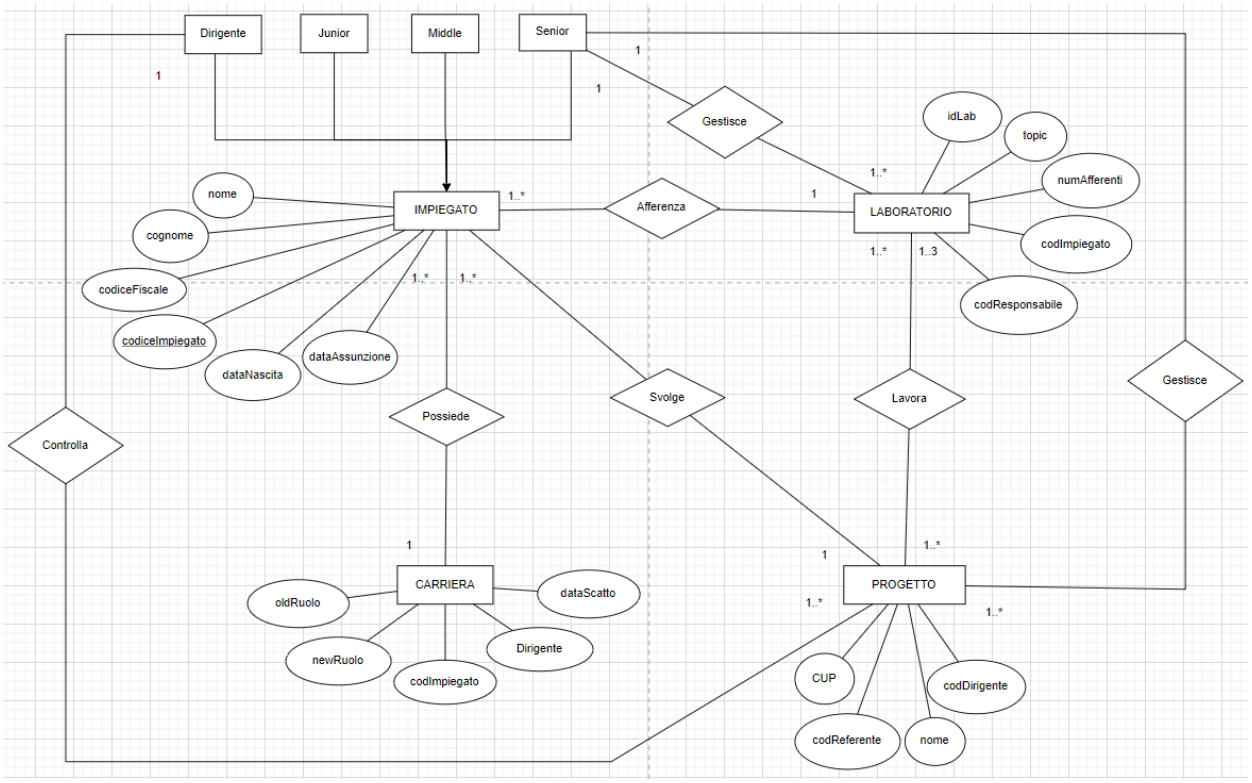
Progetto Azienda

Federica Mele, Antonio Marselli

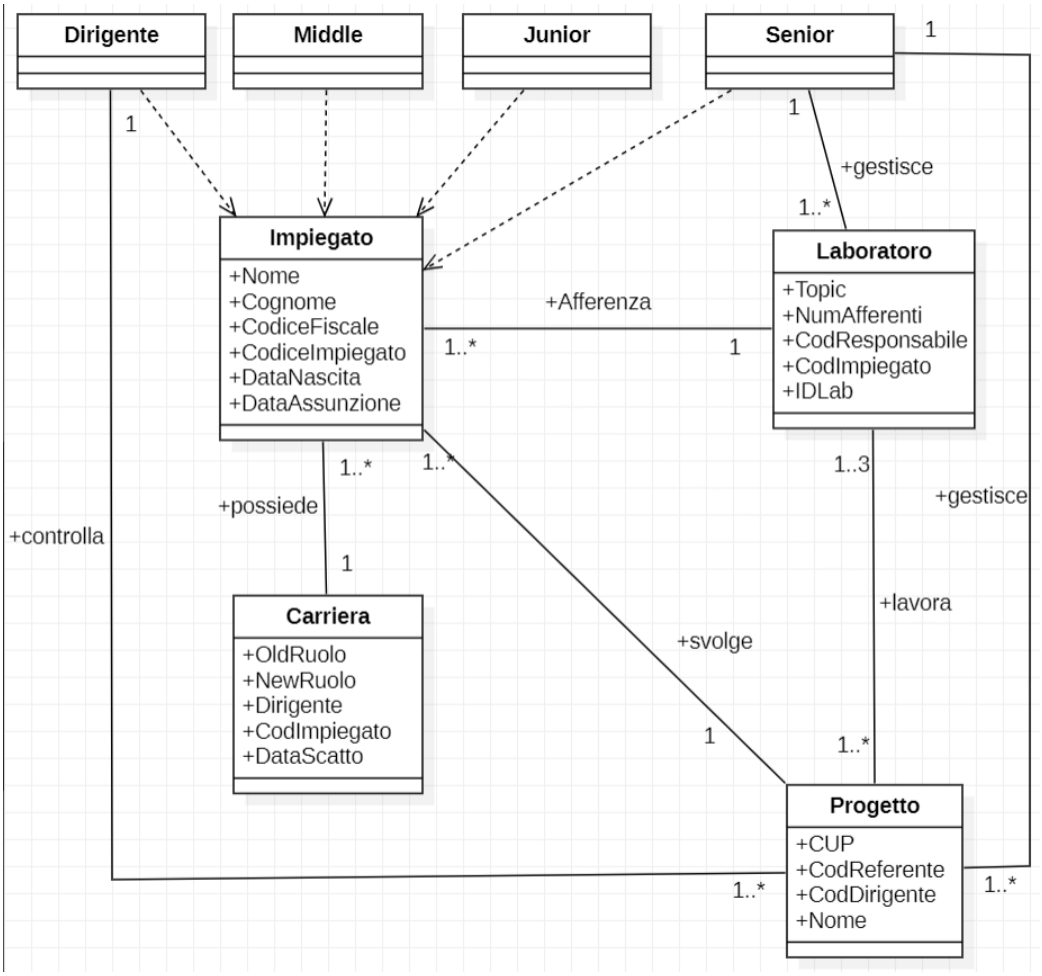
SISTEMA DI GESTIONE DEL PERSONALE E DEI PROGETTI ALL'INTERNO DI UN'AZIENDA

Si sviluppi un sistema informativo, composto da una base di dati relazionale e da un applicativo Java dotato di GUI (Swing o JavaFX), per la gestione del personale di un'azienda. L'azienda possiede un certo numero di impiegati, raggruppabili in 4 categorie: 1- Dipendente junior: colui che lavora da meno di 3 anni all'interno dell'azienda; 2- Dipendente middle: colui che lavora da meno di 7 ma da più di tre anni per l'azienda; 3- Dipendente senior: colui che lavora da almeno 7 anni per l'azienda. 4- Dirigenti: la classe dirigente non ha obblighi temporali di servizio. Chiunque può diventare dirigente, se mostra di averne le capacità. I passaggi di ruolo avvengono per anzianità di servizio. È necessario tracciare tutti gli scatti di carriera per ogni dipendente. Nell'azienda vengono gestiti laboratori e progetti. Un laboratorio ha una particolare topic di cui si occupa, un certo numero di afferenti ed un responsabile scientifico che è un dipendente senior. Un progetto è identificato da un CUP (codice unico progetto) e da un nome (unico nel sistema). Ogni progetto ha un referente scientifico, il quale deve essere un dipendente senior dell'ente, ed un responsabile che è uno dei dirigenti. Al massimo 3 laboratori possono lavorare ad un progetto.

ER NON RISTRUTTURATO



UML NON RISTRUTTURATO



MODELLO LOGICO

IMPIEGATI (codiceimpiegato, datanascita, nome, cognome, codicefiscale, dataassunzione, dirigente, ruolo).

