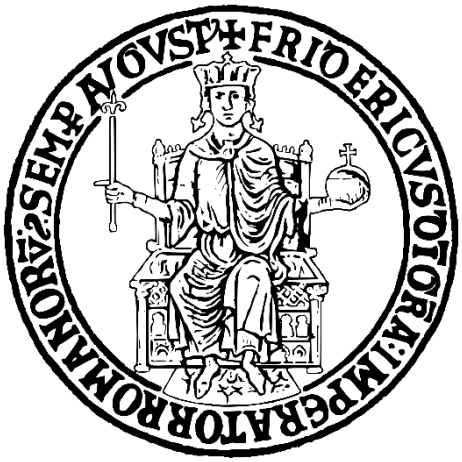
Università degli Studi di Napoli “Federico II”, CdL in Informatica



A.A 2020/21

Sistema di Planning Aziendale

Progettazione e sviluppo di una base di dati relazionale in PostgreSQL

di Fascia Fabio (N86003288) e Conte Salvatore (N86003295)

CODICE GRUPPO : 2 - CODICE TRACCIA : 2

Sommario

1. Introduzione

Il seguente elaborato ha lo scopo di documentare la progettazione e lo sviluppo di una base di dati relazionale del DBMS PostgreSQL, ad opera degli studenti Fascia Fabio e Conte Salvatore del CdL in Informatica presso l'Università degli Studi di Napoli "Federico II". Il database nasce come progetto a scopi valutativi per il corso di Basi di Dati, ed implementa un sistema di planning aziendale.

1.1. Descrizione del problema

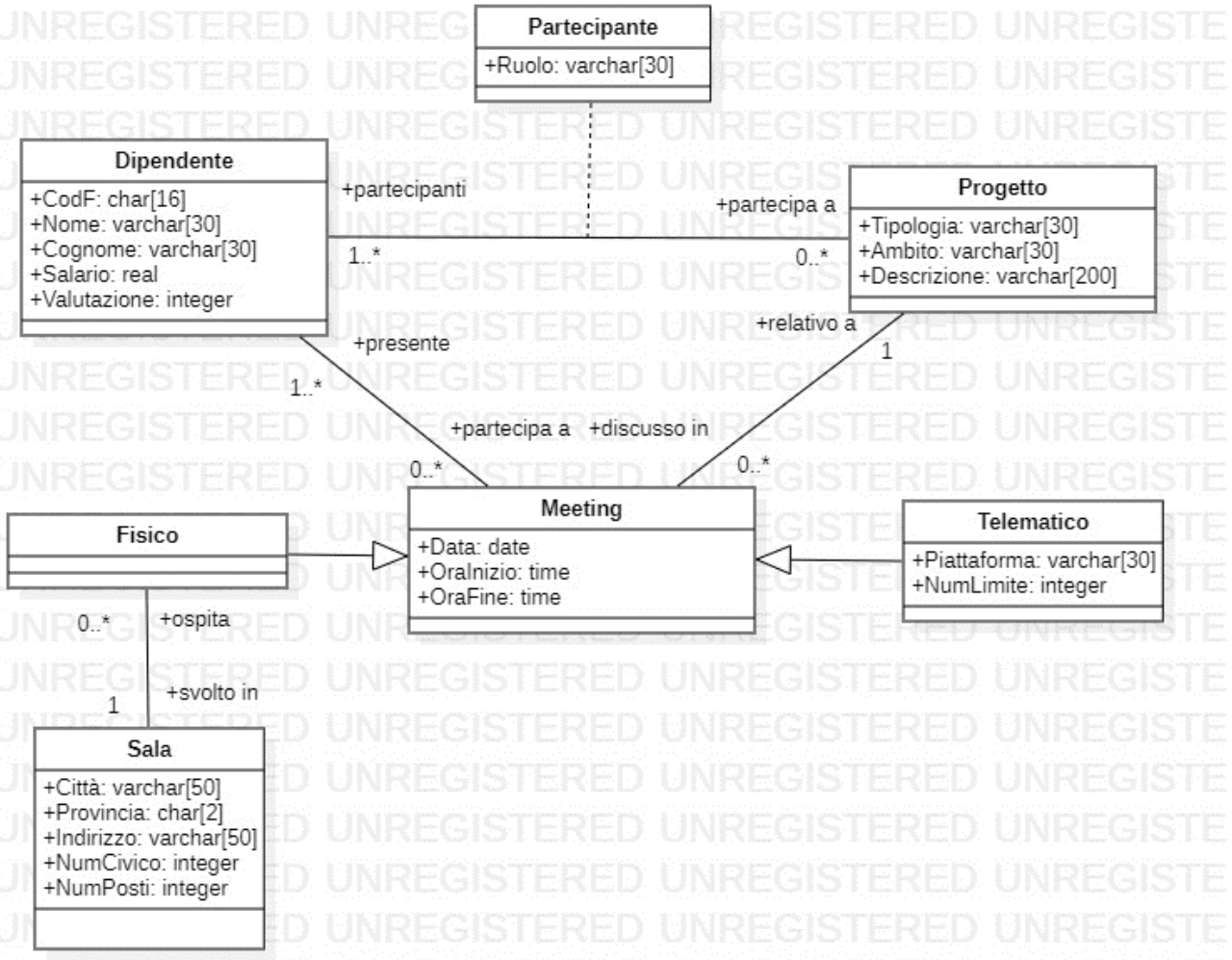
Verranno riportate progettazione e sviluppo di una base di dati relazionale, che implementi un sistema di planning aziendale che permetta l'organizzazione e la storicizzazione di progetti aziendali.

Il sistema tiene traccia dei partecipanti ai progetti, identificando ruoli per ognuno di essi. Ad ogni progetto sono associati una tipologia (ad es. "Ricerca di base", "Ricerca Industriale", "Ricerca Sperimentale" ecc.) ed uno o più ambiti (Economia, Medicina, Biologia…). Il sistema permette inoltre l'organizzazione di meeting fisicamente, in sale riunioni, o telematicamente, su una piattaforma di videoconferenza. Viene tenuta traccia delle partecipazioni a progetti e meeting ai fini della valutazione dei singoli partecipanti, gestita con un sistema a punti (ogni partecipazione vale 1 punto nella valutazione).

