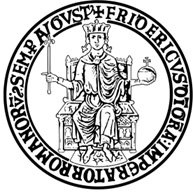
Università degli Studi di Napoli Federico II

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*

*Questa pagina è stata lasciata intenzionalmente bianca.*

**Indice**

[1. Descrizione del progetto 5](#_Toc70011591)

[1.1 Descrizione sintetica 5](#_Toc70011592)

[2. Progettazione concettuale 6](#_Toc70011593)

[2.1 Class diagram 6](#_Toc70011594)

[2.2 Ristrutturazione del class diagram 6](#_Toc70011595)

[2.2.1 Analisi delle ridondanze 6](#_Toc70011596)

[2.2.2 Analisi degli identificativi 7](#_Toc70011597)

[2.2.3 Rimozione degli attributi multipli 7](#_Toc70011598)

[2.2.4 Rimozione delle gerarchie di specializzazione 7](#_Toc70011599)

[2.3 Class diagram ristrutturato 7](#_Toc70011600)

[2.4 Dizionario dei dati 7](#_Toc70011601)

[2.4.1 Dizionario delle classi 7](#_Toc70011602)

[2.4.2 Dizionario delle associazioni 10](#_Toc70011603)

[2.4.3 Dizionario dei vincoli 12](#_Toc70011604)

[3. Progettazione logica 12](#_Toc70011605)

[3.1 Traduzione in schemi relazionali 13](#_Toc70011606)

[3.1.1 Traduzione delle associazioni 13](#_Toc70011607)

[3.2 Schema Logico 14](#_Toc70011608)

[4. Progettazione fisica 15](#_Toc70011609)

[4.1 Definizione delle tabelle 15](#_Toc70011610)

[4.1.1 Definizione della tabella Azienda 15](#_Toc70011611)

[4.1.2 Definizione della tabella Privato 15](#_Toc70011612)

[4.1.3 Definizione della tabella Società 15](#_Toc70011613)

[4.1.4 Definizione della tabella Progetto 15](#_Toc70011614)

[4.1.5 Definizione della tabella Partecipante 15](#_Toc70011615)

[4.1.6 Definizione della tabella ProgRealizzato 16](#_Toc70011616)

[4.1.7 Definizione della tabella PartecipanteProg 16](#_Toc70011617)

[4.1.8 Definizione della tabella Meeting 16](#_Toc70011618)

[4.1.9 Definizione della tabella CompMeeting 16](#_Toc70011619)

[4.1.10 Definizione della tabella Ambito 16](#_Toc70011620)

[4.1.11 Definizione della tabella ProgAmbito 16](#_Toc70011621)

[4.2 Definizione vincoli di dominio 17](#_Toc70011622)

[4.3 Definizione trigger functions 17](#_Toc70011623)

[4.3.1 Definizione trigger function Storico\_Progetti 17](#_Toc70011624)

[4.3.2 Definizione trigger function Controllo\_Partecipante\_Meeting 17](#_Toc70011625)

[4.3.3 Definizione trigger function Controllo\_Luogo 18](#_Toc70011626)

[4.3.4 Definizione trigger function Check\_Progetto\_Mismatch 18](#_Toc70011627)

[4.3.5 Definizione trigger function Min\_Partecipanti\_Meeting 18](#_Toc70011628)

[4.3.6 Definizione trigger function Meeting\_Senza\_PM 19](#_Toc70011629)

[4.3.7 Definizione trigger function Check\_Meeting\_Progetto 20](#_Toc70011630)

[4.3.8 Definizione trigger function Check\_Valutazione 20](#_Toc70011631)

[4.3.9 Definizione trigger function Check\_PM 20](#_Toc70011632)

