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 in Informatica

Insegnamento di Basi di Dati I

Anno Accademico 2020/2021

Progettazione e sviluppo di una base di dati relazionale per un sistema di planning per la gestione di progetti.

Autori:

Chehade Bianca Giada

Matricola N86003209

b.chehade@studenti.unina.it

Zaza Francesco Rosario

Matricola N86002501

fra.zaza@studenti.unina.it

Docenti:

Peron Adriano

Barra Silvio

Questa pagina è stata lasciata intenzionalmente bianca.

Indice

Capitolo 1

1. Descrizione del progetto

1.1 Descrizione sintetica

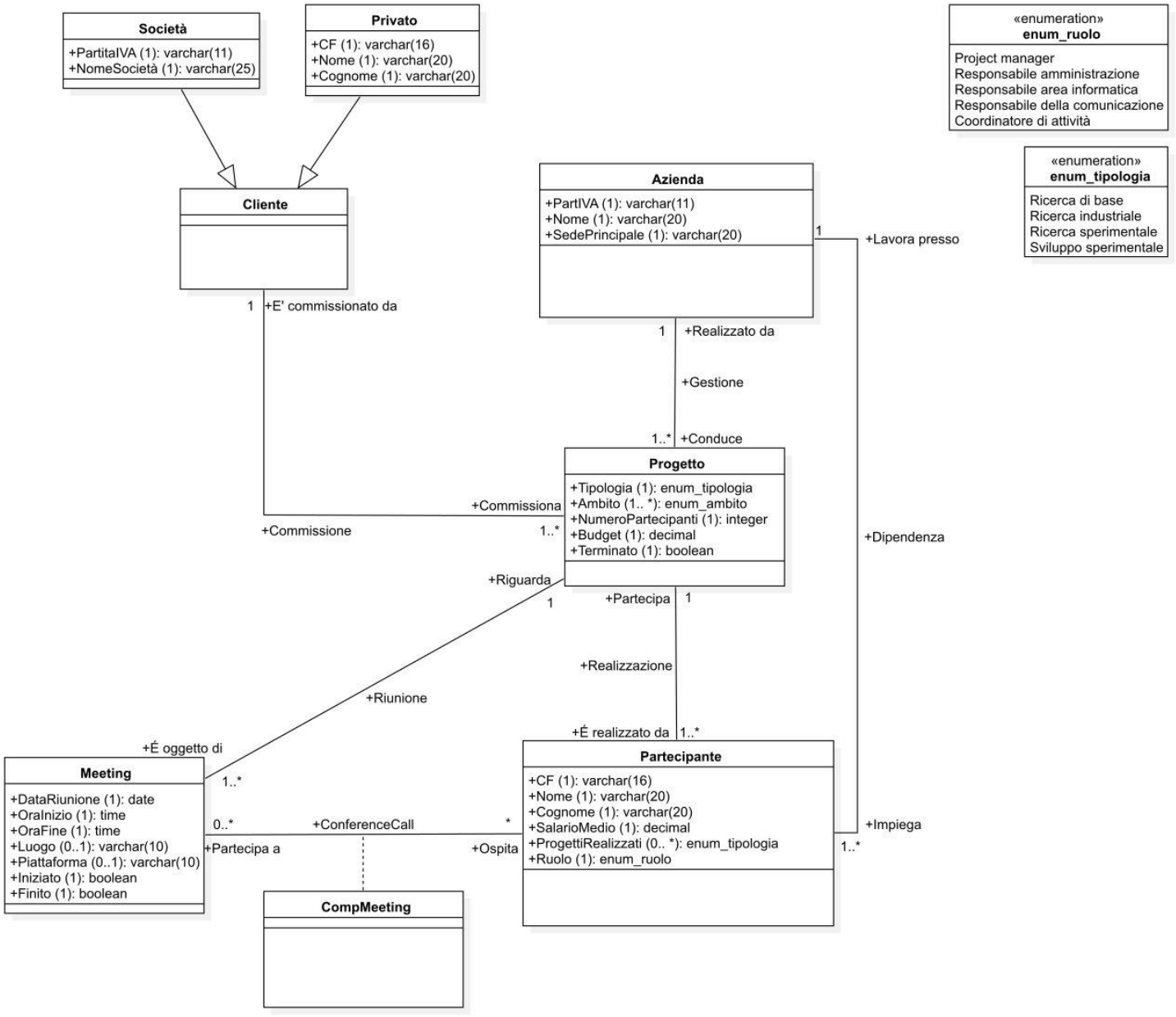
Si svilupperà ed implementerà una base di dati relazionale utile alla gestione di progetti in un'azienda. Tale database prevede la possibilità di tenere traccia dei partecipanti a ciascun progetto, identificando un ruolo per ognuno di essi (per ogni progetto ci sarà un solo project manager). Ad ogni progetto è associata una tipologia ("Ricerca di base", "Ricerca Industriale", "Ricerca sperimentale", "Sviluppo Sperimentale", ...) ed uno o più ambiti (Economia, Medicina, …). Il sistema permetterà anche l'organizzazione di meeting fisicamente, in sale riunioni, o telematicamente su una piattaforma di videoconferenza. Si terrà traccia delle partecipazioni ai progetti ed ai meeting, ai fini della valutazione del singolo partecipante. In fase di creazione di un nuovo progetto, i partecipanti dovranno essere selezionati in base a criteri di ricerca che includono anche il salario medio e la valutazione aziendale del partecipante, oltre alla tipologia di progetti cui ha preso parte.

Capitolo 2

2. Progettazione concettuale

In questo capitolo inizia la progettazione della base di dati a livello concettuale. Dal risultato dell'analisi dei requisiti che devono essere soddisfatti si arriverà ad uno schema concettuale indipendente dalla struttura dei dati e dall'implementazione fisica. Tale schema concettuale sarà rappresentato usando un class diagram UML, nel quale saranno evidenziate le entità rilevanti ai fini della rappresentazione dei dati e le relazioni che intercorrono tra esse.

2.1 Class diagram



2.2 Ristrutturazione del class diagram

Si procede alla ristrutturazione del class diagram con lo scopo di renderlo idoneo alla traduzione in schemi relazionali e di migliorare l'efficienza dell'implementazione. Al termine del procedimento il class diagram non conterrà attributi strutturati, attributi multipli e gerarchie di specializzazione.

2.2.1 Analisi delle ridondanze

Non sono presenti ridondanze significative da eliminare.