2. Progettazione Concettuale

In questo capitolo documentiamo la progettazione del database al suo livello di astrazione più alto. Partendo dall'analisi dei requisiti da soddisfare, si arriverà ad uno schema concettuale indipendente dalla struttura dei dati e dall'implementazione fisica degli stessi, rappresentato con un Class Diagram UML. Quest'ultimo evidenzierà le entità rilevanti nel problema, oltre alle relazioni che intercorrono tra esse e gli eventuali vincoli da imporre.

2.1. Class Diagram



2.2. Ristrutturazione del Class Diagram

Si procede alla ristrutturazione del Class Diagram, al fine di rendere quest'ultimo idoneo alla traduzione in schemi relazionali e di migliorarne l'efficienza. La ristrutturazione procederà secondo i seguenti punti:

- Analisi delle chiavi
- Analisi degli attributi derivati
- Analisi delle ridondanze
- Analisi degli attributi strutturati
- Analisi degli attributi a valore multiplo
- Analisi delle gerarchie di specializzazione

2.2.1. Analisi delle chiavi

Ai fini dell'efficienza nella rappresentazione delle varie entità, nello specifico Progetti, Meeting e Sale Riunioni, risulta conveniente l'introduzione di chiavi primarie "surrogate" ovvero rappresentate non da informazioni proprie della singola entità, ma da identificativi di tipo intero associati a ciascuna istanza delle stesse. Ciò permetterà di identificare in maniera computazionalmente meno dispendiosa le varie istanze delle suddette entità.

2.2.2. Analisi degli attributi derivati

Per ottimizzare ulteriormente l'utilizzo delle risorse di calcolo, analizziamo gli eventuali attributi derivati, ovvero calcolabili da altri attributi delle entità.

Il più evidente risulta essere in questo caso la Valutazione aziendale del singolo Dipendente, calcolabile a partire palle partecipazioni storicizzate di quest'ultimo a Progetti e Meeting. Ad una attenta analisi risulta essere conveniente la storicizzazione di tale valore come attributo del Dipendente stesso, ai fini di una più efficiente ricerca dei Dipendenti in funzione della loro Valutazione.

2.2.3. Analisi delle ridondanze

Analizziamo ora l'eventuale presenza di associazioni ridondanti tra le varie entità, in maniera tale da evitare incoerenze nella rappresentazione logica dei dati.

L'unico caso riscontrabile è dato dall'associazione tra Progetto e Dipendente: ogni progetto può avere infatti un qualunque numero di partecipanti, ma richiede sempre la presenza di uno e un solo Project Manager. Quest'ultimo è però a sua volta un partecipante al Progetto, motivo per cui risulta logicamente più corretto gestire l'associazione relativa al Project Manager, a questo punto evidentemente ridondante, con una serie di vincoli interrelazionali, nello specifico uno di unicità ed uno di totalità del PM.

2.2.4. Analisi degli attributi strutturati

Vanno ora analizzati e concettualmente corretti eventuali attributi strutturati presenti nelle entità. Questi infatti non sono logicamente rappresentabili all'interno di un DBMS, e vanno quindi eliminati e codificati in altro modo.

Fortunatamente, nella rappresentazione concettuale presentata non sono presenti attributi strutturati.

2.2.5. Analisi degli attributi a valore multiplo

Verifichiamo ora la presenza di eventuali attributi a valore multiplo, anch'essi non logicamente rappresentabili e quindi da eliminare nello schema concettuale ristrutturato.

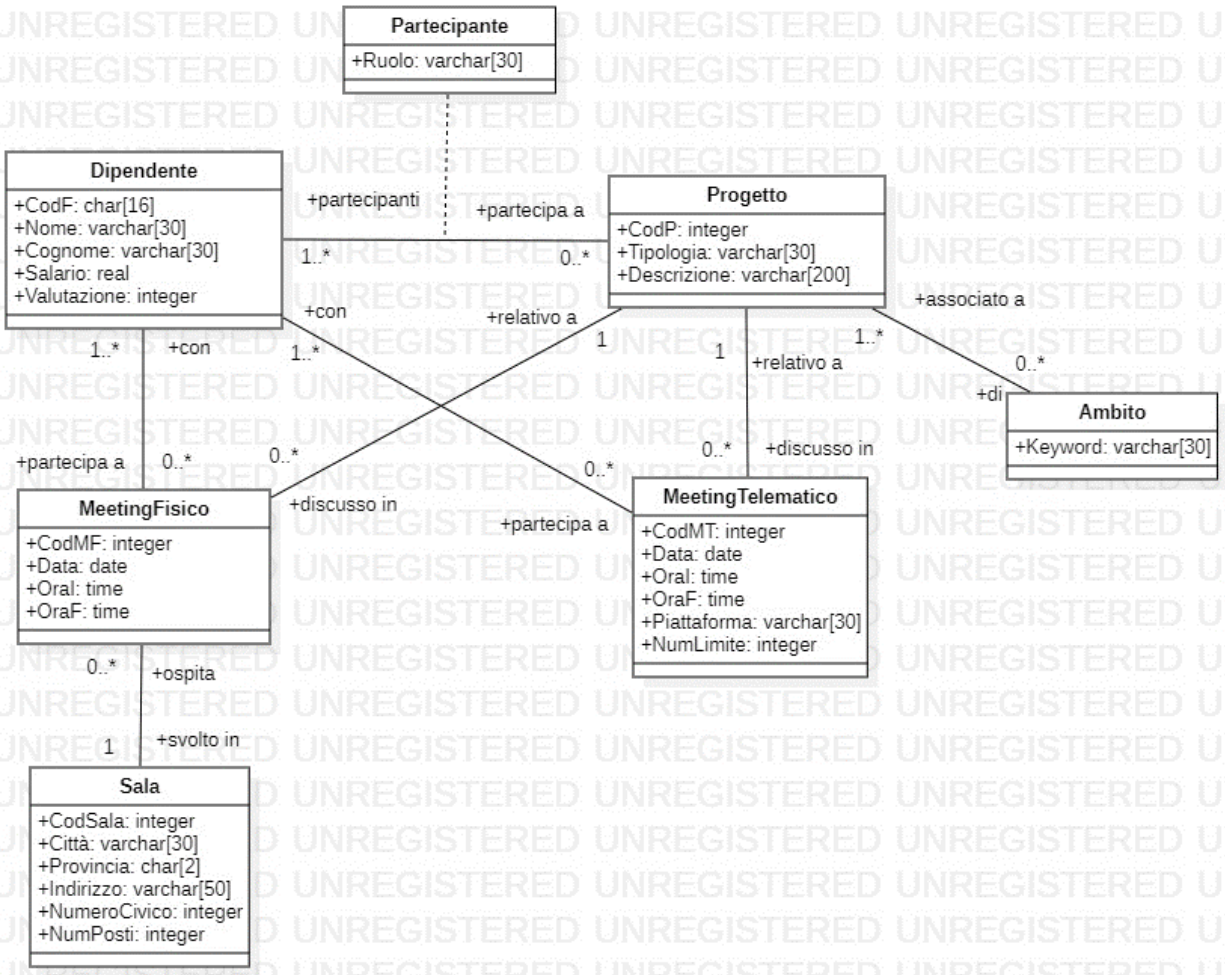
Un attributo a valore multiplo è l'Ambito di un progetto: un singolo progetto può infatti avere più di un ambito. Per correggere tale problema logico, rappresentiamo l'attributo tramite una nuova entità Ambito, associata uno a molti con i Progetti.

