



# PROGETTAZIONE E SVILUPPO DI UNA BASE DI DATI PER LA GESTIONE DI PERSONALE E PROGETTI ALL'INTERNO DI UN'AZIENDA

Marco Menale  
N86002681  
20/12/2023

# Indice

<b>1</b>	<b>Requisiti identificati</b>	<b>3</b>
<b>2</b>	<b>Progettazione concettuale</b>	<b>4</b>
2.1	Class Diagram . . . . .	4
2.2	Analisi della ristrutturazione del Class Diagram . . . . .	5
2.2.1	Analisi delle ridondanze . . . . .	5
2.2.2	Analisi degli identificativi . . . . .	5
2.2.3	Rimozione degli attributi multipli . . . . .	5
2.2.4	Rimozione degli attributi composti . . . . .	5
2.2.5	Partizione/Accorpamento delle associazioni . . . . .	5
2.2.6	Rimozione delle gerarchie . . . . .	5
2.2.7	Diagramma ristrutturato . . . . .	5
<b>3</b>	<b>Schema logico</b>	<b>9</b>
3.1	<b>Progettazione Logica</b> . . . . .	9
3.1.1	Schema Logico . . . . .	9
<b>4</b>	<b>Schema fisico</b>	<b>10</b>
4.1	Definizioni Tabelle . . . . .	10
4.1.1	Tabella Azienda . . . . .	10
4.1.2	Tabella Impiegato . . . . .	10
4.1.3	Tabella Laboratorio . . . . .	11
4.1.4	Tabella Progetto . . . . .	11
4.2	Triggers e Functions per la gestione degli Impiegati . . . . .	12
4.2.1	Trigger StipValido . . . . .	12
4.2.2	Trigger GradoTipoValido . . . . .	12
4.2.3	Trigger CheckImpiegati . . . . .	13
4.2.4	Trigger PromozioneImpiegato . . . . .	14
4.2.5	Trigger CorreggiGrado . . . . .	15
4.2.6	Trigger FixStipendio . . . . .	16
4.2.7	Trigger SetStipendio . . . . .	17
4.2.8	Function getslab . . . . .	18
4.2.9	Function getimpiegato . . . . .	19
4.2.10	Function getprog . . . . .	19
4.3	Triggers e Functions per la gestione dei Laboratori . . . . .	21
4.3.1	Trigger CheckLaboratorio . . . . .	21
4.3.2	Trigger LaboratorioAperto . . . . .	21
4.3.3	Trigger ChiusuraLaboratorio . . . . .	22
4.3.4	Function getprogfromlab . . . . .	23
4.4	Triggers e Functions per la gestione dei Progetti . . . . .	23
4.4.1	Function CheckProgetto . . . . .	23
4.4.2	Trigger ProgettoAttivo . . . . .	24
4.4.3	Trigger ChiusuraProgettoImpiegato . . . . .	25
4.4.4	Trigger ChiusuraProgLaboratorio . . . . .	26
4.5	Analisi scatti di carriera degli Impiegati . . . . .	27

# Capitolo 1

## Requisiti identificati

Si progetterà ed implementerà una base di dati che possa essere d'ausilio nella gestione di un'azienda. Il database conterrà informazioni sull'azienda, sugli impiegati che la popolano e sul loro lavoro, che prevede l'utilizzo di laboratori per lo sviluppo di progetti. Ciascun impiegato può essere un dipendente o un dirigente. I dipendenti si dividono in tre gruppi: junior, middle e senior. I dipendenti junior sono coloro che lavorano in azienda da meno di tre anni, dipendenti middle sono coloro che lavorano in azienda da più di tre anni ma da meno di sette, i dipendenti senior sono coloro che lavorano in azienda da più di sette anni. Ciascun dipendente può diventare dirigente se prova di possederne le capacità, indipendentemente dai suoi anni di servizio. I progetti avranno un referente scientifico, che sarà un dipendente senior dell'azienda, e un responsabile, che sarà un dirigente. Al massimo tre laboratori possono occuparsi di un progetto singolo. Ciascun laboratorio avrà un responsabile scientifico, che sarà un dipendente senior dell'azienda.

Sono state identificate le seguenti entità:

1. Azienda: Identifica le singole aziende.
2. Impiegato: Generalizzazione di un Dipendente o un Dirigente.
3. Dipendente: Specializzazione di un Impiegato.
4. Dirigente: Specializzazione di un Impiegato.
5. Laboratorio: Identifica i Laboratori.
6. Progetto: Identifica i progetti di cui si occupa un'azienda.