2.2.2 Analisi degli identificativi

Si procede all'aggiunta, per alcune entità, di chiavi "surrogate". Tali attributi sono identificativi numerici che permetteranno più agevolmente un'identificazione univoca per ciascuna istanza.

In particolare, tali chiavi sintetiche saranno:

- UserID, per la tabella Partecipante;
- CodMeeting, per la tabella Meeting;
- CodProgetto, per la tabella Progetto.

2.2.3 Rimozione degli attributi multipli

L'attributo Ambito della classe Progetto è da eliminare. Per andare incontro alla necessità di avere più ambiti possibili per un determinato progetto, è ragionevole procedere alla creazione di una nuova classe: Ambito.

Lo stesso procedimento è da seguire per l'attributo multiplo ProgettiRealizzati della classe Partecipante; si procede alla creazione della classe ProgRealizzato.

2.2.4 Rimozione delle gerarchie di specializzazione

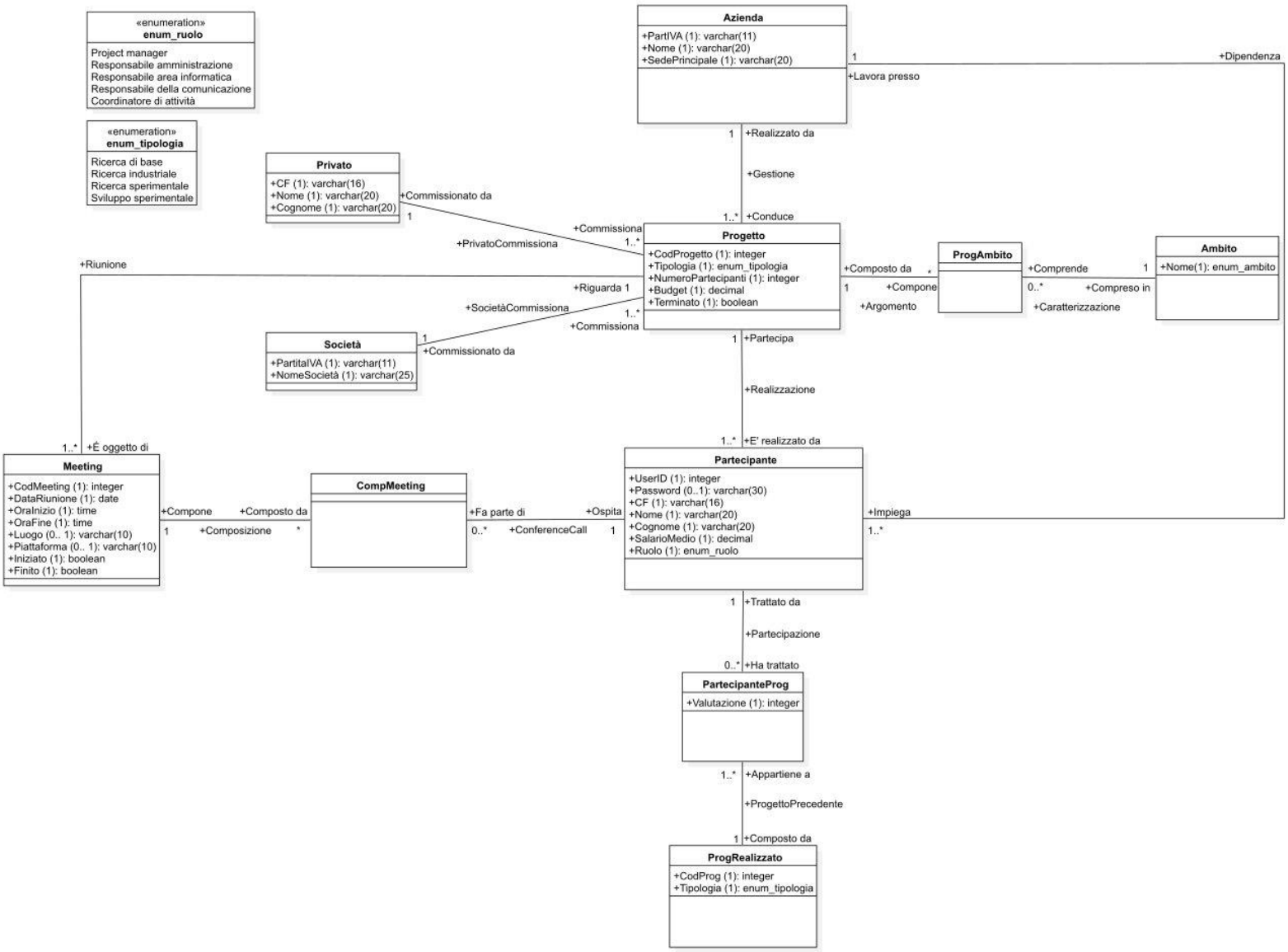
Si procede con l'eliminazione delle specializzazioni della classe Cliente. Si tratta di una specializzazione totale e disgiunta, dunque si procederà all'eliminazione "schiacciando" la superclasse nelle sottoclassi.

2.2.5 Gestione delle cardinalità molti a molti

La rimozione degli attributi multipli Ambito e ProgettiRealizzati è stata gestita con la creazione delle classi Ambito e ProgRealizzato. Tali tabelle sono legate, rispettivamente, alle classi Progetto e Partecipante tramite associazioni di cardinalità molti a molti. Si procede dunque con la creazione delle tabelle ProgAmbito e PartecipanteProg, seguita dalla revisione delle rispettive cardinalità di associazione. Per quanto riguarda l'associazione molti a molti che lega la tabella Meeting alla

tabella Partecipante, si è resa la classe di associazione CompMeeting una vera e propria classe tra le due, revisionando anche questa volta le cardinalità di associazione.

2.3 Class diagram ristrutturato



2.4 Dizionario dei dati

Nei seguenti sottoparagrafi sono esposte, nel dettaglio, le informazioni riguardanti le classi, le associazioni ed i vincoli implementati.