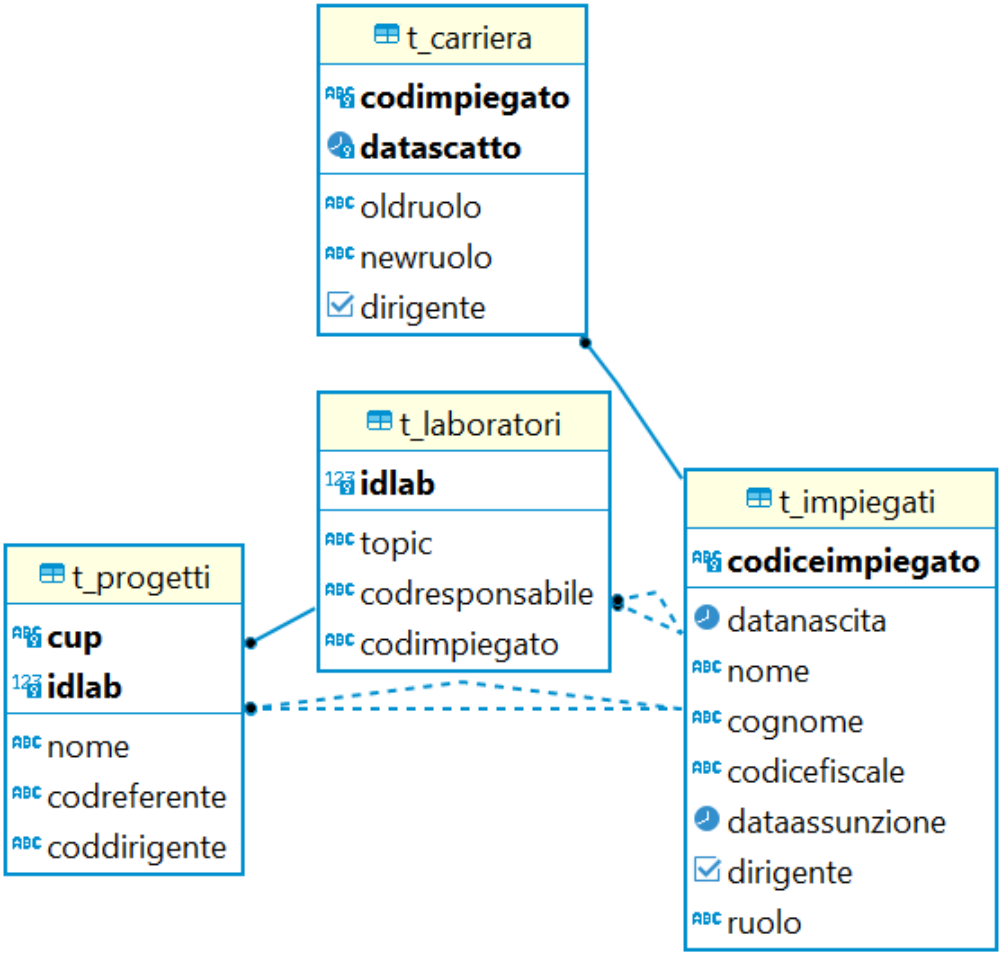
*LABORATORIO (idlab, topic, *codresponsabile*, *codimpiegato*).*
- Codresponsabile → impiegati.codiceimpiegato
- Codimpiegato → impiegati.codiceimpiegato

*PROGETTI (CUP, idlab, nome, *codreferente*, *coddirigente*).*
- Codreferente → impiegati.codiceimpiegato
- Coddirigente → impiegati.codiceimpiegato

CARRIERA (codimpiegato, datascatto, oldruolo, newruolo, dirigente).
- Codimpiegato → impiegati.codiceimpiegato

*N.B. *.....* → foreign key*

UML RISTRUTTURATO



TABELLE

1. IMPIEGATI

```
Struttura della tabella 'Impiegati'

CREATE TABLE t_impiegati (
codiceImpiegato varchar NOT NULL,
dataNascita date NOT NULL check
(DATE_PART('year', AGE(current_date, dataNascita)) < 65),
nome varchar NOT NULL,
cognome varchar NOT NULL,
codiceFiscale varchar NOT null check
(char_length(codiceFiscale) = 16),
dataAssunzione date NOT NULL check
(DATE_PART('year', AGE(dataAssunzione,
dataNascita)) > 17) ,
dirigente bool NOT NULL DEFAULT false,
ruolo varchar NULL,
CONSTRAINT t_impiegati_cf_unique UNIQUE (codiceFiscale),
CONSTRAINT t_impiegato_pk PRIMARY KEY (codiceImpiegato)
);
```