## Capitolo 2

# Progettazione concettuale

### 2.1 Class Diagram

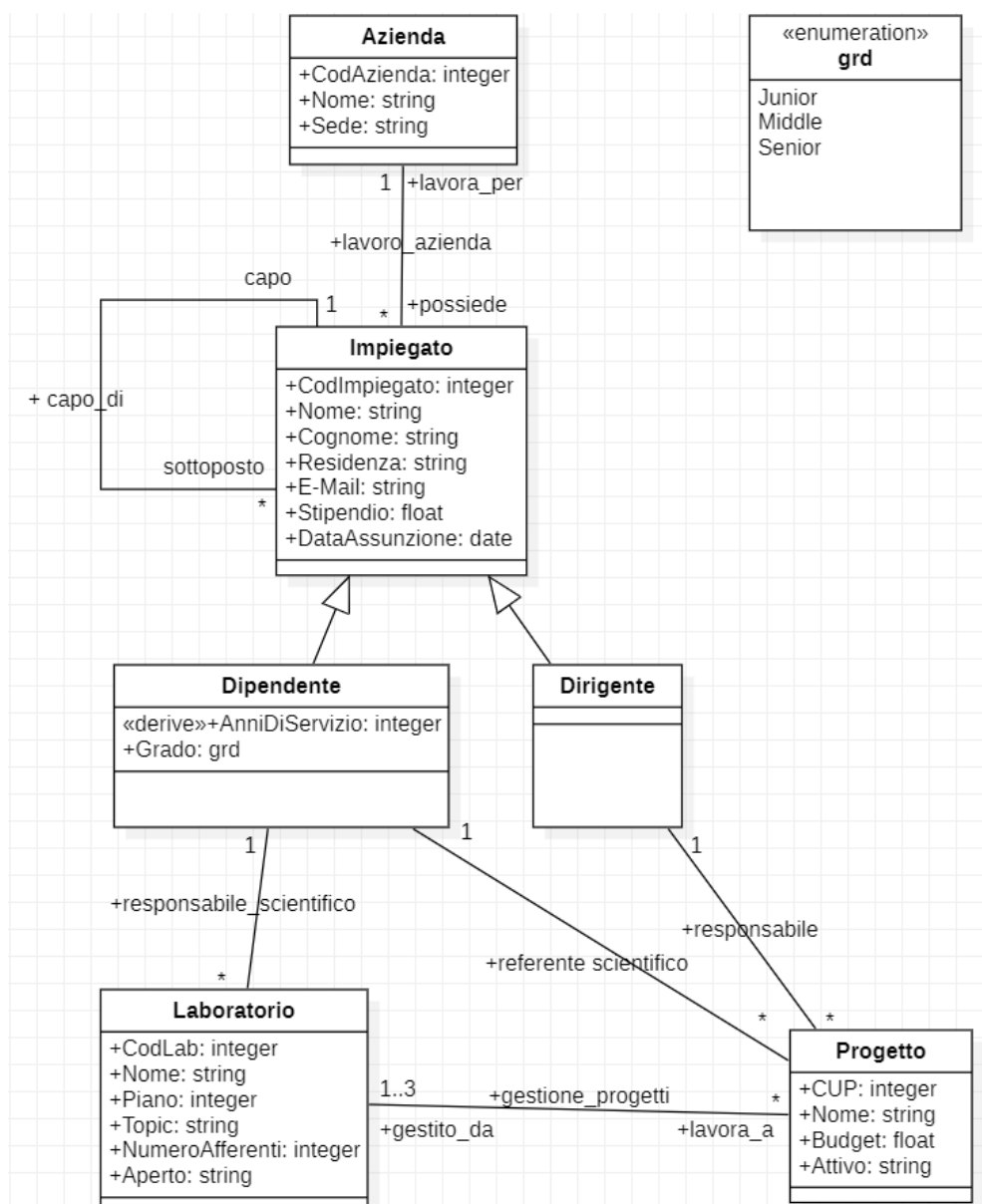


Figura 2.1: diagramma delle classi

## 2.2 Analisi della ristrutturazione del Class Diagram

In questo capitolo, verranno effettuate delle modifiche al Class Diagram in modo da renderlo più adatto ad una traduzione al modello logico.

### 2.2.1 Analisi delle ridondanze

L'attributo *AnniDiServizio* presente nella sottoclasse **Dipendente** è derivabile dall'attributo *DataDiAssunzione*. È possibile ottenerlo sottraendo l'anno corrente all'anno della data dell'assunzione dell'impiegato. Per semplicità, rimuoveremo l'attributo *AnniDiServizio*.

### 2.2.2 Analisi degli identificativi

In questa sottosezione verranno analizzati gli identificativi delle singole entità.

1. Per quanto riguarda l'entità **Azienda**, l'attributo *CodAzienda* sarà la sua chiave primaria e servirà a identificare singolarmente le varie aziende.
2. Ogni **Impiegato** sarà identificato dal suo *CodImpiegato*.
3. Ogni **Laboratorio** avrà un *CodLab* che servirà ad identificarlo univocamente.
4. Il codice *CUP* identificherà ogni **Progetto**.

### 2.2.3 Rimozione degli attributi multipli

Dato che il numero dei Laboratori che si possono occupare di un singolo progetto è compreso tra 1 e 3, nella classe **Progetto** saranno presenti tre attributi, *Lab 1*, *Lab 2* e *Lab 3* che fungeranno da chiavi esterne, tre riferimenti alla classe **Laboratorio**. Il primo dei tre dovrà necessariamente non essere nullo considerando che almeno un Laboratorio dovrà essere associato a un Progetto, mentre i restanti due potranno essere anche nulli.

### 2.2.4 Rimozione degli attributi composti

Nel Class Diagram non sono presenti attributi composti.

### 2.2.5 Partizione/Accorpamento delle associazioni

### 2.2.6 Rimozione delle gerarchie

Nel diagramma è presente la specializzazione della classe **Impiegato** nelle due sottoclassi **Dipendente** e **Dirigente**. Essa è totale e disjoint. Si è scelto di accorpare le due entità figlie nell'entità padre, creando in tal modo un'unica classe **Impiegato**. L'attributo **Tipo** può assumere solo i valori *Dipendente* e *Dirigente*, se il suo valore è *Dirigente* allora gli attributi *AnniDiServizio* e *Grado* saranno nulli, visto che per i **Dirigenti** sono irrilevanti.