2.4.1 Dizionario delle classi

Classe	Descrizione	Attributi
Azienda	Descrittore dell'azienda che realizza un progetto.	PartIVA (varchar): indica la partita IVA di un'azienda, che la identifica univocamente. Nome (varchar): indica il nome dell'azienda. SedePrincipale (varchar): indica dove è situata l'azienda.
Progetto	Descrittore del progetto realizzato da un'azienda e commissionato da un privato/società.	CodProgetto (integer): identifica univocamente ogni istanza di Progetto. Tipologia (enumerazione): indica la tipologia del progetto. NumeroPartecipanti (integer): specifica quante persone lavorano al progetto. Budget (decimal): specifica il costo totale del progetto. Terminato (boolean, valore di default false): attributo che indica se il progetto in questione è giunto al termine o è ancora in esecuzione.
Partecipante	Descrive ciascun partecipante di un progetto.	UserID (integer): identificativo numerico di ciascun partecipante al progetto. Password (varchar, opzionale): indica la sequenza di caratteri alfanumerici necessaria per accedere all'account. CF (varchar): codice fiscale del partecipante. Nome (varchar): nome del partecipante.

		Cognome (varchar): cognome del partecipante. SalarioMedio (decimal): indica il guadagno medio di un partecipante. Ruolo (enumerazione): specifica il ruolo del partecipante nel progetto a cui sta lavorando.
Società	Descrittore della società che commissiona il progetto.	PartitaIVA (varchar): partita IVA della società. NomeSocietà (varchar): indica il nome della società.
Privato	Descrittore del privato che commissiona il progetto.	CF (varchar): codice fiscale del privato. Nome (varchar): nome del privato. Cognome (varchar): cognome del privato.
Meeting	Descrittore delle riunioni.	CodMeeting (integer): sequenza numerica che permette di identificare univocamente la riunione. DataRiunione (date): indica il giorno della riunione. OraInizio (time): indica l'ora di inizio della riunione. OraFine (time): indica l'ora di fine della riunione. Luogo (varchar, opzionale): indica il luogo presso il quale si svolge la riunione. Piattaforma (varchar, opzionale): indica la piattaforma utilizzata nel caso in cui il meeting si tenga in modalità telematica. Iniziato (boolean, valore di default false): attributo che indica se la riunione è iniziata.

		Finito (boolean, valore di default false): attributo che indica se la riunione è finita.
Ambito	Descrittore dell'ambito di un progetto.	Nome (varchar): indica l'ambito di sviluppo del progetto.
ProgRealizzato	Descrittore dei progetti realizzati in passato dai progettisti.	CodProg (integer): codice identificativo di un progetto. Tipologia (enumerazione): indica di che tipo è il progetto.
CompMeeting	Codifica l'associazione molti a molti tra Meeting e Partecipante	
PartecipanteProg	Codifica l'associazione molti a molti tra Partecipante e ProgRealizzato	Valutazione (integer): fornisce una valutazione numerica di un partecipante; essa sarà influenzata dalla partecipazione del dipendente ai progetti e ai relativi meeting. Tale attributo è stato inserito nella classe PartecipanteProg (nella quale potrà essere aggiornata) piuttosto che nella classe Partecipante in quanto una valutazione per un progetto ha senso dal momento in cui tale progetto è terminato.
ProgAmbito	Codifica l'associazione molti a molti tra Progetto e Ambito	

2.4.2 Dizionario delle associazioni

Nome	Descrizione	Classi Coinvolte
Dipendenza	Esprime il legame lavorativo tra un'azienda e un lavoratore.	Partecipante[1.. *] ruolo Lavora presso: indica in quale azienda lavora un partecipante. Azienda[1] ruolo Impiega: indica quale partecipante impiega l'azienda.
Gestione	Indica da quale azienda è gestito un progetto.	Progetto [1.. *] ruolo Realizzato da: indica da quale azienda è realizzato un progetto. Azienda[1] ruolo Conduce: indica il progetto che realizza un'azienda.
Realizzazione	Esprime l'impiego di un partecipante in un progetto.	Progetto[1] ruolo E' realizzato da: indica da quale partecipante è realizzato il progetto. Partecipante[1..*] ruolo Partecipa: indica a quale progetto partecipa un progettista.
PrivatoCommissiona	Esprime la commissione di un progetto da parte di un privato.	Privato[1] ruolo Commissiona: indica quale progetto commissiona un privato. Progetto[1.. *] ruolo Commissionato da: indica da chi è commissionato il progetto.
SocietàCommissiona	Esprime la commissione di un progetto da parte di una società.	Società[1] ruolo Commissiona: indica quale progetto commissiona una società. Progetto[1.. *] ruolo Commissionato da: indica da chi è commissionato il progetto.
Argomento	Esprime la relazione tra un progetto e il suo ambito.	Progetto[1] ruolo Composto da: indica l'ambito del progetto. ProgAmbito[*] ruolo Compone: indica a quale progetto si riferisce un ambito.

Caratterizzazione	Esprime la relazione tra un progetto e il suo ambito.	Ambito[1] ruolo comprende: indica che progetto comprende ciascun ambito. ProgAmbito[*] ruolo Compreso in: indica in che ambito è compreso un progetto
Partecipazione	Esprime l'appartenenza di un partecipante a un progetto.	Partecipante[1] ruolo ha trattato: indica il tipo di progetti ai quali ha lavorato un progettista. PartecipanteProg[*] ruolo trattato da: indica da quale partecipante è stato realizzato un progetto.
ProgettoPrecedente	Esprime la relazione tra un partecipante e i progetti ai quali ha lavorato in precedenza.	ProgRealizzato[1] ruolo Appartiene a: indica a quale partecipante appartiene un progetto. PartecipanteProg[*] ruolo Composto da: indica da quale partecipante è stato realizzato un progetto.
ConferenceCall	Esprime la relazione tra una riunione e i suoi partecipanti.	Partecipante[1] ruolo Fa parte di: indica a quale riunione partecipa un progettista. CompMeeting[*] ruolo Ospita: indica quali progettisti ospita un meeting.
Composizione	Descrive le caratteristiche di ciascun meeting,	Meeting[1] ruolo Composto da: indica la composizione del meeting. CompMeeting[*] ruolo Compone: indica quale meeting ha le caratteristiche in questione.
Riunione	Descrive quale progetto viene discusso in ciascun meeting.	Meeting[1..*] ruolo Riguarda: indica quale progetto viene trattato nella riunione. Progetto[1] ruolo E' oggetto di: indica in quale riunione viene discusso un progetto.