2.2.6. Analisi delle gerarchie di specializzazione

Infine andiamo a ristrutturare eventuali gerarchie di specializzazione, altro elemento non rappresentabile in un DBMS relazionale.

L'unica gerarchia di specializzazione nel Class Diagram presentato è quella riguardante i Meeting: il generico meeting si specializza infatti in Meeting Fisico oppure in Meeting Telematico. Tale specializzazione è Totale (ogni Meeting deve essere specializzato) e Disgiunta (un Meeting non può essere sia Fisico che Telematico), dunque risulta conveniente ai fini della rappresentazione logica ricontestualizzare tale specializzazione tramite un "appiattimento": la generalizzazione Meeting viene eliminata e cede i suoi attributi e le sue associazioni alle singole specializzazioni. Il risultato saranno quindi due entità, Meeting Fisico e Meeting Telematico, indipendenti l'una dall'altra.

2.3. Class Diagram Ristrutturato



2.4. Dizionario delle Classi

Classe	Descrizione	Attributi
Dipendente	Descrive ciascun dipendente dell'azienda fruitrice del sistema.	• **CodF** (char[16]) **:** Codice Fiscale del Dipendente. • **Nome** (string): Nome del Dipendente. • **Cognome** (string) : Cognome del Dipendente. • **Salario** (float) : Salario attuale del Dipendente. • **Valutazione** (int) : Valutazione aziendale del Dipendente, espressa come punteggio in numeri interi.
Progetto	Descrive i singoli progetti portati avanti dall'azienda.	• **CodP** (int) : Codice identificativo univoco del Progetto (chiave surrogata). • **Tipologia** (string) : Tipologia del progetto. • **Descrizione** (string, parziale) : Breve descrizione del progetto.
Ambito	Codifica dell'attributo a valore multiplo di Progetto: descrive i singoli ambiti di un progetto.	• **Keyword** (string) : Nome dell'ambito.
Meeting Fisico	Descrive un Meeting Fisico tenutosi per un Progetto.	• **CodMF** (int) : Codice identificativo univoco del Meeting Fisico (chiave surrogata). • **Data** (date) : Data in cui il Meeting si è tenuto.

		• **OraI** (time) : Ora di inizio del Meeting. • **OraF** (time) : Ora in cui il Meeting è terminato.
Meeting Telematico	Descrive un Meeting Telematico tenutosi per un Progetto.	• **CodMT** (int) : Codice identificativo univoco del Meeting Telematico (chiave surrogata). • **Data** (date) : Data in cui il Meeting si è tenuto. • **OraI** (time) : Ora di inizio del Meeting. • **OraF** (time) : Ora in cui il Meeting è terminato. • **Piattaforma** (string) : Piattaforma di videoconferenze su cui il Meeting si è svolto. • **NumLimite** (int, parziale) : Eventuale limite al numero di partecipanti al Meeting Telematico, dettato dalla Piattaforma oppure dagli organizzatori.
Sala	Descrive una sala riunioni in cui è possibile organizzare Meeting Fisici.	• **CodSala** (int) : Codice identificativo univoco della Sala riunioni (chiave surrogata). • **Città** (string) : Città in cui è presente la Sala riunioni. • **Provincia** (char[2]) : Provincia della Città specificata. • **Indirizzo** (string) : Indirizzo della Sala riunioni. • **NumeroCivico** (int) : Numero Civico della Sala riunioni. • **NumPosti** (int) : Numero dei posti disponibili nella Sala.

2.5. Dizionario delle Associazioni

Nome	Descrizione	Classi coinvolte
Partecipante	Esprime la partecipazione di un Dipendente ad un determinato Progetto.	**Dipendente** [1..*] ruolo partecipante : indica un Dipendente che partecipa ad un certo Progetto. **Progetto** [1..*] ruolo partecipa a : indica il progetto a cui un certo dipendente partecipa. **Ruolo** classe di associazione : indica il ruolo che un determinato Dipendente svolge in un determinato Progetto.
Partecipante Meeting Fisico	Esprime la partecipazione di un Dipendente ad un Meeting Fisico.	**Dipendente** [1..*] ruolo partecipante : indica un Dipendente che partecipa ad un certo Meeting Fisico. **Meeting Fisico** [0..*] ruolo partecipa a : indica un Meeting Fisico a cui partecipa un determinato Dipendente.
Partecipante Meeting Telematico	Esprime la partecipazione di un Dipendente ad un Meeting Telematico.	**Dipendente** [1..*] ruolo partecipante : indica un Dipendente che partecipa ad un certo Meeting Telematico. **Meeting Telematico** [0..*] ruolo partecipa a : indica un Meeting Telematico a cui partecipa un determinato Dipendente.
Argomento Meeting Fisico	Esprime la relazione tra un Meeting Fisico ed il Progetto di cui si è discusso in esso.	**Progetto** [1] ruolo relativo a : indica il Progetto per cui si è tenuto un determinato Meeting Fisico. **Meeting Fisico** [0..*] ruolo discusso in : indica un Meeting Fisico in cui si è discusso di un determinato Progetto.