### 2.2.7 Diagramma ristrutturato

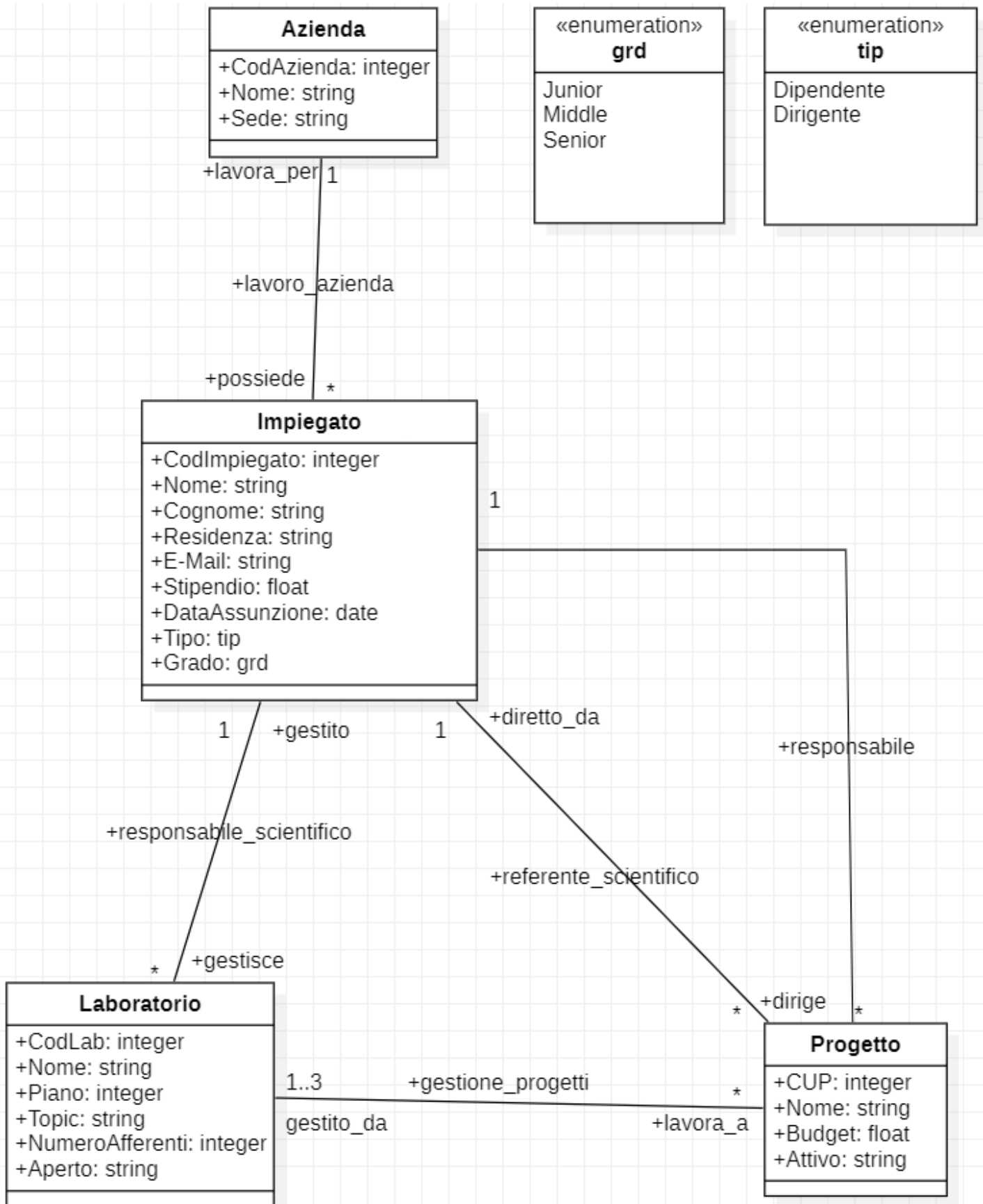


Figura 2.2: diagramma ristrutturato

Nome	Descrizione	Attributi
Azienda	Descrittore di ciascuna azienda presente nel database.	CodAzienda( <i>integer</i> ) Chiave primaria, identifica univocamente ciascuna azienda; Nome( <i>string</i> ) Nome dell'azienda; Sede( <i>string</i> ) Città in cui si trova l'azienda.
Impiegato	Descrittore di ciascun impiegato presente nel database.	CodImpiegato( <i>integer</i> ) Chiave primaria, identifica univocamente ciascun lavoratore; Nome( <i>string</i> ) Nome dell'impiegato; Cognome( <i>string</i> ) Cognome dell'impiegato; Residenza( <i>string</i> ) Città in cui risiede l'impiegato; E-Mail( <i>string</i> ) L'indirizzo E-Mail dell'impiegato; Stipendio( <i>float</i> ) Lo stipendio dell'impiegato; DataAssunzione( <i>date</i> ) Data in cui l'impiegato è stato assunto dall'azienda per cui lavora; Tipo( <i>tip</i> ) Il tipo dell'impiegato; Grado( <i>grd</i> ) Grado dell'impiegato, si può ricavare dall'attributo AnniDiServizio.
Laboratorio	Descrittore di ciascun laboratorio presente nel database.	CodLab( <i>integer</i> ) Chiave primaria, identifica univocamente ciascun laboratorio; Nome( <i>string</i> ) Nome del laboratorio; Piano( <i>integer</i> ) Piano su cui si trova il laboratorio; Topic( <i>string</i> ) Argomento specifico di cui si occupa il laboratorio; NumeroAfferenti( <i>integer</i> ) Numero di persone che lavorano nel laboratorio; Aperto( <i>string</i> ) Un carattere che indica se il Laboratorio è attualmente aperto o chiuso;
Progetto	Descrittore di ciascun progetto presente nel database.	CUP( <i>integer</i> ) Chiave primaria, identifica univocamente ciascun progetto; Nome( <i>string</i> ) Nome del progetto; Budget( <i>float</i> ) Budget utilizzato per il progetto; Attivo( <i>string</i> ) Un carattere che indica se il progetto è attivo o no;

Tabella 2.1: Dizionario delle classi

Nome	Descrizione	Classi coinvolte
lavoro-azienda	Esprime la relazione tra un impiegato e l'azienda per cui lavora.	Azienda [1..1] ruolo lavora-per: indica l'azienda per cui lavora l'impiegato; Impiegato [0..*] ruolo possiede: indica l'impiegato che lavora in Azienda.
responsabile-scientifico	Esprime la relazione tra un impiegato e il laboratorio di cui è responsabile scientifico.	Impiegato [1..1] ruolo gestito: indica l'impiegato che gestisce il laboratorio; Laboratorio [0..*] ruolo gestisce: indica il laboratorio gestito dall'impiegato.
referente-scientifico	Esprime la relazione tra l'impiegato e il progetto di cui è referente scientifico.	Impiegato [1..1] ruolo diretto-da: indica l'impiegato che gestisce il progetto Progetto [0..*] ruolo dirige: indica il progetto di cui si occupa l'impiegato
responsabile	Esprime la relazione tra l'impiegato e il progetto di cui è responsabile.	Impiegato [1..1] ruolo gestito: indica l'impiegato responsabile del progetto; Progetto [0..*] ruolo gestisce: indica il progetto di cui l'impiegato è responsabile.
gestione-progetti	Esprime la relazione tra un laboratori e progetti.	Laboratorio [1..3] ruolo gestito-da: indica laboratorio che si occupa del progetto; Progetto [0..*] ruolo lavora-a: indica il progetto di cui si occupa il laboratorio.
capo-di	Esprime la relazione tra due impiegato di cui uno è il capo e l'altro è il sottoposto	Impiegato [1..1] ruolo capo: indica il capo di un determinato impiegato; Impiegato [0..*] ruolo sottoposto: indica il sottoposto di un determinato impiegato.

Tabella 2.2: Dizionario delle associazioni



## Capitolo 3

# Schema logico

### 3.1 Progettazione Logica

In questo capitolo verrà trattata la fase successiva della progettazione, si scenderà ad un livello di astrazione più basso. Lo schema concettuale verrà tradotto in schema logico, dipendente dal tipo dei dati prescelto cioè quello relazionale puro. Le chiavi primarie saranno sottolineate una singola volta mentre le chiavi esterne avranno una doppia sottolineatura

#### 3.1.1 Schema Logico

**Azienda** (CodAzienda, Nome, Sede)

Chiavi esterne: nessuna

**Impiegato** (CodImpiegato, Nome, Cognome, Residenza, E-Mail, DataAssunzione, Tipo, Stipendio, Grado, Capo CodAzienda)

**Chiavi esterne:** CodAzienda → Azienda.CodAzienda

Capo → Impiegato.CodImpiegato

**Laboratorio** (CodLab, Nome, Piano, Topic, NumeroAfferenti, Aperto, ResponsabileScientifico)

**Chiavi esterne:** ResponsabileScientifico → Impiegato.CodImpiegato

**Progetto** (CUP, Nome, Budget, Attivo, ReferenteScientifico, Responsabile, Lab1, Lab2, Lab3)

**Chiavi esterne:** ReferenteScientifico → Impiegato.CodImpiegato

Responsabile → Impiegato.CodImpiegato

Lab1 → Laboratorio.CodLab

Lab2 → Laboratorio.CodLab

Lab3 → Laboratorio.CodLab

# Capitolo 4

## Schema fisico

### 4.1 Definizioni Tabelle

#### 4.1.1 Tabella Azienda

```
01 | CREATE TABLE Azienda(  
02 |   CodAzienda INTEGER,  
03 |   Nome VARCHAR(20) ,  
04 |   Sede VARCHAR(20) ,  
05 |  
06 |   CONSTRAINT PKA1 PRIMARY KEY (CodAzienda));
```

#### 4.1.2 Tabella Impiegato

```
01 | CREATE TABLE Impiegato(  
02 |   CodImpiegato INTEGER,  
03 |   Nome VARCHAR(20) NOT NULL,  
04 |   Cognome VARCHAR(20) NOT NULL,  
05 |   Residenza VARCHAR(20) NOT NULL,  
06 |   E-Mail VARCHAR(30) NOT NULL,  
07 |   Stipendio FLOAT NOT NULL,  
08 |   DataAssunzione DATE NOT NULL,  
09 |   Tipo VARCHAR(15) NOT NULL CHECK(Tipo IN ( 'Dipendente', 'Dirigente' )),  
10 |   Grado VARCHAR(15) NOT NULL CHECK(Grado IN ( 'Junior', 'Middle', 'Senior' )),  
11 |   CodAzienda INTEGER NOT NULL,  
12 |   Capo INTEGER,  
13 |  
14 |   CONSTRAINT PKI1 PRIMARY KEY (CodImpiegato) ,  
15 |   CONSTRAINT UNQI1 UNIQUE (E-Mail) ,  
16 |   CONSTRAINT FK11 FOREIGN KEY(CodAzienda) REFERENCES Azienda(CodAzienda)  
    ON DELETE CASCADE,  
17 |   CONSTRAINT FK12 FOREIGN KEY(Capo) REFERENCES Impiegato(CodImpiegato));
```