2.4.3 Dizionario dei vincoli

Nome	Descrizione
Controllo_CF	Controlla la validità del codice fiscale inserito.
Vincolo_Cliente	Controlla che un progetto possa essere commissionato in maniera esclusiva da un privato o da una società.
NumeroPartecipanti_Check	Controlla che il numero di partecipanti a un progetto non sia superiore a 30.
Controllo_Valutazione	Controlla che il punteggio inserito per la valutazione di un partecipante in un progetto sia compreso tra 0 e 5.
Unico_CF	Garantisce che ogni codice fiscale sia unico.
Unique_Partecipante_Meeting	Controlla che un partecipante possa partecipare a un solo meeting alla volta.
Luogo_Riunione	Controlla che una riunione possa tenersi in maniera esclusiva su un piattaforma telematica o in un luogo fisico.
FineProgetto	Quando un progetto termina e il suo stato viene aggiornato, esso viene eliminato dai processi attualmente in corso e aggiunto ai progetti realizzati.
ProgettoMismatch	Consente di partecipare ad un meeting soltanto se si sta lavorando al progetto che riguarda la riunione.
LuogoOccupato	Controlla che non possa esserci una riunione in un luogo fisico se per l'orario prefissato tale posto non è libero.
ComposizioneMeeting	Consente di avviare una riunione su un progetto solo se almeno un 1/3 del totale dei partecipanti è presente.
Meeting_Senza_Manager	Consente di avviare un meeting su un progetto soltanto se il relativo project manager è presente.
MeetingNonPermesso	Controlla che non si possa partecipare ad un meeting se non si ha alcun progetto a carico.

ValutazioneAziendale	Controlla che ogni partecipante abbia una sola valutazione per progetto.
OneMeeting	Controlla che ci sia solo un meeting in corso alla volta per lo stesso progetto.
ProjectManager	Controlla che ogni progetto abbia uno e un solo project manager.
ValiditàDataMeeting	Controlla che una riunione possa essere fissata solo in giorni successivi o uguali al giorno corrente.
ValiditàOrarioMeeting	Controlla che l'orario di fine riunione non possa essere precedente all'orario di inizio e che ogni meeting debba durare almeno 30 minuti.
MeetingFuturi	Permette di calendarizzare un meeting solo con un anticipo di almeno 14 giorni.
Check_Password	Controlla la complessità della password.
BudgetLegale	Permette solo progetti con budget superiore o uguale a 5000€.
AmbitoProgetto	Inserisce automaticamente il codice di un progetto appena inserito nella tabella con riferimento al suo ambito di realizzazione.
InserimentoMeetingAmbito	Controlla che si possa operare con un codice progetto nella tabella Meeting soltanto se per esso è stato specificato un ambito.
InserimentoPartecipanteAmbito	Controlla che si possa operare con un codice progetto nella tabella Partecipante soltanto se per esso è stato specificato un ambito.
IncrementaPartecipanti	Incrementa automaticamente l'attributo *NumeroPartecipanti* di un progetto dopo ogni inserimento in Partecipante.
StipendioLegale	Controlla che il salario medio di ogni progettista sia di almeno 1500€.
CheckPartitaIVA	Controlla la validità della partita IVA per le società commissionanti.
CheckPartIVA	Controlla la validità della partita IVA per l'azienda che conduce un progetto.

Commissione	Controlla che un’azienda non possa commissionare un progetto a sé stessa.

Capitolo 3

3. Progettazione logica

In questo capitolo sarà trattata la fase successiva della progettazione della base di dati: si tradurrà lo schema concettuale in uno schema logico. Negli schemi relazionali che seguiranno le chiavi primarie sono indicate con una singola sottolineatura mentre le chiavi esterne con una doppia sottolineatura.

3.1 Traduzione in schemi relazionali

Azienda (PartIVA, Nome, SedePrincipale).
Chiavi esterne: nessuna.

Privato (CF, Nome, Cognome)
Chiavi esterne: nessuna.

Società (PartitaIVA, NomeSocietà)
Chiavi esterne: nessuna.

Progetto (CodProgetto, Tipologia, NumeroPartecipanti, Budget, Terminato, PartIVA, CF, PartitaIVA).
Chiavi esterne: PartIVA → Azienda.PartIVA; CF → Privato.CF; PartitaIVA → Società.PartitaIVA.

Partecipante (UserID, Pw, CF, Nome, Cognome, SalarioMedio, Ruolo, CodProgetto, PartIVA).
Chiavi esterne: CodProgetto → Progetto.CodProgetto; PartIVA → Azienda.PartIVA.

ProgRealizzato (CodProg, Tipologia)
Chiavi esterne: nessuna.

PartecipanteProg (CodProg, UserID, Valutazione).
Chiavi esterne: CodProg → ProgRealizzato.CodProg; UserID → Partecipante.UserID.

Meeting (CodMeeting, DataRiunione, OraInizio, OraFine, Luogo, Piattaforma, Iniziato, Finito, CodProgetto).
Chiavi esterne: CodProgetto → Progetto.CodProgetto.

CompMeeting (CodMeeting, UserID).
Chiavi esterne: CodMeeting → Meeting.CodMeeting; UserID → Partecipante.UserID.

Ambito (Nome).

Chiavi esterne: nessuna

ProgAmbito (CodProgetto, Nome)

Chiavi esterne: CodProgetto → Progetto.CodProgetto; Nome → Ambito.Nome.

3.1.1 Traduzione delle associazioni

Associazione	Implementazione
Dipendenza	Chiave esterna in **Partecipante** → **Azienda**
Gestione	Chiave esterna in **Progetto** → **Azienda**
Realizzazione	Chiave esterna in **Partecipante** → **Progetto**
PrivatoCommissiona	Chiave esterna in **Progetto** → **Privato**
SocietàCommissiona	Chiave esterna in **Progetto** → **Società**
Argomento	Chiave esterna in **ProgAmbito** → **Progetto**
Caratterizzazione	Chiave esterna in **ProgAmbito** → **Ambito**
Partecipazione	Chiave esterna in **PartecipanteProg** → **Partecipante**
ProgettoPrecedente	Chiave esterna in **PartecipanteProg** → **ProgRealizzato**
ConferenceCall	Chiave esterna in **CompMeeting** → **Partecipante**
Composizione	Chiave esterna in **CompMeeting** → **Meeting**
Riunione	Chiave esterna in **Meeting** → **Progetto**

3.2 Schema Logico

In base a quanto detto nella sezione precedente, si previene al seguente schema logico.

