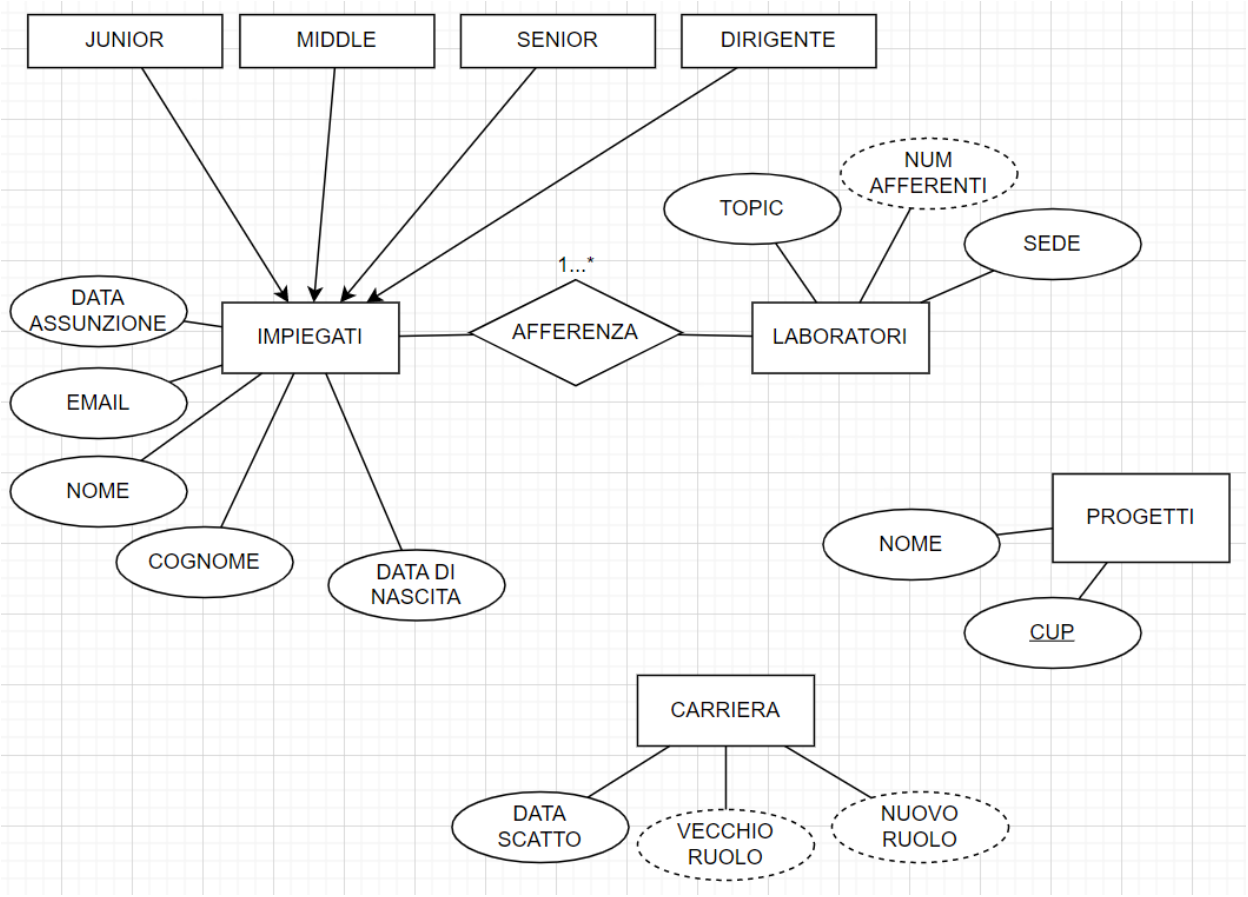
Progetto Azienda

Federica Mele, Antonio Marselli

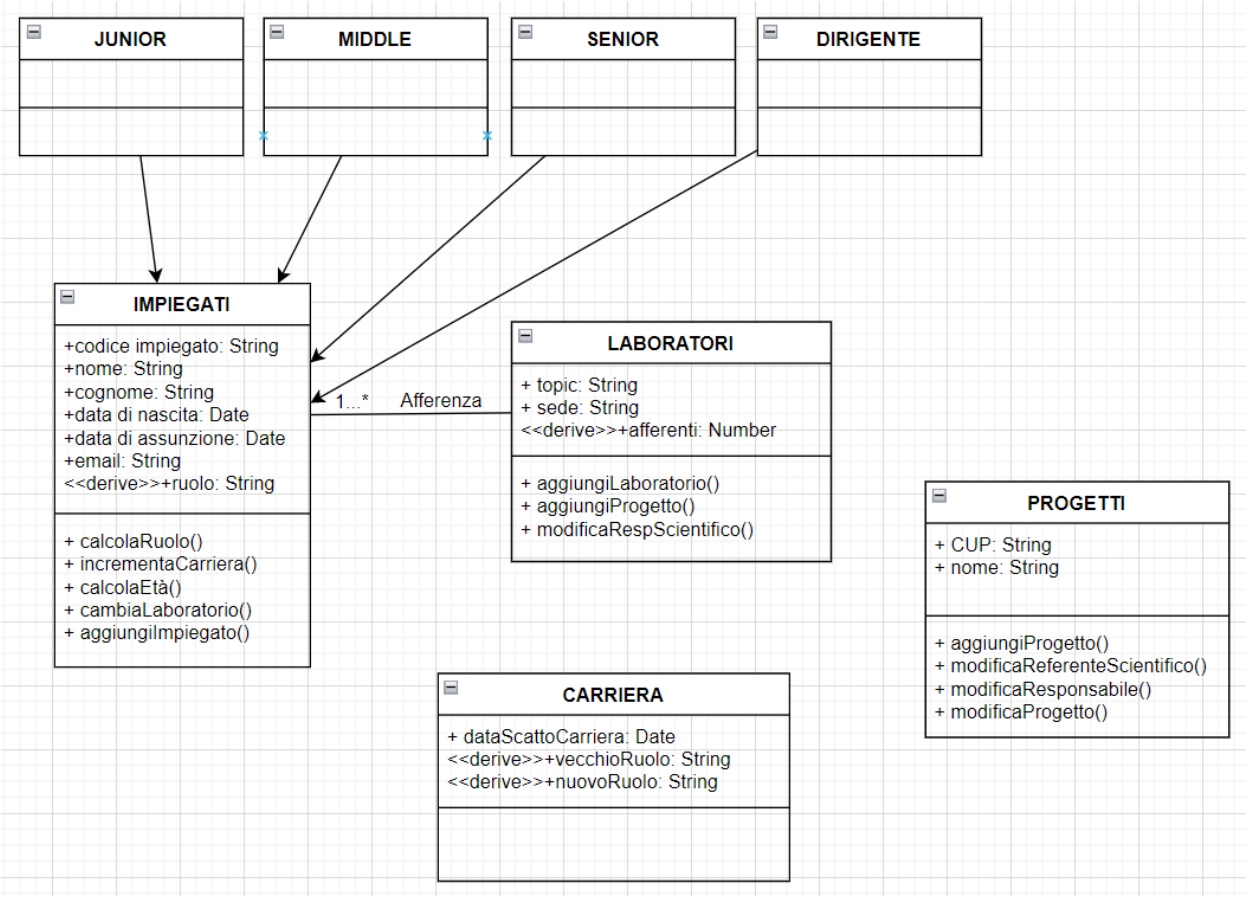
SISTEMA DI GESTIONE DEL PERSONALE E DEI PROGETTI ALL'INTERNO DI UN'AZIENDA

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



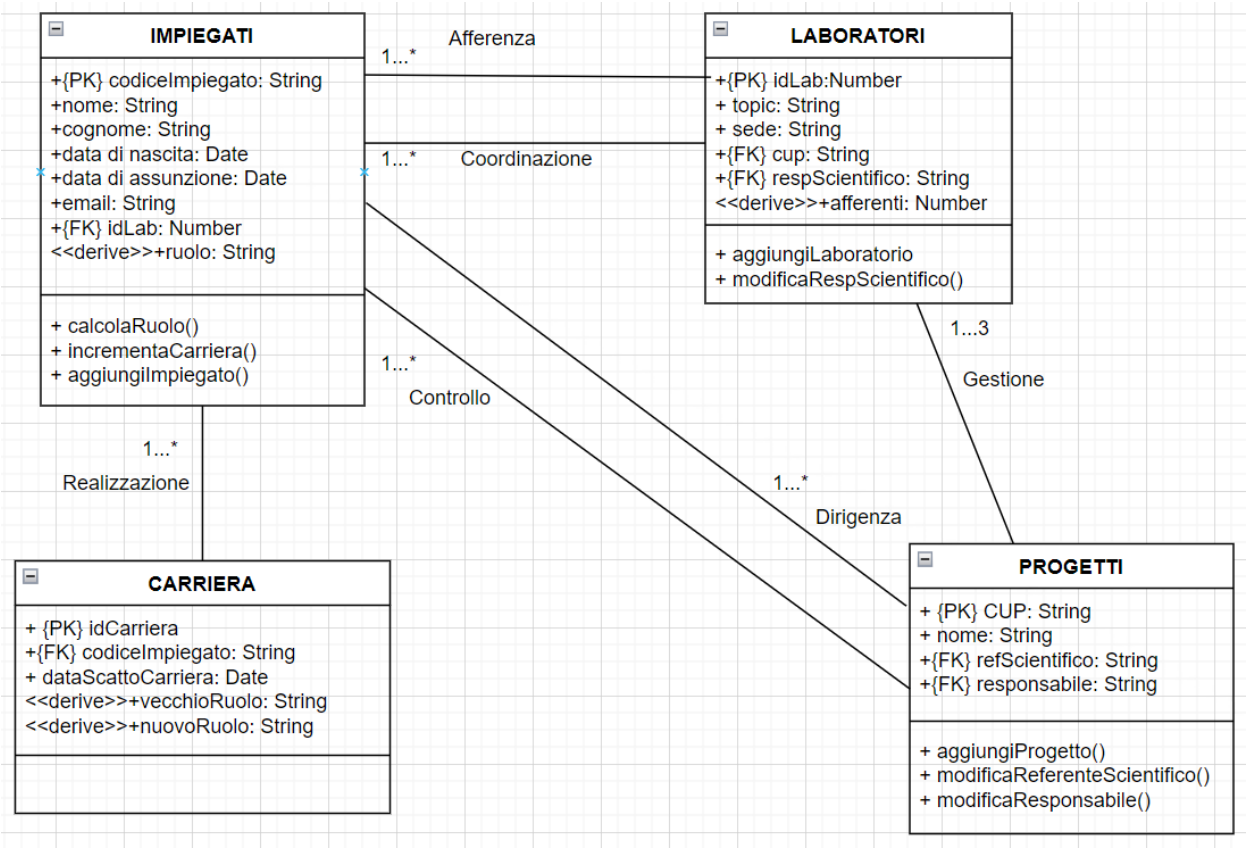
UML NON RISTRUTTURATO



DESCRIZIONE:
DA NON RISTRUTTURATO A RISTRUTTURATO

Una buona ristrutturazione di un diagramma migliora la chiarezza e riduce la complessità di un modello.
Per la ristrutturazione di un database è necessario fare un'analisi:
eliminare eventuali ridondanze, ossia tutti quei dati, superflui, presenti nel diagramma.
Durante la ristrutturazione abbiamo effettuato una generalizzazione, eliminando, in questo caso, le quattro classi (o entità) utilizzate per la rappresentazione dei ruoli dell'impiegato ('Junior', 'Middle', 'Senior' e 'Dirigente'), accorpandole alla classe padre ('Impiegato').