#### 4.1.3 Tabella Laboratorio

```
01 | CREATE TABLE Laboratorio(  
02 |   CodLab INTEGER,  
03 |   Nome VARCHAR(20) NOT NULL,  
04 |   Piano INTEGER NOT NULL,  
05 |   Topic VARCHAR(30) NOT NULL,  
06 |   NumeroAfferenti INTEGER NOT NULL,  
07 |   Aperto CHAR(1) NOT NULL CHECK(Aperto IN ('S', 'N')) ,  
08 |   ResponsabileScientifico INTEGER,  
09 |  
10 |   CONSTRAINT PKL1 PRIMARY KEY (CodLab) ,  
11 |   CONSTRAINT FKL1 FOREIGN KEY(ResponsabileScientifico) REFERENCES  
    Impiegato(CodImpiegato));
```

#### 4.1.4 Tabella Progetto

```
01 | CREATE TABLE Progetto(  
02 |   CUP INTEGER,  
03 |   Nome VARCHAR(20) ,  
04 |   Budget FLOAT,  
05 |   Attivo CHAR(1) CHECK(Attivo IN ('S', 'N'))  
06 |   ReferenteScientifico INTEGER,  
07 |   Responsabile INTEGER,  
08 |   Lab1 INTEGER,  
09 |   Lab2 INTEGER,  
10 |   Lab3 INTEGER,  
11 |  
12 |   CONSTRAINT PKP1 PRIMARY KEY (CUP) ,  
13 |   CONSTRAINT UNQP1 UNIQUE (Nome) ,  
14 |   CONSTRAINT FKP1 FOREIGN KEY(ReferenteScientifico) REFERENCES Impiegato(  
    CodImpiegato) ,  
15 |   CONSTRAINT FKP2 FOREIGN KEY(Responsabile) REFERENCES Impiegato(  
    CodImpiegato) ,  
16 |   CONSTRAINT FKP3 FOREIGN KEY(Lab1) REFERENCES Laboratorio(CodLab) ,  
17 |   CONSTRAINT FKP4 FOREIGN KEY(Lab2) REFERENCES Laboratorio(CodLab) ,  
18 |   CONSTRAINT FKP5 FOREIGN KEY(Lab2) REFERENCES Laboratorio(CodLab));
```

## 4.2 Triggers e Functions per la gestione degli Impiegati

### 4.2.1 Trigger StipValido

Un Impiegato che ha il valore Capo diverso da NULL non può avere uno stipendio che sia maggiore di quello del suo capo. Il seguente trigger agirà prima dell'inserimento nella Tabella *Impiegato* e si occuperà di verificare che la condizione sia verificata. Se il capo dovesse avere uno stipendio minore di quello del suo sottoposto, allora verrà visualizzato un messaggio di errore e l'inserimento di quel dipendente nel database sarà annullato.

```
01 | CREATE OR REPLACE FUNCTION StipValido() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |     Stip_Sottoposto Impiegato.Stipendio%Type;
05 |     Stip_Capo Impiegato.Stipendio%Type;
06 | BEGIN
07 |     SELECT I1.Stipendio, I2.Stipendio INTO Stip_Sottoposto, Stip_Capo
08 |     FROM Impiegato AS I1, Impiegato AS I2
09 |     WHERE I1.CodImpiegato = CodImp AND I1.Capo = I2.CodImpiegato;
10 |
11 |     IF (Stip_Sottoposto >= Stip_Capo) THEN
12 |         RAISE EXCEPTION 'Non valido: impossibile che lo stipendio di un
13 |         sottoposto sia maggiore o uguale di quello del suo capo';
14 |     END IF;
15 |
16 | RETURN NEW;
17 |
18 | END
19 | $$ LANGUAGE plpgsql;
20 |
21 | -----
22 |
23 | CREATE TRIGGER TriggerStipValido
24 | BEFORE INSERT ON Impiegato
25 | FOR EACH ROW
26 | EXECUTE FUNCTION StipValido();
```

### 4.2.2 Trigger GradoTipoValido

Questo trigger si occuperà di verificare la correttezza delle gerarchie all'interno dell'azienda. Il tipo e il grado di un Impiegato capo dovrà essere sempre superiore a quello del suo sottoposto. Un Impiegato di Tipo Dirigente non potrà mai avere un Capo di Tipo Dipendente. Mentre se sia il Sottoposto che il Capo sono Dipendenti, allora il Capo dovrà avere sempre un Grado maggiore (Senior > Middle > Junior).

```
01 | CREATE FUNCTION GradoTipoValido() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | TipoCapo Impiegato.Tipo%Type;
06 | TipoSottoposto Impiegato.Tipo%Type;
07 | GradoCapo Impiegato.Grado%Type;
08 | GradoSottoposto Impiegato.Grado%Type;
```

```

09 |
10 | BEGIN
11 |
12 | SELECT I1.Tipo, I2.Tipo, I1.Grado, I2.Grado INTO TipoSottoposto,
13 | TipoCapo, GradoSottoposto, GradoCapo
14 | FROM Impiegato AS I1, Impiegato AS I2
15 | WHERE I1.Capo = I2.CodImpiegato;
16 |
17 | IF (TipoCapo = 'Dipendente' AND TipoSottoposto = 'Dirigente') THEN
18 |     RAISE EXCEPTION 'Non valido: impossibile che un Dirigente abbia
19 |     un Capo Dipendente';
20 | END IF;
21 | IF (TipoCapo = 'Dipendente' AND TipoSottoposto = 'Dipendente') THEN
22 |     IF (GradoCapo = 'Junior') THEN
23 |         RAISE EXCEPTION 'Non valido: impossibile che un Dipendente
24 |         Junior sia Capo di qualcuno';
25 |     END IF;
26 |     ELSE IF (GradoCapo = 'Middle' AND GradoSottoposto = 'Senior') THEN
27 |         RAISE EXCEPTION 'Non valido: impossibile che un Dipendente
28 |         Middle sia il Capo di un Dipendente Senior';
29 |     END IF;
30 | END IF;
31 |
32 | RETURN NEW;
33 |
34 | END
35 | $$ LANGUAGE plpgsql;
36 |
37 | -----
38 |
39 | CREATE TRIGGER TriggerGradoTipiValido
40 | BEFORE INSERT ON Impiegato
41 | FOR EACH ROW
42 | EXECUTE FUNCTION GradoTipoValido();

```

#### 4.2.3 Trigger CheckImpiegati

Il Trigger CheckImpiegati verificherà la correttezza della relazione tra Grado e Tipo di un Impiegato. Se il Tipo di un Impiegato è Dirigente, allora il Grado dovrà essere NULL. Se il Tipo di un Impiegato è Dipendente, allora il Grado non può essere NULL.

```

01 | CREATE FUNCTION CheckImpiegati() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | GradoImp Impiegato.Grado%Type;
06 | TipoImp Impiegato.Tipo%Type;
07 |
08 | BEGIN
09 |
10 | SELECT I.Grado, I.Tipo INTO GradoImp, TipoImp

```

```

11 | FROM Impiegato AS I
12 | WHERE I.CodImpiegato = NEW.CodImpiegato;
13 |
14 | IF (TipoImp = 'Dipendente' AND GradoImp = NULL) THEN
15 |     RAISE EXCEPTION 'Non valido: deve avere un grado';
16 | END IF;
17 |
18 | IF (TipoImp = 'Dirigente' AND GradoImp <> NULL) THEN
19 |     RAISE EXCEPTION 'Non valido: non deve avere un grado';
20 | END IF;
21 |
22 | RETURN NEW;
23 |
24 | END
25 | $$ LANGUAGE plpgsql;
26 |
27 | -----
28 |
29 | CREATE TRIGGER TriggerCheckImpiegati
30 | BEFORE INSERT ON Impiegato
31 | FOR EACH ROW
32 | EXECUTE FUNCTION CheckImpiegati();