Azienda	(PartIVA, Nome, SedePrincipale)
Privato	(CF, Nome, Cognome)
Società	(PartitaIVA, NomeSocietà)
Progetto	(CodProgetto, Tipologia, NumeroPartecipanti, Budget, Terminato, PartIVA, CF, PartitaIVA)
Partecipante	(UserID, Pw, CF, Nome, Cognome, Ruolo, SalarioMedio, CodProgetto, PartIVA).
ProgRealizzato	(CodProg, Tipologia)
PartecipanteProg	(CodProg, UserID, Valutazione).
Meeting	(CodMeeting, DataRiunione, OraInizio, OraFine, Piattaforma, Luogo, Iniziato, Finito, CodProgetto).
CompMeeting	(CodMeeting, UserID).
Ambito	(Nome).
ProgAmbito	(CodProgetto, Nome)

Capitolo 4

4. Progettazione fisica

4.1 Definizione delle tabelle

Di seguito la definizione delle tabelle.

4.1.1 Definizione della tabella Azienda

```
--TABELLA
CREATE TABLE Azienda (
      PartIVA varchar(11) PRIMARY KEY,
      Nome varchar(20) NOT NULL,
      SedePrincipale varchar(12) NOT NULL
);

--VINCOLI
ALTER TABLE Azienda
ADD CONSTRAINT CheckPartIVA CHECK (((PartIVA)::text ~* '[0-9]{7}0[0-
9]{3}'::text));
```

4.1.2 Definizione della tabella Privato

```
--TABELLA
CREATE TABLE Privato (
      CF varchar(16) PRIMARY KEY,
      Nome varchar(20) NOT NULL,
      Cognome varchar(20) NOT NULL,
);

--VINCOLI
ALTER TABLE Privato
ADD CONSTRAINT Controllo_CF CHECK (CF ~* '[A-Z]{6}\d{2}[A-Z]\d{2}[A-
Z]\d{3}[A-Z]');
```

4.1.3 Definizione della tabella Società

```
 --TABELLA
CREATE TABLE Società (
      PartitaIVA varchar(11) PRIMARY KEY,
      NomeSocietà varchar(25) NOT NULL
);

--VINCOLI
ALTER TABLE Società
ADD CONSTRAINT CheckPartitaIVA CHECK (((PartitaIVA)::text ~* '[0-
9]{7}0[0-9]{3}'::text));
```

4.1.4 Definizione della tabella Progetto

```
--TABELLA
CREATE TABLE Progetto (
      CodProgetto integer PRIMARY KEY,
      Tipologia enum_tipologia NOT NULL,
      NumeroPartecipanti integer DEFAULT 0 NOT NULL,
      Budget decimal NOT NULL,
      PartIVA varchar(11) NOT NULL REFERENCES Azienda(PartIVA),
      CF varchar(16) REFERENCES Privato(CF),
      PartitaIVA varchar(11) REFERENCES Società(PartitaIVA),
      Terminato boolean DEFAULT false NOT NULL
);

--VINCOLI
ALTER TABLE Progetto
ADD CONSTRAINT Vincolo_Cliente CHECK ((CF IS NOT NULL AND PartitaIva IS
NULL) OR (CF IS NULL AND  PartitaIva IS NOT NULL)),
ADD CONSTRAINT NumeroPartecipanti_Check CHECK (numeropartecipanti >= 0
AND numeropartecipanti <= 30),
ADD CONSTRAINT BudgetLegale CHECK (Budget >= 5000),
ADD CONSTRAINT Commissione CHECK (PartIVA <> PartitaIVA);
```

4.1.5 Definizione della tabella Partecipante

```
--TABELLA
CREATE TABLE Partecipante (
      UserID integer PRIMARY KEY,
      Pw varchar(30),
      CF varchar(16) NOT NULL,
      Nome varchar(20) NOT NULL,
      Cognome varchar(20) NOT NULL,
      Ruolo enum_ruolo NOT NULL,
      SalarioMedio decimal NOT NULL,
      CodProgetto integer NOT NULL REFERENCES Progetto(CodProgetto),
      PartIVA varchar(11) NOT NULL REFERENCES Azienda(PartIVA)
);

--VINCOLI
ALTER TABLE Partecipante
ADD CONSTRAINT Controllo_CF CHECK (CF ~* '[A-Z]{6}\d{2}[A-Z]\d{2}[A-
Z]\d{3}[A-Z]'),
ADD CONSTRAINT Controllo_Valutazione CHECK ((Valutazione >= 0) AND
(Valutazione <=5)),
ADD CONSTRAINT Check_Password CHECK (pw ~*
'(?=^.{6,}$)((?=.*\d)|(?=.*\W+))(?![.\n])(?=.*[A-Z])(?=.*[a-z]).*$'),
ADD CONSTRAINT Unico_CF UNIQUE CF,
ADD CONSTRAINT StipendioLegale CHECK (SalarioMedio >= 1500);
```

4.1.6 Definizione della tabella ProgRealizzato

```
--TABELLA
CREATE TABLE ProgRealizzato (
      CodProg integer PRIMARY KEY,
      Tipologia enum_tipologia NOT NULL
);
```

4.1.7 Definizione della tabella PartecipanteProg

```
--TABELLA
CREATE TABLE PartecipanteProg (
      Valutazione integer,
      UserID integer NOT NULL REFERENCES Partecipante(UserID),
      CodProg integer NOT NULL REFERENCES ProgRealizzato(CodProg)
);
```

4.1.8 Definizione della tabella Meeting

```
--TABELLA
CREATE TABLE Meeting(
      CodMeeting integer PRIMARY KEY,
      DataRiunione date NOT NULL,
      OraInizio time NOT NULL,
      OraFine time NOT NULL,
      Piattaforma varchar(10),
      Luogo varchar(10),
      CodProgetto integer NOT NULL REFERENCES Progetto(CodProgetto),
      Iniziato boolean DEFAULT false NOT NULL,
      Finito boolean DEFAULT false NOT NULL
);

--VINCOLI
ALTER TABLE Meeting
ADD CONSTRAINT Luogo_Riunione CHECK ((Luogo IS NOT NULL AND Piattaforma
IS NULL) OR (Luogo IS NULL AND Piattaforma IS NOT NULL),
ADD CONSTRAINT ValiditàOrarioMeeting CHECK ( (orafine-orainizio) >
INTERVAL '30 minutes'),
ADD CONSTRAINT ValiditàDataMeeting CHECK ( datariunione >= current_date),
ADD CONSTRAINT Meeting_Futuri CHECK ( (datariunione - current_date) >=
14);
```

4.1.9 Definizione della tabella CompMeeting

```
--TABELLA
CREATE TABLE CompMeeting (
      CodMeeting integer NOT NULL REFERENCES Meeting(CodMeeting),
      UserID integer NOT NULL REFERENCES Partecipante(UserID)
);

--VINCOLI
ALTER TABLE CompMeeting
ADD CONSTRAINT Unique_Partecipante_Meeting UNIQUE (CodMeeting, UserID);
```

4.1.10 Definizione della tabella Ambito

```
--TABELLA
CREATE TABLE Ambito (
      Nome varchar(25) PRIMARY KEY
);
```

4.1.11 Definizione della tabella ProgAmbito

```
--TABELLA
CREATE TABLE ProgAmbito (
      CodProgetto integer NOT NULL REFERENCES Progetto(CodProgetto),
      Nome varchar(25) REFERENCES Ambito(Nome)
);
```

4.2 Definizione vincoli di dominio

Di seguito la definizione dei vincoli di dominio.