In seguito, abbiamo analizzato gli attributi presenti in ogni classe, per individuare eventuali ridondanze (o attributi composti) ed effettuare, quindi, una normalizzazione.
Abbiamo assegnato le primary key (chiavi primarie) e le foreign key (chiavi esterne) per garantire la coerenza dei dati e verificato che il modello non violi i vincoli dell'integrità referenziale.
Di conseguenza:
- All'interno della classe 'Impiegato' abbiamo reso il 'codiceImpiegato' una primary key che rappresenta l'identificativo univoco di un impiegato. Abbiamo inserito l'attributo 'IdLab' come foreign key, che mette in relazione (con il nome di 'Afferenza') la classe 'Impiegati' con la classe 'Laboratori'.
- All'interno della classe 'Carriera' abbiamo creato l'attributo IdCarriera e lo abbiamo reso una primary key. Abbiamo creato la relazione 'Realizzazione', tra le classi 'Carriera' ed 'Impiegati', inserendo l'attributo 'codiceImpiegato' come foreign key nella classe 'Carriera'.
- All'interno della classe 'Laboratori' abbiamo posto l'attributo 'IdLab' come primary key, per identificare un laboratorio. All'interno della classe 'Laboratori' troviamo due foreign key: l'attributo CUP (Codice Unico Progetto), che lega la classe in questione con la classe 'Progetti' creando la relazione di nome 'Gestione', e l'attributo 'RespScientifico' che corrisponde al 'codiceImpiegato' della classe 'Impiegato', fornendo una nuova relazione, di nome 'Coordinazione'.
- All'interno della classe 'Progetti' abbiamo reso l'attributo 'CUP' una primary key ed abbiamo creato le seguenti relazioni tra la classe 'Progetti' e la classe 'Impiegati' attraverso le foreign key formate dall'attributo 'RefScientifico', che crea la relazione di 'Dirigenza', e dall'attributo 'Responsabile', creando la relazione nominata Controllo'. Queste corrispondono, entrambe, al codiceImpiegato della classe 'Impiegati.