**Capitolo 1**

# **1. Descrizione del progetto**

## **1.1 Descrizione sintetica**

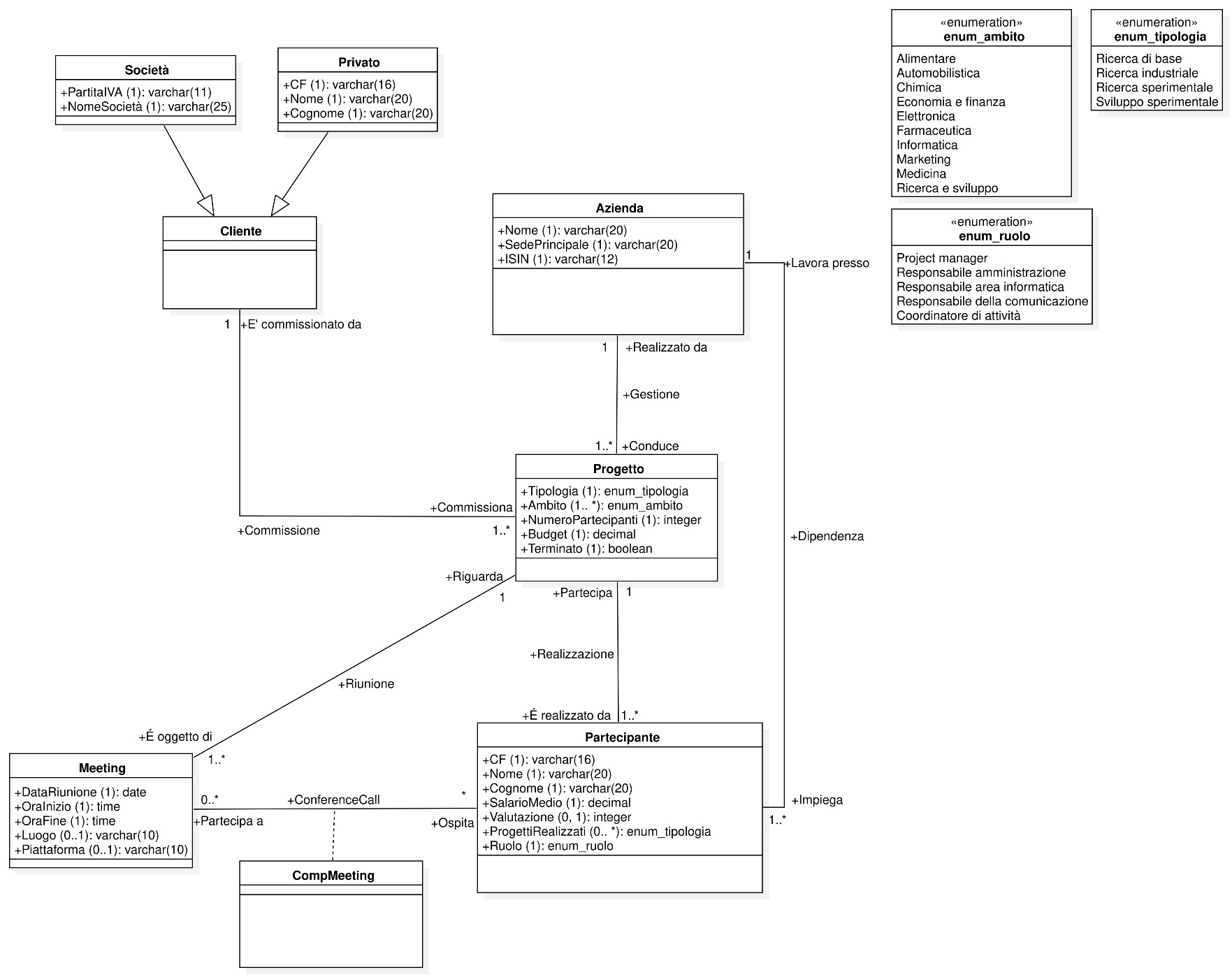
Si svilupperà ed implementerà una base di dati relazionale utile alla gestione di progetti in un’azienda. Tale database prevede la possibilità di tenere traccia dei partecipanti a ciascun progetto, identificando il ruolo per ognuno di essi (per ogni progetto ci sarà un solo project manager). Ad ogni 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.

### **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;** la cardinalità molti a molti della nuova associazione tra **Progetto** e **Ambito** sarà risolta con l’implementazione della classe **ProgAmbito**

Lo stesso procedimento è da eseguire per l’attributo multiplo **ProgettiRealizzati** della classe **Partecipante**; si procede alla creazione delle classi **ProgRealizzato e PartecipanteProg** (utile data la cardinalità molti a molti dell’associazione tra **Partecipante** e **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.