4.2.1 Dominio enum_ruolo

```
CREATE DOMAIN enum_ruolo AS character varying(50)
CONSTRAINT enum_ruolo CHECK
(upper(VALUE) = ANY ('RESPONSABILE DELLA COMUNICAZIONE', 'COORDINATORE DI
ATTIVITÀ', 'RESPONSABILE AMMINISTRAZIONE', 'RESPONSABILE AREA
INFORMATICA', 'PROJECT MANAGER'));
```

4.2.2 Dominio enum_tipologia

```
CREATE DOMAIN enum_tipologia AS character varying(50)
CONSTRAINT enum_tipologia CHECK
(upper(VALUE) = ANY ('RICERCA DI BASE', 'RICERCA INDUSTRIALE', 'RICERCA
SPERIMENTALE', 'SVILUPPO SPERIMENTALE'));
```

4.3 Definizione trigger functions

Di seguito la definizione dei trigger implementati con le relative funzioni.

4.3.1 Definizione trigger function Storico_Progetti

```
CREATE OR REPLACE FUNCTION Storico_Progetti()
RETURNS TRIGGER AS
$FineProgetto$
DECLARE Progettista integer;
BEGIN
     IF New.Terminato=TRUE THEN
          INSERT INTO ProgRealizzato
          VALUES (Old.CodProgetto, Old.Tipologia);

          FOR Progettista IN (SELECT UserID
                              FROM Partecipante
                              WHERE CodProgetto=Old.CodProgetto)

          LOOP
               INSERT INTO PartecipanteProg
               VALUES (Old.CodProgetto, Progettista);
          END LOOP;

          DELETE FROM Progetto
          WHERE CodProgetto=Old.CodProgetto;
     END IF;
RETURN NEW;
END;

$FineProgetto$ LANGUAGE PLPGSQL;

CREATE TRIGGER FineProgetto
AFTER UPDATE ON Progetto
FOR EACH ROW
EXECUTE PROCEDURE Storico_Progetti();
```

4.3.2 Definizione trigger function Controllo_Luogo

```
CREATE OR REPLACE FUNCTION Controllo_Luogo()
RETURNS TRIGGER AS
$LuogoOccupato$
DECLARE n_riunioni integer;
BEGIN
      SELECT COUNT(*) INTO n_riunioni
      FROM Meeting AS M
      WHERE LOWER(Luogo)=LOWER(New.Luogo) AND
      DataRiunione=New.DataRiunione AND(New.OraInizio>=OraInizio AND
      new.OraInizio<=OraFine);

      IF (n_riunioni>0) THEN
            RAISE 'Errore, il luogo è già occupato da una riunione';
            RETURN NULL;
      END IF;
      RETURN NEW;
END;
$LuogoOccupato$ LANGUAGE PLPGSQL;

CREATE TRIGGER LuogoOccupato
BEFORE INSERT ON Meeting
FOR EACH ROW
EXECUTE PROCEDURE Controllo_Luogo();
```

4.3.3 Definizione trigger function Check_Progetto_Mismatch

```
CREATE OR REPLACE FUNCTION Check_Progetto_Mismatch()
RETURNS TRIGGER AS
$ProgettoMismatch$
BEGIN
      IF new.UserID IN(SELECT UserID
                  FROM Partecipante AS PA NATURAL JOIN Meeting AS ME
                  WHERE ME.CodMeeting <> new.CodMeeting) THEN
            RAISE 'Il progettista può partecipare solo a meeting su
                  progetti a cui partecipa';
            RETURN NULL;
            END IF;
RETURN NEW;
END;
$ProgettoMismatch$ LANGUAGE PLPGSQL;

CREATE TRIGGER ProgettoMismatch
BEFORE INSERT ON CompMeeting
FOR EACH ROW
EXECUTE PROCEDURE Check_Progetto_Mismatch();
```

4.3.4 Definizione trigger function Min_Partecipanti_Meeting

```
CREATE OR REPLACE FUNCTION Min_Partecipanti_Meeting()
RETURNS TRIGGER AS
$ComposizioneMeeting$
DECLARE N_Partecipanti integer;
DECLARE N_Progettisti integer;

BEGIN
    IF Old.Iniziato = false THEN
        SELECT COUNT(UserID) INTO N_Partecipanti
        FROM CompMeeting
        WHERE CodMeeting = New.CodMeeting;

        SELECT COUNT(UserID) INTO N_Progettisti
        FROM Partecipante
        WHERE CodProgetto = New.CodProgetto;

        IF N_Partecipanti < (N_Progettisti::double precision)/3 THEN
           RAISE 'Non è possibile avviare un meeting su un progetto con
                  un numero di partecipanti minore ad 1/3 del numero
                 totale di partecipanti al progetto in questione.';
        RETURN NULL;
        END IF;
    END IF;
RETURN NEW;
END;
$ComposizioneMeeting$ LANGUAGE PLPGSQL;

CREATE TRIGGER Meeting
BEFORE UPDATE OF Iniziato ON Meeting
FOR EACH ROW
EXECUTE PROCEDURE Min_Partecipanti_Meeting();
```

4.3.5 Definizione trigger function Meeting_Senza_PM

```
CREATE OR REPLACE FUNCTION Meeting_Senza_PM()
RETURNS TRIGGER AS
$Meeting_Senza_Manager$
DECLARE Project_Manager integer;

BEGIN
     IF Old.Iniziato = false THEN
          (SELECT UserID INTO Project_Manager
          FROM Partecipante
          WHERE Ruolo = 'Project Manager' AND
          CodProgetto = new.CodProgetto);

          IF Project_Manager NOT IN (SELECT UserID
                                     FROM CompMeeting
                                     WHERE CodMeeting = new.CodMeeting) THEN
               RAISE 'Non è possibile avviare un meeting su un progetto
               senza il relativo project manager.';
          RETURN NULL;
          END IF;
     END IF;
RETURN NEW;
END;

$Meeting_Senza_Manager$ LANGUAGE PLPGSQL;

CREATE TRIGGER Meeting_Senza_Manager
BEFORE UPDATE OF Iniziato ON Meeting
FOR EACH ROW
EXECUTE PROCEDURE Meeting_Senza_PM();
```