Nella creazione della tabella impiegati sono stati introdotti:

a) Primary Key

codiceImpiegato è la chiave primaria della tabella t_impiegati che rappresenta l'identificativo univoco dell'impiegato.

b) Vincolo di unicità sul CF

Controllo di unicità del campo codiceFiscale nella tabella t_impiegati in quanto non si deve poter inserire un codice fiscale già esistente in tabella.

c) Vincolo sull'età dell'impiegato:

$$dataAssunzione–dataNascita > 17anni$$

L'impiegato deve essere maggiorenne, quindi verrà calcolata la differenza tra la data di assunzione e la data di nascita dell'impiegato, questa dev'essere di almeno 18.

d) Vincolo di anzianità dell'impiegato:

$$datacorrente–dataNascita < 65$$

L'impiegato deve avere meno di 65 anni, quindi verrà calcolata la differenza tra la data attuale e la data di nascita dell'impiegato, questa dev'essere minore di 65.

e) Vincolo di validazione del CF:

Il codice fiscale deve essere 16 caratteri.

I constraint possono essere aggiunti anche in seguito alla creazione tramite l'istruzione ALTER TABLE:

ALTER TABLE t_impiegati ADD CONSTRAINT t_impiegati_cf_unique UNIQUE (codiceFiscale);

ALTER TABLE t_impiegati ADD CONSTRAINT t_impiegato_pk PRIMARY KEY (codiceImpiegato);

2. LABORATORI

```
CREATE TABLE t_laboratori (
idLab int4 NOT NULL,
topic varchar NOT NULL,
codResponsabile varchar NOT NULL,
codImpiegato varchar NOT NULL,
CONSTRAINT t_laboratori_pk PRIMARY KEY (idLab,codImpiegato),
CONSTRAINT t_laboratori_unique UNIQUE (codImpiegato),
CONSTRAINT t_laboratori_imp_fk FOREIGN KEY (codImpiegato)
  REFERENCES t_impiegati(codiceImpiegato),
CONSTRAINT t_laboratori_resp_fk FOREIGN KEY (codResponsabile)
  REFERENCES t_impiegati(codiceImpiegato));
```

Nella creazione della tabella laboratori sono stati introdotti:
a) Primary key

La coppia idLab,codImpiegato è la chiave primaria che rappresenta univocamente il laboratorio.

b) Vincolo di unicità sul codImpiegato

L'impiegato può lavorare su un solo laboratorio

c) Foreign key

codResponsabile e codImpiegato devono essere presenti nella tabella t_impiegati.

I constraint possono essere aggiunti anche in seguito alla creazione tramite l'istruzione ALTER TABLE:
ALTER TABLE t_laboratori ADD CONSTRAINT t_laboratori_unique UNIQUE (codimpiegato);

ALTER TABLE t_laboratori ADD CONSTRAINT t_laboratori_pk PRIMARY KEY (idLab,codimpiegato);

```
ALTER TABLE t_laboratori ADD CONSTRAINT t_laboratori_imp_fk
FOREIGN KEY (codResponsabile) REFERENCES t_impiegati(codiceImpiegato);
```

3. PROGETTI

```
CREATE TABLE t_progetti (
CUP varchar NOT NULL,
nome varchar NOT NULL,
codReferente varchar NOT NULL,
codDirigente varchar NOT NULL,
idLab int4 NOT NULL,
CONSTRAINT t_progetto_pk PRIMARY KEY (CUP, idLab),
CONSTRAINT t_progetti_nome_unique UNIQUE (nome),
CONSTRAINT t_progetto_ref_sc_fk FOREIGN KEY (codReferente)
REFERENCES t_impiegati(codiceImpiegato),
CONSTRAINT t_progetto_resp_fk FOREIGN KEY (codDirigente)
REFERENCES t_impiegati(codiceImpiegato),
CONSTRAINT t_progetto_id_lab_fk FOREIGN KEY (idLab)
REFERENCES t_laboratori(idLab)
);
```

Nella creazione della tabella progetti sono stati introdotti:
a) Primary key

I laboratori lavorano sui progetti. Quindi per ciascun progetto possono esserci più laboratori. La chiave univoca, quindi, dovrà essere la coppia formata da CUP e idLab.

b) Vincolo di unicità sul nome del progetto

Controllo di unicità del campo nome nella tabella t_progetti in quanto non si deve poter inserire un nome di progetto già esistente in tabella.

c) Foreign key

codReferente e codDirigente devono essere presenti nella tabella t_impiegati.

idLab deve essere presente nella tabella t_laboratori.