## **2.3 Class diagram ristrutturato**

//INSERISCI CLASS DIAGRAM RISTRUTTURATO

## **2.4 Dizionario dei dati**

### **2.4.1 Dizionario delle classi**

|  |  |  |
| --- | --- | --- |
| Classe | Descrizione | Attributi |
| Azienda | Descrittore dell’azienda che realizza un progetto. | **ISIN** (varchar): chiave primaria. Codice identificativo univoco per le aziende.  **Nome** (varchar): 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): attributo booleano che indica se il progetto in questione è giunto al termine o è ancora in esecuzione. |
| Partecipante | Descrive ciascun partecipante al progetto. | **UserID** (integer): chiave surrogata. Identificativo numerico di ciascun partecipante al progetto.  **Email** (varchar, opzionale): specifica l’indirizzo di posta elettronica delpartecipante.  **Password** (varchar, opzionale): indica la sequenza di caratteri alfanumerici necessaria per accedere all’email.  **CF** (varchar): codice fiscale del partecipante, identifica anch’esso in modo univoco le persone fisiche che partecipano al progetto.  **Nome** (varchar): nome del partecipante.  **Cognome** (varchar): cognome del partecipante.  **Ruolo** (enumerazione): specifica il ruolodel dipendente nel progetto a cui sta lavorando.  **SalarioMedio** (decimal): indica il guadagno medio di un partecipante.  **Valutazione** (integer, opzionale): fornisce una valutazione numerica di un partecipante; essa sarà influenzata dalla partecipazione del dipendente ai progetti e ai relativi meeting. La valutazione è numerica e compresa tra 1e 5. |

*Continua nella pagina seguente*

|  |  |  |
| --- | --- | --- |
| Classe | Descrizione | Attributi |
| Società | Descrittore della società che commissiona il progetto. | **PartitaIVA** (varchar):sequenza di cifre che identifica univocamente un soggetto che esercita un’attività.  **NomeSocietà** (varchar): specifica il nome della società. |
| Privato | Descrittore del privato che commissiona il progetto. | **CF** (varchar): identifica in modo univoco le persone fisiche.  **Nome** (varchar): nome del privato.  **Cognome** (varchar): cognome del privato. |
| Meeting | Descrittore delle riunioni sia fisiche che telematiche. | **CodMeeting** (integer): sequenza numerica che permette di identificare univoamente 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. |
| Ambito | Descrittore dell’ambitodi un progetto. | **Nome** (enumerazione): indica in che ambito è indirizzato il progetto in questione. |
| ProgRealizzato | Descrittore dei progetti realizzati in passato dai progettisti. | **CodProg** (integer): chiave surrogata; codice identificativo di un progetto.  **Tipologia** (enumerazione): indica di che tipo è il progetto. |

### **2.4.2 Dizionario delle associazioni**

|  |  |  |
| --- | --- | --- |
| Nome | Descrizione | Classi Coinvolte |
| Dipendenza | Esprime l’appartenenza di un partecipante ad una azienda. | **Partecipante[1.. \*]** ruolo **Lavora presso**: indica in quale azienda lavora un partecipante.  **Azienda[1]** ruolo **Impiega**: indica l’azienda presso la quale il partecipante lavora. |
| Gestione | Esprime 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.  **Partecipanti[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 |

*Continua nella pagina seguente.*

|  |  |  |
| --- | --- | --- |
| Nome | Descrizione | Classi Coinvolte |
| 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 a quale meeting si riferisce la composizione.  **CompMeeting[\*]** ruolo **Compone**: indica quale meeting ha le caratteristiche in questione. |

### **2.4.3 Dizionario dei vincoli**

**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** (ISIN, Nome, SedePrincipale). |

Chiavi esterne: nessuna.

|  |
| --- |
| **Privato** (CF, Nome, Cognome) |

Chiavi esterne: nessuna.