Argomento Meeting Telematico	Esprime la relazione tra un Meeting Telematico ed il Progetto di cui si è discusso in esso.	**Progetto** [1] ruolo relativo a : indica il Progetto per cui si è tenuto un determinato Meeting Telematico. **Meeting Telematico** [0..*] ruolo discusso in : indica un Meeting Telematico in cui si è discusso di un determinato Progetto.
Ambiti Progetto	Esprime la relazione tra un Progetto e i suoi Ambiti	**Progetto** [1] ruolo di : indica il Progetto a cui associamo un insieme di Ambiti. **Ambito** [0..*] ruolo relativo a : indica un Ambito associato ad un determinato Progetto.
Sala Prenotata	Esprime la relazione tra un Meeting Fisico e la Sala riunioni in cui esso si è svolto.	**Meeting Fisico** [0..*] ruolo ospita : indica un Meeting Fisico che si è svolto in una determinata Sala riunioni. **Sala** [1] ruolo svolto in : indica la Sala riunioni in cui un determinato Meeting Fisico si è svolto.

2.6. Dizionario dei Vincoli

Nome	Descrizione
Codice Fiscale Legale	Il Codice Fiscale di un Dipendente deve rispettare la sua formattazione standard, ovvero deve contenere una sequenza di: 6 caratteri, 2 interi, 1 char, 2 int, 1 char, 3 int, 1 char.
Nome Legale	Il Nome di un Dipendente deve essere composto unicamente di caratteri.
Cognome Legale	Il Cognome di un Dipendente deve essere composto unicamente di caratteri.
Consistenza Valutazione	La valutazione aziendale di un Dipendente deve essere sempre uguale alla somma dei Progetti e dei Meeting a cui ha partecipato.
Consistenza Partecipante	Ciascun Partecipante ad un Progetto può avere al più 1 ruolo, ovvero lo stesso Dipendente non può partecipare ad uno stesso Progetto con più ruoli differenti.
Unicità Project Manager	Ad un Progetto può partecipare un solo Project Manager.
Totalità Project Manager	Ad un Progetto deve sempre partecipare un Project Manager.
Consistenza Meeting Fisico	Due Meeting Fisici distinti non si possono tenere nella stessa Sala, alla stessa ora dello stesso giorno.
Consistenza Partecipazione Meeting	Un Dipendente non può partecipare a più Meeting (Fisici/Telematici) contemporaneamente.
Capienza Sala Riunioni	Il numero di Partecipanti ad un Meeting Fisico non può eccedere il limite di posti della Sala in cui esso si svolge.
Limite Piattaforma Videoconferenze	Il numero di Partecipanti ad un Meeting Telematico non può eccedere il limite di partecipanti al Meeting (se esiste).

3. Progettazione Logica

In questo capitolo tratteremo la seconda fase della progettazione, scendendo ad un livello di astrazione più basso rispetto al precedente.

Lo schema concettuale verrà tradotto, anche grazie alla predisposizione conseguente la ristrutturazione, in uno schema logico, questa volta dipendente dalla struttura dei dati prescelta, nello specifico quella relazionale pura.

3.1. Schema Logico

Di seguito è riportato lo schema logico della base di dati. Al suo interno, le chiavi primarie sono indicate con una sottolineatura singola mentre le chiavi esterne con una sottolineatura doppia.

- Dipendente (CodF, Nome, Cognome, Salario, Valutazione)
- Progetto (CodP, Tipologia)
- Ambito (CodP, Keyword)
  CodP → Progetto.CodP
- Partecipante (CodF, CodP, Ruolo)
  CodF →Dipendente.CodF ; CodP → Progetto.CodP
- Sala (CodSala, Città, Provincia, Indirizzo, NumCivico, NumPosti)
- MeetingF (CodMF, Data, OraI, OraF, CodSala)
  CodSala → Sala.CodSala
- MeetingT (CodMT, Data, OraI, OraF, Piattaforma, NumLimite)
- PartecipaMF (CodF, CodMF)
  CodF → Dipendente.CodF ; CodMF → MeetingF.CodMF
- PartecipaMT (CodF, CodMT)
  CodF → Dipendente.CodF ; CodMT → MeetingT.CodMT

4. Progettazione Fisica

In questo capitolo verrà riportata l'implementazione dello schema logico sopra descritto nel DBMS PostgreSQL.

4.1. Definizione Tabelle

Di seguito sono riportate le definizioni delle tabelle, dei loro vincoli intrarelazionali e di eventuali semplici strutture per la loro gestione.

4.1.1. Definizione della Tabella DIPENDENTE

```
1. --Definizione tabella
2. CREATE TABLE DIPENDENTE
3. (
4.     CodF CHAR(16) PRIMARY KEY DEFAULT 'AAAAAA00A00A000A'
5.   CHECK (CodF ~* '^[A-Za-z]{6}[0-9]{2}[A-Za-z][0-9]{2}[A-Za-z][0-9]{3}[A-
   Za-z]$'),
6.     Nome VARCHAR(30) NOT NULL
7.   CHECK (Nome ~* '^[A-Za-z ]+$'),
8.     Cognome VARCHAR(30) NOT NULL
9.   CHECK (Cognome ~* '^[A-Za-z ]+$'),
10.      Salario REAL,
11.     Valutazione INTEGER NOT NULL
12.  );
```

4.1.2. Definizione della Tabella PROGETTO

```
1. --Definizione tabella
2. CREATE TABLE PROGETTO
3. (
4.     CodP INTEGER PRIMARY KEY,
5.     Tipologia VARCHAR(30) NOT NULL,
6.     Descrizione VARCHAR(200)
7. );
8.
9. --Sequenza che gestisce la generazione delle chiavi surrogate per Progetti
10.  CREATE SEQUENCE N_PROGETTO
11.  START WITH 0
12.  MINVALUE 0
13.  INCREMENT BY 1;
14.
15.  --Trigger per impostare, se necessario, la chiave primaria surrogata in
   automatico
16.  CREATE OR REPLACE FUNCTION Funzione_Sequenza_Progetto()
17.  RETURNS TRIGGER AS
18.  $$
19.  BEGIN
20.      IF (NEW.CodP IS NULL) THEN
21.          NEW.CodP := NEXTVAL('N_PROGETTO');
22.      END IF;
23.
24.      RETURN NEW;
25.  END;
26.  $$
27.  LANGUAGE plpgsql;
28.
29.  CREATE TRIGGER Trigger_Sequenza_Progetto
30.  BEFORE INSERT ON PROGETTO
31.  FOR EACH ROW
32.  EXECUTE PROCEDURE Funzione_Sequenza_Progetto();
```