```

#### 4.2.4 Trigger PromozioneImpiegato

Un Impiegato Dipendente potrà ricevere una promozione aumentando il suo Grado, oppure addirittura diventando un Dirigente. In quest'ultimo caso, il Grado del neo Dirigente dovrà essere settato a NULL.

```

01 | CREATE FUNCTION PromozioneImpiegato() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | Imp Impiegato.CodImpiegato%Type;
06 |
07 | BEGIN
08 | SELECT I.CodImpiegato INTO Imp
09 | FROM Impiegato AS I
10 | WHERE I.CodImpiegato = NEW.CodImpiegato;
11 |
12 | UPDATE Impiegato AS I
13 | SET Grado = NULL
14 | WHERE I.CodImpiegato = Imp AND OLD.Tipo = 'Dipendente' AND NEW.Tipo = '
    Dirigente';
15 |
16 | RETURN NEW;
17 |
18 | END
19 | $$ LANGUAGE plpgsql;
20 |
21 | -----
22 |

```

```

23 | CREATE TRIGGER TriggerPromozioneImpiegato
24 | AFTER UPDATE ON Impiegato
25 | FOR EACH ROW
26 | EXECUTE FUNCTION PromozioneImpiegato();

```

#### 4.2.5 Trigger CorreggiGrado

Il seguente trigger setterà il Grado di un Dipendente appena inserito nel Database. Ci si baserà sugli anni di servizio prestati nell'Azienda per cui lavora, calcolati sottraendo l'anno ricavato dall'attributo DataAssunzione dall'anno corrente. Se il Dipendente avrà prestato meno di 3 anni di servizio, il suo Grado sarà settato a Junior. Per un valore compreso tra 3 e 6, il suo Grado sarà settato a Middle. Mentre se avrà lavorato per l'Azienda per più di 7 anni, sarà un Dipendente Senior.

```

01 | CREATE OR REPLACE FUNCTION az.CorreggiGrado () RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | AnniDiServizio integer;
06 | GradoImpiegato az.Impiegato.Grado%Type;
07 | TipoImpiegato az.Impiegato.Tipo%Type;
08 |
09 | BEGIN
10 |
11 | SELECT EXTRACT(YEAR FROM CURRENT_DATE) - EXTRACT(YEAR FROM I.
    DataAssunzione) INTO AnniDiServizio
12 | FROM az.Impiegato AS I
13 | WHERE I.CodImpiegato = NEW.CodImpiegato;
14 |
15 | SELECT I.Grado, I.Tipo INTO GradoImpiegato, TipoImpiegato
16 | FROM az.Impiegato AS I
17 | WHERE I.CodImpiegato = NEW.CodImpiegato;
18 |
19 | IF (TipoImpiegato = 'Dipendente') THEN
20 |     IF (AnniDiServizio < 3 AND Grado <> 'Junior') THEN
21 |         UPDATE az.Impiegato AS I
22 |         SET I.Grado = 'Junior'
23 |         WHERE I.CodImpiegato = NEW.CodImpiegato;
24 |     END IF;
25 |
26 |     IF (AnniDiServizio >=3 AND AnniDiServizio < 7 AND Grado <> 'Middle'
    ) THEN
27 |         UPDATE az.Impiegato AS I
28 |         SET I.Grado = 'Middle'
29 |         WHERE I.CodImpiegato = NEW.CodImpiegato;
30 |     END IF;
31 |
32 |     IF (AnniDiServizio >= 7 AND Grado <> 'Senior') THEN
33 |         UPDATE az.Impiegato AS I
34 |         SET I.Grado = 'Senior'
35 |         WHERE I.CodImpiegato = NEW.CodImpiegato;
36 |     END IF;

```

```

37 | END IF ;
38 |
39 | RETURN NEW;
40 |
41 | END
42 | $$ LANGUAGE plpgsql;
43 |
44 | _____
45 |
46 | CREATE TRIGGER TriggerCorreggiGrado
47 | BEFORE INSERT ON Impiegato
48 | FOR EACH ROW
49 | EXECUTE FUNCTION CorreggiGrado();

```

#### 4.2.6 Trigger FixStipendio

Il seguente trigger agirà in seguito all'aggiornamento della tabella Impiegato. Quando un Impiegato viene promosso ad un rango superiore, il trigger provvederà ad aumentare il suo stipendio di 5000.

```

01 | CREATE OR REPLACE FUNCTION FixStipendio() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 | TipoImp Impiegato.Tipo%TYPE;
05 | GradoImp Impiegato.Grado%TYPE;
06 |
07 | BEGIN
08 | SELECT I.Grado, I.Tipo INTO GradoImp, TipoImp
09 | FROM Impiegato AS I
10 | WHERE OLD.Grado <> NEW.Grado OR OLD.Tipo <> NEW.Tipo;
11 |
12 | IF ((OLD.Tipo = 'Dirigente' AND NEW.Tipo = 'Dipendente') OR (OLD.Grado
    = 'Senior' AND (NEW.Grado = 'Junior' OR NEW.Grado = 'Middle'))) OR (
    OLD.Grado = 'Middle' AND NEW.Grado = 'Junior')) THEN
13 |     RAISE EXCEPTION 'Passaggio non valido';
14 |
15 | ELSE
16 |     IF (OLD.Tipo = 'Dipendente' AND NEW.Tipo = 'Dipendente') THEN
17 |         UPDATE Impiegato AS I
18 |         SET I.Stipendio = I.Stipendio + 5000
19 |         WHERE OLD.CodImpiegato = NEW.CodImpiegato;
20 |
21 |     ELSE IF (OLD.Tipo = 'Dipendente' AND NEW.Tipo = 'Dirigente') THEN
22 |         UPDATE Impiegato AS I
23 |         SET I.Stipendio = 20000
24 |         WHERE OLD.CodImpiegato = NEW.CodImpiegato;
25 |
26 |     END IF;
27 | END IF;
28 |
29 | RETURN NEW;
30 |

```



```

31 | END
32 | $$ LANGUAGE plpgsql;
33 |
34 | ---
35 |
36 | CREATE TRIGGER TriggerFixStipendio
37 | AFTER UPDATE ON Impiegato
38 | FOR EACH ROW
39 | EXECUTE FUNCTION FixStipendio();

```

#### 4.2.7 Trigger SetStipendio

Il seguente trigger si provvederà ad impostare gli stipendi degli Impiegati nel Database.