UML RISTRUTTURATO



MODELLO LOGICO

Entità e Attributi:

*1. Impiegati (CodiceImpiegato, Nome, Cognome, CodiceFiscale, DataNascita, DataAssunzione, Ruolo, Email, *IdLab*)*
*2. Carriera (IdCarriera, DataScatto, NuovoRuolo, VecchioRuolo, *codImpiegato*)*
*3. Laboratori (IdLab, Topic, *RespScientifico*, Sede, Afferenti)*
*4. Progetti (CUP, Nome, *RefScientifico*, *Responsabile*)*

Relazioni:
- *La relazione 'Afferenza', tramite 'IdLab', tiene traccia del numero degli impiegati che frequentano un laboratorio (Impiegati-Laboratori).*
- *La relazione 'Coordinazione' mostra il responsabile scientifico di un laboratorio, tramite 'RespScientifico'(Impiegati-Laboratori).*
- *La relazione 'Realizzazione' rappresenta lo storico della carriera di un impiegato, utilizzando 'codiceImpiegato', (Impiegati-Carriera).*
- *La relazione 'Dirigenza', tramite 'RefScientifico' mostra il referente scientifico di un progetto (Impiegati-Progetto).*
- *La relazione 'Controllo', tramite 'Responsabile' mostra il responsabile di un progetto (Impiegati-Progetti).*
- *La relazione 'Gestione', tramite 'CUP', tiene traccia dei laboratori che gestiscono un determinato progetto (Laboratori-Progetti).*

Note:
- *Le primary key sono rappresentate dalla sottolineatura.*
- *Le foreign key sono rappresentate tra due asterischi (*...*).*

TABELLE

1. IMPIEGATI

Struttura della tabella 't_Impiegati':

```
CREATE TABLE public.t_impiegati (
codiceimpiegato varchar NOT NULL,
datanascita date NOT NULL,
nome varchar NOT NULL,
cognome varchar NOT NULL,
codicefiscale varchar NOT NULL,
dataassunzione date NOT NULL,
ruolo varchar NULL,
idlab int8 NULL,
email varchar NULL,
CONSTRAINT t_impiegati_cf_unique UNIQUE (codicefiscale),
CONSTRAINT t_impiegati_check CHECK
((date_part('year'::text,
age((dataassunzione)::timestamp with time zone,
(datanascita)::timestamp with time zone)) > (17)::double precision)),
CONSTRAINT t_impiegati_codicefiscale_check CHECK
((char_length((codicefiscale)::text) = 16)),
CONSTRAINT t_impiegati_datanascita_check CHECK
((date_part('year'::text,
age((CURRENT_DATE)::timestamp with time zone,
(datanascita)::timestamp with time zone)) < (65)::double precision)),
CONSTRAINT t_impiegati_pk PRIMARY KEY (codiceimpiegato),
CONSTRAINT t_impiegati_lab_fk FOREIGN KEY
(idlab) REFERENCES public.t_laboratori(idlab)
);
```

Nella creazione della tabella t_impiegati sono stati introdotti:

a) Primary Key
Il campo codiceimpiegato è la chiave primaria della tabella t_impiegati, rappresenta l'identificativo univoco di un impiegato.

b) Foreign Key
Il campo idlab deve essere presente nella tabella t_laboratori.

c) Vincolo di unicità sul CF
Sul campo codicefiscale nella tabella t_impiegati avviene un controllo, in quanto non si deve poter inserire un codice fiscale già esistente in tabella.

d) 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.

e) 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.

f) Vincolo di validazione del CF:
Il codice fiscale deve essere di 16 caratteri.