4.1.3. Definizione della Tabella AMBITO

```
1. --Definizione tabella
2. CREATE TABLE AMBITO
3. (
4.     CodP INTEGER ,
5.     Keyword VARCHAR(30) NOT NULL,
6.     CONSTRAINT fk_ambito FOREIGN KEY (CodP)
7.         REFERENCES PROGETTO(CodP)
8.   ON UPDATE CASCADE
9.   ON DELETE NO ACTION
10.  );
```

4.1.4. Definizione della Tabella PARTECIPANTE

```
1. --Definizione tabella
2. CREATE TABLE PARTECIPANTE
3. (
4.     CodF CHAR(16),
5.     CodP INTEGER,
6.     Ruolo VARCHAR(30) NOT NULL,
7.
8.     CONSTRAINT fk_CodF FOREIGN KEY(CodF)
9.         REFERENCES DIPENDENTE(CodF)
10.          ON UPDATE CASCADE
11.          ON DELETE NO ACTION,
12.      CONSTRAINT fk_CodP FOREIGN KEY(CodP)
13.          REFERENCES PROGETTO(CodP)
14.          ON UPDATE CASCADE
15.          ON DELETE NO ACTION,
16.      CONSTRAINT Consistenza_Partecipante UNIQUE (CodF,CodP)
17.  );
```

4.1.5. Definizione della Tabella SALA

```
1. --Definizione tabella
2. CREATE TABLE SALA
3. (
4.     CodSala INTEGER PRIMARY KEY,
5.     Città VARCHAR(30) NOT NULL CHECK (Città ~* '^[A-Za-z ]+$'),
6.     Provincia CHAR(2) NOT NULL CHECK (Provincia ~* '^[A-Z]{2}$'),
7.     Indirizzo VARCHAR(50) NOT NULL CHECK (Indirizzo ~* '^[A-Za-z ]+$'),
8.     NumCivico INTEGER,
9.     NumPosti INTEGER NOT NULL
10.  );
11.
12.  --Sequenza che gestisce la generazione delle chiavi surrogate per Sale
   riunioni
13.   CREATE SEQUENCE N_SALA
14.  START WITH 0
15.  MINVALUE 0
16.  INCREMENT BY 1;
17.
18.  --Trigger per impostare, se necessario, la chiave primaria surrogata in
   automatico
19.  CREATE OR REPLACE FUNCTION Funzione_Sequenza_Sala()
20.  RETURNS TRIGGER AS
21.  $$
22.  BEGIN
23.      IF (NEW.CodSala IS NULL) THEN
24.          NEW.CodSala := NEXTVAL('N_SALA');
25.      END IF;
26.
27.      RETURN NEW;
28.  END;
29.  $$
30.  LANGUAGE plpgsql;
31.
32.  CREATE TRIGGER Trigger_Sequenza_Sala
33.  BEFORE INSERT ON SALA
34.  FOR EACH ROW
35.  EXECUTE PROCEDURE Funzione_Sequenza_Sala();
```

4.1.6. Definizione della Tabella MEETINGF

```
1. --Definizione tabella
2. CREATE TABLE MEETINGF
3. (
4.     CodMF INTEGER PRIMARY KEY,
5.     CodP INTEGER,
6.     Data DATE NOT NULL,
7.     OraI TIME NOT NULL,
8.     OraF TIME NOT NULL,
9.     CodSala INTEGER,
10.
11.      CONSTRAINT fk_mf_progetto FOREIGN KEY (CodP)
12.          REFERENCES PROGETTO(CodP)
13.          ON UPDATE CASCADE
14.          ON DELETE NO ACTION,
15.      CONSTRAINT fk_sala FOREIGN KEY (CodSala)
16.          REFERENCES SALA(CodSala)
17.          ON UPDATE CASCADE
18.          ON DELETE NO ACTION
19.  );
20.
21.  --Sequenza che gestisce la generazione delle chiavi surrogate per
   Meeting Fisici
22.  CREATE SEQUENCE N_MEETINGF
23.  START WITH 0
24.  MINVALUE 0
25.  INCREMENT BY 1;
26.
27.  --Trigger per impostare, se necessario, la chiave primaria surrogata in
   automatico
28.  CREATE OR REPLACE FUNCTION Funzione_Sequenza_MeetingF()
29.  RETURNS TRIGGER AS
30.  $$
31.  BEGIN
32.      IF (NEW.CodMF IS NULL) THEN
33.          NEW.CodMF := NEXTVAL('N_MEETINGF');
34.      END IF;
35.
36.      RETURN NEW;
37.  END;
38.  $$
39.  LANGUAGE plpgsql;
40.
41.  CREATE TRIGGER Trigger_Sequenza_MeetingF
42.  BEFORE INSERT ON MEETINGF
43.  FOR EACH ROW
44.  EXECUTE PROCEDURE Funzione_Sequenza_MeetingF();
```

4.1.7. Definizione della Tabella MEETINGT

```
1. --Definizione tabella
2. CREATE TABLE MEETINGT
3. (
4.     CodMT INTEGER PRIMARY KEY,
5.     CodP INTEGER,
6.     Data DATE NOT NULL,
7.     OraI TIME NOT NULL,
8.     OraF TIME NOT NULL,
9.     Piattaforma VARCHAR(30) NOT NULL,
10.      NumLimite INTEGER,
11.
12.      CONSTRAINT fk_mt_progetto FOREIGN KEY (CodP)
13.          REFERENCES PROGETTO(CodP)
14.          ON UPDATE CASCADE
15.          ON DELETE NO ACTION
16.  );
17.
18.  --Sequenza che gestisce la generazione delle chiavi surrogate per
   Meeting Telematici
19.  CREATE SEQUENCE N_MEETINGT
20.  START WITH 0
21.  MINVALUE 0
22.  INCREMENT BY 1;
23.
24.  --Trigger per impostare, se necessario, la chiave primaria surrogata in
   automatico
25.  CREATE OR REPLACE FUNCTION Funzione_Sequenza_MeetingT()
26.  RETURNS TRIGGER AS
27.  $$
28.  BEGIN
29.      IF (NEW.CodMT IS NULL) THEN
30.          NEW.CodMT := NEXTVAL('N_MEETINGT');
31.      END IF;
32.
33.      RETURN NEW;
34.  END;
35.  $$
36.  LANGUAGE plpgsql;
37.
38.  CREATE TRIGGER Trigger_Sequenza_MeetingT
39.  BEFORE INSERT ON MEETINGT
40.  FOR EACH ROW
41.  EXECUTE PROCEDURE Funzione_Sequenza_MeetingT();
```