- I Dipendenti di grado Junior percepiranno uno stipendio di 5000
- I Dipendenti di grado Middle percepiranno uno stipendio di 10000
- I Dipendenti di grado Senior percepiranno uno stipendio di 15000
- I Dirigenti percepiranno uno stipendio di 20000

```

01 | CREATE OR REPLACE FUNCTION SetStipendio() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | GradoImp Impiegato.Grado%TYPE;
06 | TipoImp Impiegato.Tipo%TYPE;
07 | StipendioImp Impiegato.Stipendio%TYPE;
08 |
09 | BEGIN
10 |
11 | SELECT NEW.Grado, NEW.Tipo, NEW.Stipendio INTO GradoImp, TipoImp,
    StipendioImp
12 | FROM Impiegato AS I
13 | WHERE I.CodImpiegato = NEW.CodImpiegato;
14 |
15 |
16 | IF (TipoImp = 'Dipendente' AND GradoImp = 'Junior' AND StipendioImp <>
    5000) THEN
17 |     UPDATE Impiegato AS I
18 |     SET I.Stipendio = 5000
19 |     WHERE I.CodImpiegato = NEW.CodImpiegato;
20 |
21 | ELSE IF (TipoImp = 'Dipendente' AND GradoImp = 'Middle' AND
    StipendioImp <> 10000) THEN
22 |     UPDATE Impiegato AS I
23 |     SET I.Stipendio = 10000
24 |     WHERE I.CodImpiegato = NEW.CodImpiegato;
25 |
26 | ELSE IF (TipoImp = 'Dipendente' AND GradoImp = 'Senior' AND
    StipendioImp <> 15000) THEN

```

```

27 | UPDATE Impiegato AS I
28 | SET I.Stipendio = 15000
29 | WHERE I.CodImpiegato = NEW.CodImpiegato;
30 |
31 | ELSE IF (TipoImp = 'Dirigente' AND StipendioImp <> 20000) THEN
32 | UPDATE Impiegato AS I
33 | SET I.Stipendio = 20000
34 | WHERE I.CodImpiegato = NEW.CodImpiegato;
35 |
36 | END IF;
37 |
38 | RETURN NEW;
39 | END
40 | $$ LANGUAGE plpgsql;
41 |
42 | 
43 |
44 | CREATE TRIGGER TriggerSetStipendio
45 | AFTER INSERT ON Impiegato
46 | FOR EACH ROW
47 | EXECUTE FUNCTION SetStipendio();

```

#### 4.2.8 Function getslab

La funzione prenderà in input un impiegato e, nel caso si trattasse di un Dipendente Senior dell'Azienda, restituirà tutti i nomi dei Laboratori di cui è Responsabile Scientifico.

```

01 | CREATE OR REPLACE FUNCTION getlabs(imp Impiegato.CodImpiegato%TYPE)
02 | RETURNS Laboratorio.Nome%TYPE AS
03 | $$
04 | DECLARE
05 | NomeLab Laboratorio.Nome%TYPE;
06 | TipoImp Impiegato.Tipo%TYPE;
07 | GradoImp Impiegato.Grado%TYPE;
08 | res VARCHAR(100) = " ";
09 | curslab CURSOR FOR(
10 | SELECT L.Nome
11 | FROM Laboratorio AS L
12 | WHERE L.ResponsabileScientifico = imp);
13 |
14 | BEGIN
15 | SELECT I.Tipo, I.Grado INTO TipoImp, GradoImp
16 | FROM Impiegato AS I
17 | WHERE CodImpiegato = imp;
18 |
19 | IF (TipoImp <> 'Dipendente' AND GradoImp <> 'Senior') THEN
20 | RAISE EXCEPTION 'Non valido: deve essere un dipendente senior';
21 | END IF;
22 |
23 | OPEN curslab;
24 | LOOP

```

```

25 |             FETCH curslab INTO NomeLab;
26 |             EXIT WHEN curslab %NOT FOUND;
27 |             res = res || NomeLab;
28 |     END LOOP;
29 |     CLOSE curslab;
30 | RETURN res;
31 | END
32 | $$ LANGUAGE plpgsql;

```

#### 4.2.9 Function getimpiegato

La seguente funzione prenderà in input il codice di un'azienda e restituirà la lista di nomi e cognomi degli impiegati che vi lavorano.

```

01 | CREATE OR REPLACE FUNCTION getimpiegato(az Azienda.CodAzienda%TYPE)
02 | RETURNS VARCHAR(1000) AS
03 | $$
04 | DECLARE
05 |     NomeImp Impiegato.Nome%TYPE;
06 |     CognomeImp Impiegato.Cognome%TYPE;
07 |     res VARCHAR(100) = "Per la suddetta azienda lavorano: ";
08 |     ris VARCHAR(1000) = " ";
09 |     risultatofinale VARCHAR(1000) = " ";
10 |
11 |     cursaz CURSOR FOR(
12 |     SELECT I.Nome, I.Cognome
13 |     FROM Impiegato AS I JOIN Azienda AS A ON I.CodAzienda = A.CodAzienda
14 |     WHERE A.CodAzienda = az);
15 |
16 | BEGIN
17 |
18 |     OPEN cursaz;
19 |     LOOP
20 |         FETCH cursaz INTO NomeImp, CognomeImp;
21 |         EXIT WHEN cursprogfromlab %NOT FOUND;
22 |
23 |         ris = ris || NomeImp || CognomeImp;
24 |     END LOOP;
25 |     risultatofinale = res || ris;
26 |     CLOSE cursaz;
27 | RETURN risultatofinale;
28 | END
29 | $$ LANGUAGE plpgsql;

```

#### 4.2.10 Function getprog

La seguente funzione prenderà in input il codice di un impiegato. Se il suddetto impiegato è un Dirigente, verranno restituiti i nomi dei Progetti di cui è il Responsabile. Se è un Dipendente Senior verrà restituita la lista dei nomi di Progetto di cui è Referente Scientifico. Altrimenti, verrà restituito un messaggio di errore.

```

01 | CREATE OR REPLACE FUNCTION getprog(imp Impiegato.CodImpiegato%TYPE)
02 | RETURNS Progetto.Nome%TYPE AS
03 | $$
04 | DECLARE
05 | NomeProg Progetto.Nome%TYPE;
06 | TipoImp Impiegato.Tipo%TYPE;
07 | GradoImp Impiegato.Grado%TYPE;
08 | res1 CHAR(100) = "Il suddetto impiegato      il referente scientifico di:
      ";
09 | res2 CHAR(100) = "Il suddetto impiegato      il responsabile di: ";
10 | ris VARCHAR(1000) = " ";
11 | risultatofinale VARCHAR(1000) = " ";
12 | cursprog CURSOR FOR(
13 | SELECT P.Nome
14 | FROM Progetto AS P
15 | WHERE P.ReferenteScientifico = imp OR P.Responsabile = imp);
16 |
17 | BEGIN
18 |     SELECT I.Tipo, I.Grado INTO TipoImp, GradoImp
19 |     FROM Impiegato AS I
20 |     WHERE CodImpiegato = imp;
21 |
22 |     IF ((TipoImp = 'Dipendente' AND GradoImp <> 'Senior') OR (TipoImp <>
      'Dirigente')) THEN
23 |         RAISE EXCEPTION 'Non valido: deve essere un dipendente senior o un
      dirigente';
24 |     END IF;
25 |
26 |
27 |     OPEN cursprog;
28 |     IF (TipoImp = 'Dipendente' AND GradoImp = 'Senior') THEN
29 |         LOOP
30 |             FETCH cursprog INTO NomeProg;
31 |             EXIT WHEN curslab %NOT FOUND;
32 |
33 |             ris = ris || NomeProg;
34 |         END LOOP;
35 |         risultatofinale = res1 || ris;
36 |     END IF;
37 |
38 |     ELSE IF (TipoImp = 'Dirigente') THEN
39 |         LOOP
40 |             FETCH cursprog INTO NomeProg;
41 |             EXIT WHEN curslab %NOT FOUND;
42 |
43 |             ris = ris || NomeProg;
44 |         END LOOP;
45 |         risultatofinale = res2 || ris;
46 |     END IF;
47 |     CLOSE cursprog;
48 |

```

```

49 | RETURN risultatofinale;
50 | END
51 | $$ LANGUAGE plpgsql;