4.3.6 Definizione trigger function Check_Meeting_Progetto

```
CREATE OR REPLACE FUNCTION Check_Meeting_Progetto()
RETURNS TRIGGER AS
$MeetingNonPermesso$
DECLARE Progettista_Libero integer;
BEGIN
      FOR Progettista_Libero IN (SELECT UserID
                                 FROM Partecipante
                                 WHERE CodProgetto IS NULL)
      LOOP
            IF New.UserID = Progettista_Libero THEN
                  RAISE 'Impossibile partecipare a questo meeting: il
                  progettista non ha attualmente alcun progetto a carico';
                  RETURN NULL;
            END IF;
      END LOOP;
RETURN NEW;
END;
$MeetingNonPermesso$ LANGUAGE PLPGSQL;

CREATE TRIGGER MeetingNonPermesso
BEFORE INSERT ON CompMeeting
FOR EACH ROW
EXECUTE PROCEDURE Check_Meeting_Progetto();
```

4.3.7 Definizione trigger function Check_Valutazione

```
CREATE OR REPLACE FUNCTION Check_Valutazione()
RETURNS TRIGGER AS
$ValutazioneAziendale$
BEGIN
      IF Old.Valutazione IS NOT NULL THEN
            RAISE 'Un partecipante ha già una valutazione
                  per questo progetto';
            RETURN NULL;
      END IF;
RETURN NEW;
END;
$ValutazioneAziendale$ LANGUAGE PLPGSQL;

CREATE TRIGGER ValutazioneAziendale
BEFORE UPDATE OF Valutazione ON PartecipanteProg
FOR EACH ROW WHEN
EXECUTE PROCEDURE Check_Valutazione();
```

4.3.8 Definizione trigger function Check_PM

```
CREATE OR REPLACE FUNCTION Check_PM()
RETURNS TRIGGER AS
$ProjectManager$
DECLARE npm integer;
BEGIN
      IF UPPER(new.Ruolo) <> 'PROJECT MANAGER' THEN
            RETURN NEW;
      ELSE
            SELECT COUNT(*) INTO npm
            FROM Partecipante
            WHERE UPPER(ruolo) = 'PROJECT MANAGER' AND
            CodProgetto=New.CodProgetto;

            IF (npm > 0) THEN
                  RAISE 'Errore, un progetto non può avere più di un
                  project manager';
                  RETURN NULL;
            END IF;
      RETURN NEW;
      END IF;
END;
$ProjectManager$ LANGUAGE PLPGSQL;

CREATE TRIGGER ProjectManager
BEFORE INSERT ON Partecipante
FOR EACH ROW
EXECUTE PROCEDURE Check_PM();
```

4.3.9 Definizione trigger function Un_Meeting_Alla_Volta

```
CREATE OR REPLACE FUNCTION Un_Meeting_Alla_Volta()
RETURNS TRIGGER AS
$OneMeeting$
BEGIN
      IF New.CodProgetto IN(SELECT CodProgetto
                            FROM Meeting AS M
                            WHERE OraFine IS NULL) THEN
            RAISE 'Un meeting per questo progetto è già calendarizzato';
            RETURN NULL;
      END IF;
RETURN NEW;
END
$OneMeeting$ LANGUAGE PLPGSQL;

CREATE TRIGGER OneMeeting
BEFORE INSERT ON Meeting
FOR EACH ROW
EXECUTE PROCEDURE Un_Meeting_Alla_Volta();
```

4.3.10 Definizione trigger function AmbitoProgetto

```
CREATE FUNCTION AmbitoProgetto()
RETURNS TRIGGER AS
$ProjectTopic$
BEGIN
      INSERT INTO ProgAmbito (codprogetto)
      VALUES (New.CodProgetto);
RETURN NEW;
END;
$ProjectTopic$ LANGUAGE PLPGSQL;

CREATE TRIGGER ProjectTopic
AFTER INSERT ON Progetto
FOR EACH ROW
EXECUTE PROCEDURE AmbitoProgetto();
```

4.3.11 Definizione trigger function InserimentoMeetingAmbito

```
CREATE OR REPLACE FUNCTION InserimentoMeetingAmbito()
RETURNS TRIGGER AS
$AmbitoProgettoMeeting$
BEGIN
      IF New.CodProgetto IN(SELECT CodProgetto
                            FROM ProgAmbito AS M
                            WHERE Nome IS NULL) THEN
            RAISE 'Per il progetto in questione non è stato inserito alcun
                  ambito di appartenenza. Per programmare un meeting,
                  inserisci almeno un ambito';
            RETURN NULL;
      END IF;
RETURN NEW;
END
$AmbitoProgettoMeeting$ LANGUAGE PLPGSQL;

CREATE TRIGGER AmbitoProgettoMeeting
BEFORE INSERT ON Meeting
FOR EACH ROW
EXECUTE PROCEDURE InserimentoMeetingAmbito();
```

4.3.12 Definizione trigger function InserimentoPartecipanteAmbito

```
CREATE OR REPLACE FUNCTION InserimentoPartecipanteAmbito()
RETURNS TRIGGER AS
$AmbitoProgettoPartecipante$
BEGIN
      IF New.CodProgetto IN(SELECT CodProgetto
                            FROM ProgAmbito AS M
                            WHERE Nome IS NULL) THEN
            RAISE 'Per il progetto in questione non è stato inserito alcun
                  ambito di appartenenza. Per programmare un meeting,
                  inserisci almeno un ambito';
            RETURN NULL;
      END IF;
RETURN NEW;
END
$AmbitoProgettoPartecipante$ LANGUAGE PLPGSQL;

CREATE TRIGGER AmbitoProgettoPartecipante
BEFORE INSERT ON Partecipante
FOR EACH ROW
EXECUTE PROCEDURE InserimentoPartecipanteAmbito();
```

4.3.13 Definizione trigger function IncrementaPartecipanti

```
CREATE FUNCTION IncrementaPartecipanti()
RETURNS TRIGGER AS
$NuovoPartecipante$
BEGIN
      UPDATE Progetto
      SET NumeroPartecipanti=NumeroPartecipanti+1
      WHERE CodProgetto=New.CodProgetto;
RETURN NEW;
END;
$NuovoPartecipante$ LANGUAGE PLPGSQL;

CREATE TRIGGER NuovoPartecipante
BEFORE UPDATE OF CodProgetto ON Partecipante
FOR EACH ROW
EXECUTE PROCEDURE IncrementaPartecipanti();
```

4.4 Definizione viste

Di seguito la definizione delle viste implementate.