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

ALTER TABLE public.t_impiegati ADD CONSTRAINT t_impiegati_pk PRIMARY KEY (codiceimpiegato)

ALTER TABLE public.t_impiegati ADD CONSTRAINT t_impiegati_cf_unique UNIQUE (codicefiscale)

ALTER TABLE public.t_impiegati ADD CONSTRAINT t_impiegati_codicefiscale_check CHECK ((char_length((codicefiscale)::text) = 16))

ALTER TABLE public.t_impiegati ADD CONSTRAINT t_impiegati_check CHECK ((date_part('year'::text, age((dataassunzione)::timestamp with time zone, (datanascita)::timestamp with time zone)) ¿ (17)::double precision))

ALTER TABLE public.t_impiegati ADD CONSTRAINT t_impiegati_datanascita_check CHECK ((date_part('year'::text, age((CURRENT_DATE)::timestamp with time zone, (datanascita)::timestamp with time zone)) ¡ (65)::double precision))

ALTER TABLE public.t_impiegati ADD CONSTRAINT t_impiegati_lab_fk FOREIGN KEY (idlab) REFERENCES t_laboratori(idlab)

2. LABORATORI

```
Struttura della tabella 't_laboratori':

CREATE TABLE public.t_laboratori (
idlab int4 NOT NULL,
topic varchar NOT NULL,
respscientifico varchar NOT NULL,
sede varchar NULL,
cup varchar NULL,
afferenti int4 NULL DEFAULT 0,
CONSTRAINT t_laboratori_pk PRIMARY KEY (idlab),
CONSTRAINT t_laboratori_cup_fk FOREIGN KEY (cup)
REFERENCES public.t_progetti(cup),
CONSTRAINT t_laboratori_resp_fk FOREIGN KEY (respscientifico)
REFERENCES public.t_impiegati(codiceimpiegato)
);
```

Nella creazione della tabella laboratori sono stati introdotti:

a) Primary key
Il campo idlab è la chiave primaria della tabella t_laboratori, rappresenta l'identificativo univoco di un laboratorio.

b) Foreign Key
Il campo cup deve essere presente nella tabella t_progetti.

c) Foreign Key
Il campo respscientifico deve essere presente nella tabella t_impiegati.

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

```
ALTER TABLE public.t_laboratori ADD CONSTRAINT t_laboratori_pk
PRIMARY KEY (idlab)
```

```
ALTER TABLE public.t_laboratori ADD CONSTRAINT t_laboratori_cup_fk
FOREIGN KEY (cup) REFERENCES t_progetti(cup)
```

```
ALTER TABLE public.t_laboratori ADD CONSTRAINT t_laboratori_resp_fk
FOREIGN KEY (respscientifico) REFERENCES t_impiegati(codiceimpiegato)
```

3. PROGETTI

```
Struttura della tabella 't_progetti'

CREATE TABLE public.t_progetti (
cup varchar NOT NULL,
nome varchar NOT NULL,
refscientifico varchar NOT NULL,
responsabile varchar NOT NULL,
CONSTRAINT t_progetti_nome_unique UNIQUE (nome),
CONSTRAINT t_progetti_pk PRIMARY KEY (cup),
CONSTRAINT t_progetti_ref_fk FOREIGN KEY (refscientifico)
REFERENCES public.t_impiegati(codiceimpiegato),
CONSTRAINT t_progetti_resp_fk FOREIGN KEY (responsabile)
REFERENCES public.t_impiegati(codiceimpiegato)
);
```

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

Il campo CUP è la chiave primaria della tabella t_progetti, rappresenta l'identificativo univoco di un progetto.

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

c) Foreign key
i campi refscientifico e responsabile devono essere presenti nella tabella t_impiegati.

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

ALTER TABLE public.t_progetti ADD CONSTRAINT t_progetti_pk
PRIMARY KEY (cup)

ALTER TABLE public.t_progetti ADD CONSTRAINT t_progetti_nome_unique
UNIQUE (nome)

ALTER TABLE public.t_progetti ADD CONSTRAINT t_progetti_ref_fk
FOREIGN KEY (refscientifico) REFERENCES t_impiegati(codiceimpiegato)

ALTER TABLE public.t_progetti ADD CONSTRAINT t_progetti_resp_fk
FOREIGN KEY (responsabile) REFERENCES t_impiegati(codiceimpiegato)

4. CARRIERA

```
Struttura della tabella 't_carriera'

CREATE TABLE public.t_carriera (
codimpiegato varchar NOT NULL,
oldruolo varchar NULL,
newruolo varchar NULL,
datascatto date NOT NULL,
idcarriera int8 NOT NULL,
CONSTRAINT t_carriera_pk PRIMARY KEY (idcarriera),
CONSTRAINT t_carriera_imp_fk FOREIGN KEY (codimpiegato)
REFERENCES public.t_impiegati(codiceimpiegato)
);
```