I constraint possono essere aggiunti anche in seguito alla creazione tramite l'istruzione ALTER TABLE.

Per esempio, per il constraint unique:

ALTER TABLE t_impiegati ADD CONSTRAINT t_progetti_nome_unique UNIQUE (nome);

ALTER TABLE t_impiegati ADD CONSTRAINT t_progetto_pk PRIMARY KEY (CUP, idLab);

ALTER TABLE t_impiegati ADD CONSTRAINT t_progetto_ref_sc_fk FOREIGN KEY (codReferente) REFERENCES t_impiegati(codiceImpiegato);

ALTER TABLE t_impiegati ADD CONSTRAINT t_progetto_resp_fk FOREIGN KEY (codDirigente) REFERENCES t_impiegati(codiceImpiegato);

ALTER TABLE t_impiegati ADD CONSTRAINT t_progetto_id_lab_fk FOREIGN KEY (idLab) REFERENCES t_laboratori(idLab);

4. CARRIERA

```
CREATE TABLE t_carriera (
codImpiegato varchar NOT NULL,
oldRuolo varchar NULL,
newRuolo varchar NULL,
dataScatto date NOT NULL,
dirigente bool NULL DEFAULT false,
CONSTRAINT t_carriera_pk PRIMARY KEY
(codImpiegato, dataScatto),
CONSTRAINT t_carriera_imp_fk FOREIGN KEY (codImpiegato)
REFERENCES t_impiegati(codiceImpiegato)
);
```

Nella creazione della tabella carriera sono stati introdotti:
a) Primary key

Per ciascun impiegato verrà tracciato il passaggio di carriera e/o la promozione a dirigente. La chiave univoca, quindi, dovrà essere la coppia formata dal codImpiegato e la data in cui avviene lo scatto di carriera (ruolo o dirigenza).

b) Foreign key

codImpiegato deve essere presente nella tabella t_impiegati

I constraint possono essere aggiunti anche in seguito alla creazione tramite l'istruzione ALTER TABLE:
ALTER TABLE t_impiegati ADD CONSTRAINT t_carriera_pk PRIMARY KEY (codImpiegato, dataScatto);

ALTER TABLE t_impiegati ADD CONSTRAINT t_carriera_imp_fk FOREIGN KEY (codImpiegato) REFERENCES t_impiegati(codiceImpiegato);

TRIGGER E FUNZIONI

1. TRIGGER RUOLO

   Abbiamo creato un trigger che agisce prima dell'inserimento di un impiegato e/o prima della modifica delle informazioni di un impiegato.
   Il trigger richiama una funzione che determina il ruolo dell'impiegato in base agli anni che sono trascorsi dalla data di assunzione.
   La funzione valorizzerà quindi opportunamente il campo ruolo della tabella t_impiegati

```
CREATE OR REPLACE FUNCTION checkRole() RETURNS TRIGGER AS $$
begin

if(NEW."dirigente" is not null and
NEW."dirigente" = false) then
if ( (select DATE_PART('year',
AGE(current_date, new."dataassunzione"))
AS year) <3) THEN
NEW."ruolo" := 'JUNIOR';
elseif ( (select DATE_PART('year',
AGE(current_date, new."dataassunzione"))
AS year) > 7) THEN
NEW."ruolo" := 'SENIOR';
else
NEW."ruolo" :='MIDDLE';
end if;
else
NEW."ruolo" := null;
end if;
RETURN new;
END $$ LANGUAGE plpgsql;

CREATE TRIGGER writeRole BEFORE INSERT OR UPDATE ON t_impiegati
FOR EACH ROW
EXECUTE PROCEDURE checkRole();
```