4.1.8. Definizione della Tabella PARTECIPAMF

```
1. --Definizione tabella
2.
3. CREATE TABLE PARTECIPAMF
4. (
5.     CodF CHAR(16),
6.     CodMF INTEGER,
7.
8.     CONSTRAINT u_pmf UNIQUE(CodF, CodMF),
9.     CONSTRAINT fk1_MF FOREIGN KEY(CodF)
10.         REFERENCES DIPENDENTE(CodF)
11.         ON UPDATE CASCADE
12.         ON DELETE NO ACTION,
13.      CONSTRAINT fk2_MF FOREIGN KEY(CodMF)
14.         REFERENCES MEETINGF(CodMF)
15.         ON UPDATE CASCADE
16.         ON DELETE NO ACTION
17.  );
```

4.1.9. Definizione della Tabella PARTECIPAMT

```
1. --Definizione tabella
2. CREATE TABLE PARTECIPAMT
3. (
4.     CodF CHAR(16),
5.     CodMT INTEGER,
6.
7.     CONSTRAINT u_pmf UNIQUE(CodF, CodMT),
8.     CONSTRAINT fk1_MT FOREIGN KEY(CodF)
9.   REFERENCES DIPENDENTE(CodF)
10.         ON UPDATE CASCADE
11.         ON DELETE NO ACTION,
12.      CONSTRAINT fk2_MT FOREIGN KEY(CodMT)
13.         REFERENCES MEETINGT(CodMT)
14.         ON UPDATE CASCADE
15.         ON DELETE NO ACTION
16.  );
```

4.2. Implementazione dei vincoli

Di seguito sono riportate le implementazioni dei vincoli che non sono già stati mostrati nelle definizioni delle tabelle.

4.2.1. Implementazione del vincolo Consistenza Valutazione

```
1. --La valutazione di un dipendente sale di 1 punto quando inserito come
   partecipante ad un progetto o ad un meeting:
2.
3. CREATE OR REPLACE FUNCTION Funzione_Aumento_Valutazione()
4. RETURNS TRIGGER AS
5. $$
6. DECLARE
7.     New_Val INTEGER;
8. BEGIN
9.     SELECT Valutazione INTO New_Val
10.      FROM DIPENDENTE
11.      WHERE CodF = NEW.CodF;
12.
13.      New_Val := New_Val + 1;
14.
15.      UPDATE DIPENDENTE
16.      SET Valutazione = New_Val
17.      WHERE CodF = NEW.CodF;
18.
19.      RETURN NULL;
20.  END;
21.  $$
22.  LANGUAGE plpgsql;
23.
24.  CREATE TRIGGER Trigger_Aumento_Valutazione_Progetto
25.  AFTER INSERT ON PARTECIPANTE
26.  FOR EACH ROW
27.  EXECUTE PROCEDURE Funzione_Aumento_Valutazione();
28.
29.  CREATE TRIGGER Trigger_Aumento_Valutazione_MeetingF
30.  AFTER INSERT ON PARTECIPAMF
31.  FOR EACH ROW
32.  EXECUTE PROCEDURE Funzione_Aumento_Valutazione();
33.
34.  CREATE TRIGGER Trigger_Aumento_Valutazione_MeetingT
35.  AFTER INSERT ON PARTECIPAMT
36.  FOR EACH ROW
37.  EXECUTE PROCEDURE Funzione_Aumento_Valutazione();
38.
39.
40.  --La valutazione di un dipendente scende di 1 punto quando cancellato
   come partecipante ad un progetto o ad un meeting:
41.
42.  CREATE OR REPLACE FUNCTION Funzione_Diminuzione_Valutazione()
43.  RETURNS TRIGGER AS
44.  $$
```

```
45.  DECLARE
46.      New_Val INTEGER;
47.  BEGIN
48.      SELECT Valutazione INTO New_Val
49.      FROM DIPENDENTE
50.      WHERE CodF = OLD.CodF;
51.
52.      New_Val := New_Val - 1;
53.
54.      UPDATE DIPENDENTE
55.      SET Valutazione = New_Val
56.      WHERE CodF = OLD.CodF;
57.
58.      RETURN NULL;
59.  END;
60.  $$
61.  LANGUAGE plpgsql;
62.
63.  CREATE TRIGGER Trigger_Diminuzione_Valutazione_Progetto
64.  AFTER DELETE ON PARTECIPANTE
65.  FOR EACH ROW
66.  EXECUTE PROCEDURE Funzione_Diminuzione_Valutazione();
67.
68.  CREATE TRIGGER Trigger_Diminuzione_Valutazione_MeetingF
69.  AFTER DELETE ON PARTECIPAMF
70.  FOR EACH ROW
71.  EXECUTE PROCEDURE Funzione_Diminuzione_Valutazione();
72.
73.  CREATE TRIGGER Trigger_Diminuzione_Valutazione_MeetingT
74.  AFTER DELETE ON PARTECIPAMT
75.  FOR EACH ROW
76.  EXECUTE PROCEDURE Funzione_Diminuzione_Valutazione();
77.
78.
79.  --Quando si modifica un partecipante a un progetto o meeting, scende la
     valutazione del vecchio dipendente e sale quella del nuovo:
80.
81.  CREATE TRIGGER Trigger_Modifica_Valutazione_Up
82.  AFTER UPDATE ON PARTECIPANTE
83.  FOR EACH ROW
84.  EXECUTE PROCEDURE Funzione_Aumento_Valutazione();
85.
86.  CREATE TRIGGER Trigger_Modifica_Valutazione_Down
87.  AFTER UPDATE ON PARTECIPANTE
88.  FOR EACH ROW
89.  EXECUTE PROCEDURE Funzione_Diminuzione_Valutazione();
```