```

## 4.3 Triggers e Functions per la gestione dei Laboratori

### 4.3.1 Trigger CheckLaboratorio

Ogni Laboratorio avrà un Referente Scientifico, il quale deve essere un Dipendente Senior dell'azienda. Il seguente trigger verificherà la correttezza di questa condizione.

```

01 | CREATE FUNCTION CheckLaboratorio () RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | RespScientifico Impiegato.CodImpiegato%Type;
06 | Imp Impiegato.CodImpiegato%Type;
07 | GradoImp Impiegato.Grado%Type;
08 | TipoImp Impiegato.Tipo%Type;
09 |
10 | BEGIN
11 |
12 | SELECT I.Grado INTO GradoImp
13 | FROM Impiegato AS I JOIN Laboratorio AS L ON I.CodImpiegato =
14 | L.ResponsabileScientifico
15 | WHERE L.CodLab = NEW.CodLab;
16 |
17 | IF (GradoImp <> 'Senior') THEN
18 |     RAISE EXCEPTION 'Non valido: deve essere un dipendente senior';
19 | END IF;
20 |
21 | RETURN NEW;
22 |
23 | END
24 | $$ LANGUAGE plpgsql;
25 |
26 | -----
27 |
28 | CREATE TRIGGER TriggerCheckLaboratorio
29 | BEFORE INSERT ON Laboratorio
30 | FOR EACH ROW
31 | EXECUTE FUNCTION CheckLaboratorio();

```

### 4.3.2 Trigger LaboratorioAperto

Se il Responsabile Scientifico di un Laboratorio appena inserito nel Database è NULL, allora il valore di Aperto sarà settato automaticamente a 'N'.

```

01 |
02 | CREATE OR REPLACE FUNCTION LaboratorioAperto() RETURNS TRIGGER AS
03 | $$

```

```

04 | DECLARE
05 |
06 | BEGIN
07 |
08 | IF (NEW.ResponsabileScientifico = NULL) THEN
09 |     UPDATE Laboratorio
10 |     SET Aperto = 'N';
11 |
12 | ELSE
13 |     UPDATE Laboratorio
14 |     SET Aperto = 'S';
15 | END IF;
16 |
17 | RETURN NEW;
18 |
19 | END
20 | $$ LANGUAGE plpgsql;
21 |
22 | -----
23 |
24 | CREATE TRIGGER TriggerLaboratorioAperto
25 | BEFORE INSERT ON Laboratorio
26 | FOR EACH ROW
27 | EXECUTE FUNCTION LaboratorioAperto();

```

#### 4.3.3 Trigger ChiusuraLaboratorio

Il seguente trigger si occuperà di settare l'attributo *Aperto* a 'N' e l'attributo *ResponsabileScientifico* a NULL qualora l'Impiegato che svolge il ruolo dovesse essere cancellato dal DB.

```

01 | CREATE FUNCTION ChiusuraLaboratorio() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | BEGIN
06 |
07 | UPDATE Laboratorio AS L
08 | SET L.ResponsabileScientifico = NULL,
09 | Aperto = 'N'
10 | WHERE L.ResponsabileScientifico = OLD.CodImpiegato;
11 |
12 | RETURN NEW;
13 |
14 | END
15 | $$ LANGUAGE plpgsql;
16 |
17 | -----
18 |
19 | CREATE TRIGGER TriggerChiusuraLab
20 | AFTER DELETE ON Impiegato
21 | FOR EACH ROW

```

```
22 | EXECUTE FUNCTION ChiusuraLaboratorio();
```

#### 4.3.4 Function getprogfromlab

La seguente funzione prenderà in input il codice di un laboratorio e restituirà i nomi dei progetti di cui il suddetto lab si occupa.

```
01 | CREATE OR REPLACE FUNCTION getprogfromlab(lab Laboratorio.CodLab%TYPE)
02 | RETURNS Progetto.Nome%TYPE AS
03 | $$
04 | DECLARE
05 | NomeProg Progetto.Nome%TYPE;
06 | res VARCHAR(100) = " ";
07 |
08 | cursprogfromlab CURSOR FOR(
09 | SELECT P.Nome
10 | FROM Progetto AS P
11 | WHERE P.Lab1 = lab OR P.Lab2 = lab OR P.Lab3 = lab);
12 |
13 | BEGIN
14 |
15 |     OPEN cursprogfromlab;
16 |     LOOP
17 |         FETCH cursprogfromlab INTO NomeProg;
18 |         EXIT WHEN cursprogfromlab %NOT FOUND;
19 |
20 |         res = res || NomeProg;
21 |     END LOOP;
22 |     CLOSE cursprogfromlab;
23 | RETURN res;
24 | END
25 | $$ LANGUAGE plpgsql;
```

### 4.4 Triggers e Functions per la gestione dei Progetti

#### 4.4.1 Function CheckProgetto

Ogni Progetto ha un Referente Scientifico e un Responsabile, che sono rispettivamente un Dipendente Senior e un Dirigente dell'Azienda. Il Trigger verificherà la correttezza del Tipo e del Grado degli Impiegati che assumono i due ruoli.

```
01 | CREATE FUNCTION CheckProgetto() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | GradoResp Impiegato.Grado%Type;
06 | GradoReferente Impiegato.Grado%Type;
07 | TipoResp Impiegato.Tipo%Type;
08 | TipoReferente Impiegato.Tipo%Type;
09 |
10 | BEGIN
```

```

11 |
12 | SELECT I.Grado, I.Tipo INTO GradoResp, TipoResp
13 | FROM Impiegato AS I JOIN Progetto AS PR ON I.CodImpiegato = PR.
    Responsabile
14 | WHERE I.CodImpiegato = NEW.Responsabile;
15 |
16 | SELECT I.Grado, I.Tipo INTO GradoReferente, TipoReferente
17 | FROM Impiegato AS I JOIN Progetto AS PR ON I.CodImpiegato = PR.
    ReferenteScientifico
18 | WHERE I.CodImpiegato = NEW.ReferenteScientifico;
19 |
20 | IF (TipoReferente = 'Dirigente' OR (TipoReferente = 'Dipendente' AND
    GradoReferente <> 'Senior')) THEN
21 | RAISE EXCEPTION 'Non valido: il referente scientifico deve essere un
    dipendente senior';
22 | END IF;
23 |
24 | IF (TipoResp <> 'Dirigente') THEN
25 | RAISE EXCEPTION 'Non valido: il responsabile deve essere un dirigente';
26 | END IF;
27 |
28 | RETURN NEW;
29 |
30 | END
31 | $$ LANGUAGE plpgsql;
32 |
33 | -----
34 |
35 | CREATE TRIGGER TriggerCheckProgetto
36 | BEFORE INSERT ON Progetto
37 | FOR EACH ROW
38 | EXECUTE FUNCTION CheckProgetto();

```

#### 4.4.2 Trigger ProgettoAttivo

Un Progetto può essere inattivo in due casi:

- Uno tra il Referente Scientifico e il Responsabile è NULL
- Tutti e tre i possibili Laboratori che potrebbero occuparsi di esso sono NULL

Se si verifica almeno una di queste due condizioni all'inserimento di un Progetto nel Database, allora l'attributo *Attivo* sarà settato a 'N'. Altrimenti, sarà settato a 'S'.

```

01 | CREATE FUNCTION ProgettoAttivo() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | RefScientifico Impiegato.CodImpiegato%Type;
06 | Resp Impiegato.CodImpiegato%Type;
07 | PrimoLaboratorio Laboratorio.CodLab%Type;
08 | SecondoLaboratorio Laboratorio.CodLab%Type;
09 | TerzoLaboratorio Laboratorio.CodLab%Type;