2. TRIGGER CARRIERA

Abbiamo creato un trigger che agisce dopo l'inserimento di un impiegato e/o dopo la modifica delle sue informazioni.
Il trigger richiama una funzione che inserisce, sulla tabella t_carriera, l'eventuale modifica del ruolo o della qualifica di dirigente.

```
CREATE OR REPLACE FUNCTION checkcareer()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
begin

CREATE OR REPLACE FUNCTION public.checkcareer()
RETURNS trigger
LANGUAGE plpgsql
AS $function$

begin

if ( OLD."ruolo" is null and (NEW."dirigente" is null
 or NEW."dirigente" = false)) THEN
INSERT INTO t_carriera (codImpiegato,oldRuolo, newRuolo,
dirigente, dataScatto)
VALUES(NEW.codiceImpiegato,
OLD.ruolo,
'JUNIOR',
false,
(select dataAssunzione
from t_impiegati where
codiceImpiegato = NEW.codiceImpiegato));
if(NEW.ruolo = 'MIDDLE' or NEW.ruolo = 'SENIOR') THEN
INSERT INTO t_carriera (codImpiegato,oldRuolo,
newRuolo, dirigente, dataScatto)
VALUES(NEW.codiceImpiegato,
'JUNIOR',
'MIDDLE',
false,
(select dataAssunzione + interval '3 year'
```

```
from t_impiegati where
codiceImpiegato = NEW.codiceImpiegato));
end if;
if(NEW.ruolo = 'SENIOR') THEN
INSERT INTO t_carriera  (codImpiegato,oldRuolo,
newRuolo, dirigente, dataScatto)
VALUES(NEW.codiceImpiegato,
'MIDDLE',
'SENIOR',
false,
(select dataAssunzione + interval '7 year'
from t_impiegati where
codiceImpiegato = NEW.codiceImpiegato));
end if;
elseif ( OLD."ruolo" != NEW."ruolo" and
 (NEW."dirigente" is null or NEW."dirigente" = false)) THEN
INSERT INTO t_carriera (codImpiegato,oldRuolo, newRuolo,
dirigente, dataScatto)
VALUES(NEW.codice_impiegato,
OLD.ruolo,
NEW.ruolo,
false,
CURRENT_DATE);
elseif (NEW."dirigente" is not null and
NEW."dirigente" = true ) then
INSERT INTO t_carriera (codImpiegato,oldRuolo, newRuolo,
dirigente, dataScatto)
VALUES(NEW.codiceImpiegato,
OLD.ruolo,
NEW.ruolo,
true,
CURRENT_DATE);
end if;
RETURN new;
END $function$
;


CREATE TRIGGER writeCareer
AFTER insert or UPDATE ON t_impiegati
FOR EACH ROW
```

```
EXECUTE FUNCTION checkCareer();
```

3. TRIGGER CHECK INSERIMENTO PROGETTO

È stato creato un trigger che agisce prima dell'inserimento di un progetto e/o prima della modifica delle informazioni del progetto. Il trigger richiama una funzione che controlla:
a) Nel caso di insert, il numero di laboratori associati allo stesso progetto. La funzione ritornerà quindi errore ('Too Many Lab') se si prova ad associare più di 3 laboratori ad uno stesso progetto.
b) Il referente scientifico del progetto deve essere un impiegato con ruolo 'SENIOR'. La funzione ritornerà quindi errore (Referente Scientifico non è SENIOR) se si prova ad inserire come referente scientifico un impiegato che non ha il ruolo 'SENIOR'.
c) Il responsabile dirigente deve essere un impiegato con la qualifica di dirigente. La funzione ritornerà quindi errore (Responsabile dirigente non è DIRIGENTE) se si prova ad inserire come responsabile dirigente un impiegato che non ha la qualifica di dirigente.

```
CREATE OR REPLACE FUNCTION checkProgetto()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
begin
if  (TG_OP = 'INSERT') then
if ( select distinct count(idLab) from t_progetti p where
p.CUP = new.CUP) = 3
THEN raise exception 'Too Many Lab';
end if;
end if;
if ( select ruolo from t_impiegati where codiceImpiegato =
new.codReferente) !=  'SENIOR'
THEN raise exception 'Referente Scientifico non è SENIOR';
end if;
if ( select dirigente from t_impiegati where codiceImpiegato =
new.codDirigente) !=  true
THEN raise exception 'Responsabile dirigente non è DIRIGENTE';

end if;
return new;
END $function$;
```

```
CREATE TRIGGER insertProgetto BEFORE INSERT
 OR UPDATE ON t_progetti
FOR EACH ROW
EXECUTE PROCEDURE checkProgetto();
```