4.2.2. Implementazione del vincolo Unicità Project Manager

```
1. --Ad un progetto deve partecipare un unico project manager:
2.
3. CREATE OR REPLACE FUNCTION Funzione_Unicità_ProjectManager()
4. RETURNS TRIGGER AS
5. $$
6. BEGIN
7.     IF EXISTS (SELECT *
8.                FROM PARTECIPANTE AS P
9.                WHERE P.CodP = NEW.CodP AND P.Ruolo = 'project manager')
   THEN
10.          RAISE EXCEPTION 'ERRORE ESISTE GIA UN PROJECT MANAGER ASSOCIATO
   AL PROGETTO';
11.      END IF;
12.
13.      RETURN NEW;
14.  END;
15.  $$
16.  LANGUAGE plpgsql;
17.
18.  CREATE TRIGGER Vincolo_Unicità_ProjectManager
19.  BEFORE INSERT ON PARTECIPANTE
20.  FOR EACH ROW
21.  WHEN (NEW.Ruolo = 'project manager')
22.  EXECUTE PROCEDURE Funzione_Unicità_ProjectManager();
```

4.2.3. Implementazione del vincolo Totalità Project Manager

```
1. --Un progetto deve avere un project manager:
2.
3. CREATE OR REPLACE FUNCTION Funzione_Totalità_ProjectManager()
4. RETURNS TRIGGER AS
5. $$
6. BEGIN
7.     IF NOT EXISTS (SELECT *
8.               FROM DIPENDENTE
9.               WHERE CodF = 'AAAAAA00A00A000A') THEN
10.          INSERT INTO DIPENDENTE (CodF,Nome,Cognome,Valutazione)
11.              VALUES ('AAAAAA00A00A000A', 'A', 'A', 0);
12.      END IF;
13.
14.      INSERT INTO PARTECIPANTE (CodF, CodP, Ruolo)
15.          VALUES('AAAAAA00A00A000A', NEW.CodP, 'project manager');
16.
17.      RETURN NEW;
18.  END;
19.  $$
20.  LANGUAGE plpgsql;
21.
22.  CREATE TRIGGER Vincolo_Totalità_ProjectManager
23.  AFTER INSERT ON PROGETTO
24.  FOR EACH ROW
25.  EXECUTE PROCEDURE Funzione_Totalità_ProjectManager();
```

4.2.4. Implementazione del vincolo Consistenza Meeting Fisico

```
1. --Due meeting fisici distinti non si possono tenere nella stessa sala,
   alla stessa ora dello stesso giorno:
2.
3. CREATE OR REPLACE FUNCTION Funzione_Consistenza_MeetingF()
4. RETURNS TRIGGER AS
5. $$
6. BEGIN
7.     IF EXISTS (SELECT *
8.             FROM MEETINGF AS MF
9.             WHERE NEW.CodSala = MF.CodSala AND NEW.Data = MF.Data AND
10.                    ((NEW.OraI BETWEEN MF.OraI AND MF.OraF) OR
11.                    (NEW.OraF BETWEEN MF.OraI AND MF.OraF))) THEN
12.         RAISE EXCEPTION 'ERRORE: Un altro meeting è in conflitto con
   quello inserito';
13.     END IF;
14.
15.     RETURN NEW;
16. END;
17. $$
18. LANGUAGE plpgsql;
19.
20. CREATE TRIGGER Vincolo_Consistenza_MeetingF
21. BEFORE INSERT ON MEETINGF
22. FOR EACH ROW
23. EXECUTE PROCEDURE Funzione_Consistenza_MeetingF();
```

4.2.5. Implementazione del vincolo Consistenza Partecipazione Meeting

```
1. --Un dipendente non può partecipare a due meeting (fisici/telematici)
   contemporaneamente:
2.
3.   --Per meeting fisici:
4. CREATE OR REPLACE FUNCTION Funzione_Partecipa_MeetingF()
5. RETURNS TRIGGER AS
6. $$
7. DECLARE
8.     data_meeting DATE;
9.     ora_inizio TIME;
10.      ora_fine TIME;
11.  BEGIN
12.      SELECT MF.Data, MF.OraI, MF.OraF INTO data_meeting, ora_inizio,
   ora_fine
13.      FROM PARTECIPAMF AS PMF NATURAL JOIN MEETINGF AS MF
14.      WHERE NEW.CodF = PMF.CodF AND NEW.CodMF = MF.CodMF;
15.
16.      IF EXISTS (SELECT *
17.                      FROM PARTECIPAMF AS PMF NATURAL JOIN
18.                      MEETINGF AS MF
19.                      WHERE NEW.CodF = PMF.CodF AND data_meeting = MF.Data
   AND
20.                          ((ora_inizio BETWEEN MF.OraI AND MF.OraF) OR
21.                           (ora_fine BETWEEN MF.OraI AND MF.OraF)))
22.          OR EXISTS (SELECT *
23.                           FROM PARTECIPAMT AS PMT NATURAL JOIN
24.                                   MEETINGT AS MT
25.                           WHERE NEW.CodF = PMT.CodF AND data_meeting =
   MT.Data AND
26.                               ((ora_inizio BETWEEN MT.OraI AND MT.OraF) OR
27.                               (ora_fine BETWEEN MT.OraI AND MT.OraF))) THEN
28.  RAISE EXCEPTION 'ERRORE: Questo dipendente partecipa già ad un altro
   meeting in quel lasso di tempo';
29.      END IF;
30.
31.      RETURN NEW;
32.  END;
33.  $$
34.  LANGUAGE plpgsql;
35.
36.  CREATE TRIGGER Vincolo_Partecipa_MeetingF
37.  BEFORE INSERT ON PARTECIPAMF
38.  FOR EACH ROW
39.  EXECUTE PROCEDURE Funzione_Partecipa_MeetingF();
40.
41.          --Per meeting telematici:
42.  CREATE OR REPLACE FUNCTION Funzione_Partecipa_MeetingT()
43.  RETURNS TRIGGER AS
44.  $$
45.  DECLARE
46.      data_meeting DATE;
47.      ora_inizio TIME;
48.      ora_fine TIME;
49.  BEGIN
50.      SELECT MT.Data, MT.OraI, MT.OraF INTO data_meeting, ora_inizio,
   ora_fine
51.      FROM PARTECIPAMT AS PMT NATURAL JOIN
```

```
52.          MEETINGT AS MT
53.      WHERE NEW.CodF = PMT.CodF AND NEW.CodMT = MT.CodMT;
54.
55.      IF EXISTS (SELECT *
56.                      FROM PARTECIPAMF AS PMF NATURAL JOIN
57.                           MEETINGF AS MF
58.                      WHERE NEW.CodF = PMF.CodF AND data_meeting = MF.Data
   AND
59.                          ((ora_inizio BETWEEN MF.OraI AND MF.OraF) OR
60.                           (ora_fine BETWEEN MF.OraI AND MF.OraF)))
61.          OR EXISTS (SELECT *
62.                           FROM PARTECIPAMT AS PMT NATURAL JOIN
63.                                   MEETINGT AS MT
64.                           WHERE NEW.CodF = PMT.CodF AND data_meeting =
   MT.Data AND
65.                               ((ora_inizio BETWEEN MT.OraI AND MT.OraF) OR
66.                               (ora_fine BETWEEN MT.OraI AND MT.OraF))) THEN
67.  RAISE EXCEPTION 'ERRORE: Questo dipendente partecipa già ad un altro
   meeting in quel lasso di tempo';
68.      END IF;
69.
70.      RETURN NEW;
71.  END;
72.  $$
73.  LANGUAGE plpgsql;
74.
75.  CREATE TRIGGER Vincolo_Partecipa_MeetingT
76.  BEFORE INSERT ON PARTECIPAMT
77.  FOR EACH ROW
78.  EXECUTE PROCEDURE Funzione_Partecipa_MeetingT();
```