|  |
| --- |
| **Società** (PartitaIVA, NomeSocietà) |

Chiavi esterne: nessuna.

|  |
| --- |
| **Progetto** (CodProgetto, Tipologia, NumeroPartecipanti, Budget, ISIN, CF, PartitaIVA). |

Chiavi esterne: ISIN → Azienda.ISIN; CF → Privato.CF; PartitaIVA → Società.PartitaIVA.

|  |
| --- |
| **Partecipante** (UserID, Email, Pw, CF, Nome, Cognome, Ruolo, SalarioMedio, Valutazione, CodProgetto, ISIN). |

Chiavi esterne: CodProgetto → Progetto.CodProgetto; ISIN → Azienda.ISIN.

|  |
| --- |
| **ProgRealizzato** (CodProg, Tipologia) |

Chiavi esterne: nessuna.

|  |
| --- |
| **PartecipanteProg** (CodProg, UserID). |

Chiavi esterne: CodProg → ProgRealizzato.CodProg; UserID → Partecipante.UserID.

|  |
| --- |
| **Meeting** (CodMeeting, DataRiunione, OraInizio, OraFine, Piattaforma, Luogo). |

Chiavi esterne: nessuna.

|  |
| --- |
| **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** → **Progetto** |
| ProgettoPrecedente | Chiave esterna in **PartecipanteProg** → **ProgRealizzato** |
| ConferenceCall | Chiave esterna in **CompMeeting** → **Partecipante** |
| Composizione | Chiave esterna in **CompMeeting** → **Meeting** |

## **3.2 Schema Logico**

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

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

**Capitolo 4**

# **4. Progettazione fisica**

## **4.1 Definizione delle tabelle**

### **4.1.1 Definizione della tabella Azienda**

// TABELLA

CREATE TABLE Azienda (

ISIN varchar(12) PRIMARY KEY,

Nome varchar(20) NOT NULL,

SedePrincipale varchar(12) NOT NULL

);

### **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

);

### 

### **4.1.4 Definizione della tabella Progetto**

// TABELLA

CREATE TABLE Progetto (

CodProgetto integer PRIMARY KEY,

Tipologia enum\_tipologia NOT NULL,

NumeroPartecipanti integer NOT NULL,

Budget decimal NOT NULL,

ISIN varchar(12) NOT NULL REFERENCES Azienda(ISIN),

CF varchar(16) REFERENCES Privato(CF),

PartitaIVA varchar(11) REFERENCES Società(PartitaIVA),

);

// 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 Progetto\_NumeroPartecipanti\_Check CHECK (numeropartecipanti > 0 AND numeropartecipanti <= 30);

### 

### **4.1.5 Definizione della tabella Partecipante**

// TABELLA

CREATE TABLE Partecipante (

UserID integer PRIMARY KEY,

Email varchar(30),

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,

Valutazione integer,

CodProgetto integer NOT NULL REFERENCES Progetto(CodProgetto),

ISIN varchar(12) NOT NULL REFERENCES Azienda(ISIN)

);

// 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)),

/// manca constraint check\_password

ADD CONSTRAINT Unico\_CF (UNIQUE CF),

ADD CONSTRAINT Unique\_Account UNIQUE (Email, CF);

### **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 (

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,

OraFine time,

Piattaforma varchar(10),

Luogo varchar(10),

CodProgetto integer NOT NULL REFERENCES Progetto(CodProgetto)

);

// 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);

### **4.1.9 Definizione della tabella CompMeeting**

CREATE TABLE CompMeeting (

CodMeeting integer NOT NULL REFERENCES Meeting(CodMeeting),

UserID integer NOT NULL REFERENCES Partecipante(UserID)

);

### 

### **4.1.10 Definizione della tabella Ambito**

CREATE TABLE Ambito (

Nome enum\_ambito PRIMARY KEY

);

### 

### **4.1.11 Definizione della tabella ProgAmbito**

CREATE TABLE ProgAmbito (

CodProgetto integer NOT NULL REFERENCES Progetto(CodProgetto),

Nome enum\_ambito NOT NULL REFERENCES Ambito(Nome)

);