```



```

10 |
11 | BEGIN
12 |
13 | SELECT P.ReferenteScientifico , P.Responsabile , P.Lab1 , P.Lab2 , P.Lab3
14 | INTO RefScientifico , Resp , PrimoLaboratorio , SecondoLaboratorio ,
      TerzoLaboratorio
15 | FROM Progetto AS P
16 | WHERE P.CUP = NEW.CUP;
17 |
18 | IF (( RefScientifico = NULL OR Resp = NULL) OR (PrimoLaboratorio = NULL
19 | AND SecondoLaboratorio = NULL AND TerzoLaboratorio = NULL)) THEN
20 |     UPDATE Progetto AS P
21 |     SET Attivo = 'N'
22 |     WHERE P.CUP = NEW.CUP;
23 |
24 | ELSE
25 |     UPDATE Progetto AS P
26 |     SET Attivo = 'S'
27 |     WHERE P.CUP = NEW.CUP;
28 | END IF;
29 |
30 | RETURN NEW;
31 |
32 | END
33 | $$ LANGUAGE plpgsql;
34 |
35 | -----
36 |
37 | CREATE TRIGGER TriggerProgettoAttivo
38 | BEFORE INSERT ON Progetto
39 | FOR EACH ROW
40 | EXECUTE FUNCTION ProgettoAttivo();

```

#### 4.4.3 Trigger ChiusuraProgettoImpiegato

Il seguente trigger sistemerà la Tabella Progetto in caso uno tra il Referente Scientifico e il Responsabile venga cancellato dal Database. Il relativo attributo sarà settato a NULL e Attivo a 'N'.

```

01 | CREATE FUNCTION ChiusuraProgettoImpiegato() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | ImpiegatoCancellato Impiegato.CodImpiegato%Type;
06 |
07 | BEGIN
08 |
09 | SELECT I.CodImpiegato INTO ImpiegatoCancellato
10 | FROM Impiegato AS I
11 | WHERE I.CodImpiegato = OLD.CodImpiegato;
12 |
13 | IF(ImpiegatoCancellato = ReferenteScientifico) THEN

```

```

14 |     UPDATE Progetto AS P
15 |     SET P.ReferenteScientifico = NULL, P.Attivo = 'N'
16 |     WHERE P.ReferenteScientifico = OLD.CodImpiegato;
17 | END IF;
18 |
19 | IF (ImpiegatoCancellato = Responsabile) THEN
20 |     UPDATE Progetto AS P
21 |     SET P.Responsabile = NULL, P.Attivo = 'N'
22 |     WHERE P.Responsabile = OLD.CodImpiegato;
23 | END IF;
24 |
25 | RETURN NEW;
26 |
27 | END
28 | $$ LANGUAGE plpgsql;
29 |
30 | -----
31 |
32 | CREATE TRIGGER TriggerChiusuraProgettoImpiegato
33 | AFTER DELETE ON Impiegato
34 | FOR EACH ROW
35 | EXECUTE FUNCTION ChiusuraProgettoImpiegato();

```

#### 4.4.4 Trigger ChiusuraProgLaboratorio

Invece, qualora dovesse essere cancellato uno dei Laboratori che si occupa di un Progetto, si verificherà se almeno uno degli altri due sia aperto. In caso contrario, imposteremo Attivo a 'N'.

```

01 | CREATE FUNCTION ChiusuraProgLaboratorio() RETURNS TRIGGER AS
02 | $$
03 | DECLARE
04 |
05 | LabCancellato Laboratorio.CodLab%Type;
06 | PrimoLaboratorio Laboratorio.CodLab%Type;
07 | SecondoLaboratorio Laboratorio.CodLab%Type;
08 | TerzoLaboratorio Laboratorio.CodLab%Type;
09 |
10 | BEGIN
11 |
12 | SELECT L.CodLab, L.Lab1, L.Lab2, L.Lab3 INTO LabCancellato,
13 |     PrimoLaboratorio, SecondoLaboratorio, TerzoLaboratorio
14 | FROM Laboratorio AS L
15 | WHERE L.CodLab = OLD.CodLab;
16 |
17 | IF (LabCancellato = PrimoLaboratorio AND SecondoLaboratorio = NULL AND
18 |     TerzoLaboratorio = NULL) THEN
19 |     UPDATE Progetto AS P
20 |     SET P.Lab1 = NULL, P.Attivo = 'N'
21 |     WHERE P.Lab1 = OLD.CodLab;
22 | END IF;

```

```

23 | IF(LabCancellato = SecondoLaboratorio AND PrimoLaboratorio = NULL AND
24 | TerzoLaboratorio = NULL) THEN
25 |     UPDATE Progetto AS P
26 |     SET P.Lab2 = NULL, P.Attivo = 'N'
27 |     WHERE P.Lab2 = OLD.CodLab;
28 | END IF;
29 |
30 | IF(LabCancellato = TerzoLaboratorio AND PrimoLaboratorio = NULL AND
31 | SecondoLaboratorio = NULL) THEN
32 |     UPDATE Progetto AS P
33 |     SET P.Lab3 = NULL, P.Attivo = 'N'
34 |     WHERE P.Lab3 = OLD.CodLab;
35 | END IF;
36 |
37 | RETURN NEW;
38 |
39 | END
40 | $$ LANGUAGE plpgsql;
41 |
42 | -----
43 |
44 | CREATE TRIGGER TriggerChiusuraProgLaboratorio
45 | AFTER DELETE ON Laboratorio
46 | FOR EACH ROW
47 | EXECUTE FUNCTION ChiusuraProgLaboratorio();

```

## 4.5 Analisi scatti di carriera degli Impiegati

Il sistema vorrà tenere traccia di tutti gli scatti di carriera degli Impiegati. Verrà creata una nuova vista, chiamata *ScattiDiCarriera*, che verrà inizializzata inserendo i valori attualmente presenti nella tabella Impiegato. Successivamente, ogni volta che il Grado di un Impiegato sarà aggiornato (può diventare Middle, Senior oppure NULL se l'Impiegato è diventato un Dirigente), allora esso sarà re-inserito all'interno della vista con il nuovo grado, il nuovo stipendio e l'anno in cui è stato promosso.

```

01 | CREATE VIEW ScattiDiCarriera (CodImpiegato, Nome, Cognome, Tipo, Grado,
02 | Stipendio, AnnoDiPromozione) AS
03 | SELECT I.CodImpiegato, I.Nome, I.Cognome, I.Tipo, I.Grado, I.Stipendio,
04 |     EXTRACT(YEAR FROM CURRENTDATE)
05 | FROM Impiegato AS I
06 |
07 | CREATE FUNCTION ScattiDiCarriera() RETURNS TRIGGER AS
08 | $$
09 | DECLARE
10 |
11 | BEGIN
12 |
13 | INSERT INTO ScattiDiCarriera(
14 | SELECT I.CodImpiegato, I.Nome, I.Cognome, I.Tipo, I.Grado, I.Stipendio,
15 | EXTRACT(YEAR FROM CURRENT_DATE)
16 | FROM Impiegato AS I

```

```
17 | WHERE I.CodImpiegato = NEW.CodImpiegato AND OLD.Grado <> NEW.Grado);
18 |
19 | RETURN NEW;
20 | END
21 |
22 | $$ LANGUAGE plpgsql;
23 |
24 | _____
25 |
26 | CREATE TRIGGER TriggerScattiDiCarriera
27 | AFTER UPDATE ON Impiegato
28 | FOR EACH ROW
29 | EXECUTE FUNCTION ScattiDiCarriera();
```