

### 3.4.5 Partecipanti\_Conferenze

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

### 3.4.6 Sessioni

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

# Appendice A

## Dizionari

### A.1 Dizionario dei dati

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

*Continua nella prossima pagina*

Continua dalla pagina precedente

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

Continua nella prossima pagina

Continua dalla pagina precedente

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

Continua nella prossima pagina

Continua dalla pagina precedente

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

## A.2 Dizionario delle associazioni

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

Continua nella pagina successiva

Continua dalla pagina precedente

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

Continua nella pagina successiva



*Continua dalla pagina precedente*

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

### 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.