4.4.1 Definizione vista PartecipantiLiberi

```
CREATE VIEW PartecipantiLiberi (UserID) AS
(SELECT UserID
FROM Partecipante
WHERE CodProgetto IS NULL);
```

4.4.2 Definizione vista ValutazioneMedia

```
CREATE VIEW ValutazioneMedia (UserID, ValutazioneMedia) AS
(SELECT UserID, AVG(Valutazione)
FROM PartecipanteProg
GROUP BY UserID);
```

4.4.3 Definizione vista MeetingImminenti

```
CREATE VIEW MeetingImminenti (CodMeeting, CodProgetto) AS
(SELECT CodMeeting, Codprogetto
FROM Meeting AS M
WHERE ((datariunione-current_date) <= 7));
```

4.4.4 Definizione vista NumProgetti

```
CREATE VIEW NumProgetti (UserID, NumeroProgettiRealizzati) AS
(SELECT UserID, COUNT(CodProg)
FROM PartecipanteProg
GROUP BY UserID);
```

4.4.5 Definizione vista TipologieProgetti

```
CREATE VIEW TipologieProgetti (Tipologia, NumeroProgettiAssociati) AS
(SELECT Tipologia, COUNT(CodProgetto)
FROM Progetto
GROUP BY Tipologia);
```

Capitolo 5

5. Esempio di popolamento del database

5.1 Inserimenti tabella Azienda

```
INSERT INTO Azienda
VALUES
('12345670333', 'Basi Di Dati', 'Napoli'),
('12345670222', 'PostgreSQL', 'Roma'),
('12345670444', 'pgAdmin4', 'Via Claudio');
```

5.2 Inserimenti tabella Privato

```
INSERT INTO Privato
VALUES
('PLLCMN00S17H860L', 'Carmine', 'Paolella');
```

5.3 Inserimenti tabella Società

```
INSERT INTO Società
VALUES
('12345670555', 'SpaceX');
```

5.4 Inserimenti tabella Progetto*

```
INSERT INTO Progetto (Tipologia, Budget, PartIVA, CF)
VALUES
('Ricerca sperimentale', 10000, '12345670333', 'PLLCMN00S17H860L');
INSERT INTO Progetto (Tipologia, Budget, PartIVA, PartitaIVA)
VALUES
('Ricerca industriale', 15000, '12345670333', '12345670555');
```

*Negli inserimenti si può omettere il codice del progetto, chiave primaria dello stesso: per ciascun inserimento verranno automaticamente inseriti degli identificativi progressivi.

5.5 Inserimenti tabella Ambito

```
INSERT INTO Ambito
VALUES
('Elettronica'), ('Medicina'), ('Economia e finanza'), ('Informatica');
```

5.6 Inserimenti tabella ProgAmbito*

```
UPDATE ProgAmbito
SET Nome = 'Informatica'
WHERE CodProgetto = 19;

UPDATE ProgAmbito
SET Nome = 'Elettronica'
WHERE CodProgetto = 20;
```

*Per la tabella ProgAmbito gli inserimenti consistono semplicemente nell'aggiornamento dell'attributo *Nome*, in quanto gli inserimenti di *CodProgetto* sono realizzati in automatico tramite il trigger *AmbitoProgetto*.

5.7 Inserimenti tabella Partecipante*

```
INSERT INTO Partecipante (CF, Nome, Cognome, Ruolo, SalarioMedio,
PartIVA, CodProgetto)
VALUES
('CHHBCG00E47F799U', 'Bianca Giada', 'Chehade', 'Project Manager', 3500,
'12345670333', 19),
('LGRMRA00E06B963U', 'Mario', 'Liguori', 'Responsabile Area Informatica',
2500, '12345670333', 19),
('SMNCRS00E16F839Q', 'Christian', 'Simeone', 'Coordinatore Di Attività',
2500, '12345670333', 19),
('SLVNZE01C56F839S', 'Enza', 'Silvis', 'Coordinatore Di Attività', 2400,
'12345670333', 19),
('ZZAFNC97R05F839R', 'Francesco', 'Zaza', 'Project Manager', 2000,
'12345670222', 20),
('VRRLSS00T45F799E', 'Alessia', 'Verrazzo', 'Responsabile Della
Comunicazione', 2100, '12345670222', 20),
('RLNPQL00D27L259K', 'Pasquale', 'Orlando', 'Responsabile Area
Informatica', 2500, '12345670222', 20);
```

*Negli inserimenti si può omettere l'UserID del partecipante, chiave primaria della tabella: per ciascun inserimento verranno automaticamente inseriti degli identificativi progressivi.

5.8 Inserimenti tabella Meeting

```
INSERT INTO Meeting (DataRiunione, OraInizio, OraFine, Piattaforma,
CodProgetto)
VALUES
('01-08-2021', '9:00', '11:00', 'Skype', 19);
INSERT INTO Meeting (DataRiunione, OraInizio, OraFine, Luogo,
CodProgetto)
VALUES
('04-08-2021', '16:30', '17:30', 'Sala 1', 20);
```

*Negli inserimenti si può omettere il codice del meeting, chiave primaria della tabella: per ciascun inserimento verranno automaticamente inseriti degli identificativi progressivi.

5.9 Inserimenti tabella CompMeeting

```
INSERT INTO CompMeeting
VALUES
(42, 17), (42, 18), (42, 19), (42, 20), (43, 21), (43, 22), (43, 23);
```

5.10 Inserimenti tabella ProgRealizzato

Non è necessario inserire elementi nella tabella ProgRealizzato; gli inserimenti avverranno in automatico grazie al trigger *FineProgetto*.

5.11 Inserimenti tabella PartecipanteProg

Non è necessario inserire elementi nella tabella PartecipanteProg; gli inserimenti avverranno in automatico grazie al trigger *FineProgetto*; è però possibile aggiornare l'attributo *Valutazione*.