## 

## **4.2 Definizione vincoli di dominio**

## **4.3 Definizione trigger functions**

### 

### **4.3.1 Definizione trigger function Storico\_Progetti**

CREATE OR REPLACE FUNCTION Storico\_Progetti()

RETURNS TRIGGER AS

$FineProgetto$

BEGIN

IF new.Terminato = TRUE THEN

INSERT INTO ProgRealizzato

VALUES (old.CodProgetto, old.Tipologia);

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\_Partecipante\_Meeting**

CREATE OR REPLACE FUNCTION Controllo\_Partecipante\_Meeting()

RETURNS TRIGGER AS

$PartecipanteMeeting$

DECLARE Partecipanti REFCURSOR;

DECLARE Partecipante\_Occupato integer;

BEGIN

OPEN Partecipanti FOR (SELECT UserID

FROM CompMeeting AS C NATURAL JOIN Meeting AS M

WHERE OraInizio IS NOT NULL AND OraFine IS NULL);

LOOP

EXIT WHEN NOT FOUND;

FETCH Partecipanti INTO Partecipante\_Occupato;

IF new.UserID = Partecipante\_Occupato THEN

RAISE 'Un dipendente può partecipare a un solo meeting alla volta.';

RETURN NULL;

END IF;

END LOOP;

CLOSE Partecipanti;

RETURN NEW;

END;

$PartecipanteMeeting$ LANGUAGE PLPGSQL;

CREATE TRIGGER PartecipanteMeeting

BEFORE INSERT ON CompMeeting

FOR EACH ROW

EXECUTE PROCEDURE Controllo\_Partecipante\_Meeting();

### 

### **4.3.3 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 Luogo = 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();

### **4.3.4 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 JOIN Meeting AS ME ON PA.CodProgetto=ME.CodProgetto WHERE new.CodMeeting IN (SELECT CodMeeting FROM Meeting

WHERE CodProgetto <> PA.CodProgetto)) 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();

### **4.3.5 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.OraInizio IS NULL THEN

SELECT COUNT(UserID) INTO N\_Partecipanti

FROM CompMeeting

WHERE CodMeeting = new.CodMeeting;

SELECT COUNT(UserID) INTO N\_Progettisti

FROM PartecipanteProg

WHERE CodProg = new.CodMeeting;

IF N\_Partecipanti < (1/3 \* (N\_Progettisti)) 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 Orainizio ON Meeting

FOR EACH ROW

EXECUTE PROCEDURE Min\_Partecipanti\_Meeting();

### **4.3.6 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.OraInizio IS NULL 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 OraInizio ON Meeting

FOR EACH ROW

EXECUTE PROCEDURE Meeting\_Senza\_PM();

### **4.3.7 Definizione trigger function Check\_Meeting\_Progetto**

CREATE OR REPLACE FUNCTION Check\_Meeting\_Progetto()

RETURNS TRIGGER AS

$MeetingNonPermesso$

DECLARE utenti\_liberi REFCURSOR;

DECLARE progettista\_libero integer;

BEGIN

OPEN utenti\_liberi FOR (SELECT UserID FROM Partecipante WHERE CodProgetto IS NULL);

LOOP

EXIT WHEN NOT FOUND;

FETCH utenti\_liberi INTO progettista\_libero;

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;

RETURN NEW;

END LOOP;

CLOSE utenti\_liberi;

END;

$MeetingNonPermesso$ LANGUAGE PLPGSQL;

CREATE TRIGGER MeetingNonPermesso

BEFORE INSERT ON CompMeeting

FOR EACH ROW

EXECUTE PROCEDURE Check\_Meeting\_Progetto();

### **4.3.8 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 non può avere più di una valutazione per progetto';

RETURN NULL;

ELSE

UPDATE Partecipante

SET Valutazione = new.Valutazione;

END IF;

RETURN NEW;

END;

$ValutazioneAziendale$ LANGUAGE PLPGSQL;

CREATE TRIGGER ValutazioneAziendale

BEFORE UPDATE OF Valutazione ON Partecipante

FOR EACH ROW

EXECUTE PROCEDURE Check\_Valutazione();

### **4.3.9 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();