4. TRIGGER CHECK INSERIMENTO LABORATORIO

È stato creato un trigger che agisce prima di un inserimento di un laboratorio e/o prima della modifica delle informazioni di un laboratorio. Il trigger richiama una funzione che controlla:
a) Il responsabile scientifico del laboratorio deve essere un impiegato con ruolo 'SENIOR' La funzione ritornerà quindi errore (Responsabile Scientifico non è SENIOR) se si prova ad inserire come responsabile scientifico un impiegato che non ha il ruolo 'SENIOR'.
b) Il responsabile scientifico deve essere sempre lo stesso per uno specifico laboratorio. La funzione ritornerà quindi errore (Responsabile Scientifico diverso da quello già esistente per laboratorio) se si prova ad inserire come responsabile scientifico un codice diverso da quello già presente per uno specifico laboratorio.
c) Un impiegato non può essere anche il responsabile scientifico per uno specifico laboratorio. La funzione ritornerà quindi errore (Responsabile Scientifico ed impiegato non possono essere uguali) se si prova ad inserire lo stesso codice sia come responsabile scientifico che come impiegato.

```
CREATE OR REPLACE FUNCTION checkLaboratorio()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
begin
if ( select ruolo from t_impiegati where codiceImpiegato =
new.codResponsabile) !=  'SENIOR'
THEN raise exception 'Responsabile Scientifico non è SENIOR';

end if;
if ( select count(*) from t_laboratori where codResponsabile !=
new.codResponsabile and idlab=new.idLab) > 0
THEN raise exception 'Responsabile Scientifico diverso da
quello già esistente per laboratorio';
end if;
if ( new.codResponsabile = new.codImpiegato )
THEN raise exception 'Responsabile Scientifico ed impiegato
```

```
non possono essere uguali';
end if;
return new;
END $function$;

CREATE TRIGGER insertLaboratorio BEFORE INSERT
OR UPDATE ON t_laboratori
FOR EACH ROW
EXECUTE PROCEDURE checkLaboratorio();
```

INSERT INTO TABELLE

Poiché queste 4 classi sono in relazione tra di loro tramite foreign key, l'eventuale svuotamento di queste deve avvenire secondo il giusto ordine:

TRUNCATE TABLE T_CARRIERA;
TRUNCATE TABLE T_PROGETTI;
TRUNCATE TABLE T_LABORATORI;
TRUNCATE TABLE T_IMPIEGATI;

```
INSERT INTO t_impiegati
(codiceImpiegato, nome, cognome, dataNascita,
codiceFiscale, dataAssunzione,  dirigente)
VALUES('A0001', 'Federica', 'Mele', '2003-03-25',
'MLEFRC03C65F839Z', '2021-03-26', true);
INSERT INTO t_impiegati
(codiceImpiegato, nome, cognome, dataNascita,
codiceFiscale, dataAssunzione,  dirigente)
VALUES('A0002', 'Antonio', 'Marselli', '2001-06-11',
'MRSANTXXXXXXXXX', '2019-06-12', true);
INSERT INTO t_impiegati
(codiceImpiegato, nome, cognome, dataNascita,
codiceFiscale, dataAssunzione,  dirigente)
VALUES('A0003', 'Mario', 'Rossi', '1973-09-24',
'RSSIMRCXXXXXXXXX', '1991-09-25', false);
INSERT INTO t_impiegati
(codiceImpiegato, nome, cognome, dataNascita,
codiceFiscale, dataAssunzione,  dirigente)
VALUES('A0004', 'Alice', 'Bianchi', '1984-09-16',
'BCHILCEXXXXXXXXX', '2002-09-17', false);

INSERT INTO t_laboratori
(idLab, codImpiegato, codResponsabile, topic)
VALUES(2, 'A0001', 'A0003', 'radiografia');


INSERT INTO t_progetti
```

```
(CUP, nome, idLab, codDirigente,
 codReferente)
VALUES('R0300', 'Radiologia', 2, 'A0001', 'A0003');
```