4.2.6. Implementazione del vincolo Capienza Sala Riunioni

```
1. --Il numero di partecipanti ad un meeting fisico non può eccedere la
   capienza della sala:
2.
3. CREATE OR REPLACE FUNCTION Funzione_Capienza_Sala()
4. RETURNS TRIGGER AS
5. $$
6. DECLARE
7.     Num_Posti INTEGER;
8.     Num_Dipendenti INTEGER;
9. BEGIN
10.      SELECT NumPosti INTO Num_Posti
11.      FROM MEETINGF AS MF NATURAL JOIN SALA AS S
12.      WHERE MF.CodMF = NEW.CodMF;
13.
14.      SELECT COUNT(CodF) INTO Num_Dipendenti
15.      FROM PARTECIPAMF AS PF
16.      WHERE PF.CodMF = NEW.CodMF
17.      GROUP BY PF.CodMF;
18.
19.      IF (Num_Dipendenti >= Num_Posti) THEN
20.          RAISE EXCEPTION 'ERRORE: Il limite di partecipanti al meeting è
   già stato raggiunto';
21.      END IF;
22.      RETURN NEW;
23.  END;
24.  $$
25.  LANGUAGE plpgsql;
26.
27.  CREATE TRIGGER Vincolo_Capienza_Sala
28.  BEFORE INSERT ON PARTECIPAMF
29.  FOR EACH ROW
30.  EXECUTE PROCEDURE Funzione_Capienza_Sala();
```

4.2.7. Implementazione del vincolo Limite Piattaforma Videoconferenze

```
1. --Il numero di partecipanti ad un meeting telematico non può eccedere il
   limite di partecipanti dichiarato (se esiste):
2. CREATE OR REPLACE FUNCTION Funzione_Limite_MeetingT()
3. RETURNS TRIGGER AS
4. $$
5. DECLARE
6.      Num_Massimo INTEGER;
7.      Num_Partecipanti INTEGER;
8. BEGIN
9.      SELECT NumLimite INTO Num_Massimo
10.       FROM MEETINGT AS MT
11.       WHERE MT.CodMT = NEW.CodMT;
12.
13.       SELECT COUNT(CodF) INTO Num_Partecipanti
14.       FROM PARTECIPAMT AS PT
15.       WHERE PT.CodMT = NEW.CodMT
16.       GROUP BY PT.CodMT;
17.
18.       IF(Num_Massimo IS NOT NULL AND Num_Partecipanti >= Num_Massimo) THEN
19.               RAISE EXCEPTION 'ERRORE: É stato superato il limite massimo
   di partecipanti al meeting';
20.       END IF;
21.       RETURN NEW;
22.   END;
23.   $$
24.   LANGUAGE plpgsql;
25.
26.   CREATE TRIGGER Vincolo_Limite_MeetingT
27.   BEFORE INSERT ON PARTECIPAMT
28.   FOR EACH ROW
29.   EXECUTE PROCEDURE Funzione_Limite_MeetingT();
```

4.3. Funzioni, Procedure e altre automazioni

Di seguito sono riportate le stored function e le stored procedures che si è deciso di implementare per semplificare alcuni aspetti dell'utilizzo della base di dati.

4.3.1. Inserimento di un Progetto

La stored procedure Inserisci_Progetto(tipologia, codpm, descrizione), dati in input la tipologia di un progetto, il codice fiscale del suo Project Manager ed una sua breve descrizione (anche vuota), inserisce nel database un nuovo progetto con i dati inseriti e inserisce il dipendente scelto come Project Manager del progetto nella tabella PARTECIPANTE. A seguire la definizione:

```
1. CREATE OR REPLACE PROCEDURE Insert_Progetto
2. (
3.     IN tipologia VARCHAR(30),
4.     IN codpm CHAR(16),
5.     IN descrizione VARCHAR(200)
6. )
7. AS
8. $$
9. BEGIN
10.      INSERT INTO PROGETTO (CodP, Tipologia, Descrizione)
11.          VALUES (0, tipologia, descrizione);
12.
13.      UPDATE PARTECIPANTE
14.      SET CodF = codpm
15.      WHERE CodP = new_prog AND Ruolo ILIKE 'Project Manager';
16.  END;
17.  $$
18.  LANGUAGE plpgsql;
```

4.3.2. Calcolo automatico del salario medio aziendale

La stored function get_Salario_Medio() calcola e restituisce il salario medio tra tutti i dipendenti dell'azienda. Di seguito è riportata la definizione:

```
1. CREATE OR REPLACE FUNCTION get_Salario_Medio()
2. RETURNS REAL
3. AS
4. $$
5. DECLARE
6.   ret REAL;
7. BEGIN
8.      ret = (SELECT AVG(D.Salario)
9.             FROM DIPENDENTE AS D);
10.
11.          RETURN ret;
12.  END
13.  $$
14.  LANGUAGE plpgsql;
```