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

Il campo idCarriera è la chiave primaria. Per ciascun impiegato verrà tracciato il passaggio di carriera e/o la promozione a dirigente.

b) Foreign key

Il campo 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 public.t_carriera ADD CONSTRAINT t_carriera_pk PRIMARY KEY (idcarriera)

ALTER TABLE public.t_carriera ADD CONSTRAINT t_carriera_imp_fk FOREIGN KEY (codimpiegato) REFERENCES t_impiegati(codiceimpiegato)

TRIGGER E FUNZIONI

1. 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.
Nel caso di una modifica sul ruolo di senior di un impiegato, la funzione verificherà che lo stesso impiegato non svolga il ruolo di referente di un progetto e/o responsabile di un laboratorio, altrimenti restituirà l'eccezione 'Referente scientifico in un progetto' e/o 'Responsabile in un laboratorio'.
Nel caso di una modifica sul ruolo di dirigente di un impiegato, la funzione verificherà che lo stesso impiegato non svolga il ruolo di responsabile di un progetto, altrimenti restituirà l'eccezione 'Responsabile di un progetto'. La funzione valorizzerà quindi opportunamente il campo ruolo della tabella t_impiegati

```
create trigger writerole before
insert
or
update
on
public.t_impiegati for each row execute function checkrole();
```

```
CREATE OR REPLACE FUNCTION public.checkrole()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
begin

if(NEW."ruolo" is null or NEW."ruolo" != 'DIRIGENTE') 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;
end if;
if(OLD."ruolo" = 'SENIOR' and
(NEW."ruolo" is null or NEW."ruolo" != 'SENIOR')) then
if ( select distinct count(idlab) from t_laboratori p where
p.respScientifico = new.codiceimpiegato) > 0
THEN raise exception 'Responsabile scientifico in un laboratorio';
end if;
if ( select distinct count(cup) from t_progetti p where
p.refScientifico = new.codiceimpiegato) > 0
THEN raise exception 'Referente scientifico in un progetto';
end if;
end if;
if(OLD."ruolo" = 'DIRIGENTE' and
(NEW."ruolo" is null or NEW."ruolo" != 'DIRIGENTE')) then
if ( select distinct count(cup) from t_progetti p where
p.responsabile = new.codiceimpiegato) > 0
THEN raise exception 'Responsabile in un progetto';
end if;
end if;
RETURN new;
END $function$
;
```

2. CALCOLO AFFERENZA

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 numero di afferenti in un laboratorio.
La funzione valorizzerà quindi opportunamente il campo afferenti della tabella t_laboratori

```
create trigger checklabafferenti before
insert
or
update
on
public.t_impiegati for each row execute function calcolalabafferenti();

CREATE OR REPLACE FUNCTION public.calcolalabafferenti()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
begin
if  (TG_OP = 'INSERT') then
update t_laboratori set afferenti=1 +
(select afferenti from t_laboratori where
idlab = new.idlab) where idlab = new.idlab;
end if;
if  (TG_OP = 'UPDATE' and
(new.idlab != old.idlab or old.idlab is null
or new.idlab is null)) then
if  (new.idlab is not null) then
update t_laboratori set afferenti=1 +
(select afferenti from t_laboratori where
idlab = new.idlab) where idlab = new.idlab;
end if;
if  (old.idlab is not null) then
update t_laboratori set afferenti=
(select afferenti from t_laboratori where
idlab = old.idlab) -1 where idlab = old.idlab;
end if;
```

```
    end if;

    return new;
    END;$function$
    ;
```

3. 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.

```
create trigger writecareer after
insert
or
update
on
public.t_impiegati for each row execute function checkcareer();

CREATE OR REPLACE FUNCTION public.checkcareer()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
declare maxIdCarriera integer;
begin
select coalesce(MAX(idcarriera),0) into maxIdCarriera from t_carriera;
if ( OLD."ruolo" is null and NEW."ruolo" != 'DIRIGENTE') THEN
INSERT INTO t_carriera
(idcarriera, codImpiegato,oldRuolo, newRuolo, dataScatto)
VALUES(maxIdCarriera+1,
NEW.codiceimpiegato,
OLD.ruolo,
'JUNIOR',
(select dataAssunzione
from t_impiegati where
codiceimpiegato = NEW.codiceimpiegato));
if(NEW.ruolo = 'MIDDLE' or NEW.ruolo = 'SENIOR') THEN
INSERT INTO t_carriera
(idcarriera,codImpiegato,oldRuolo, newRuolo, dataScatto)
VALUES(maxIdCarriera+2,
NEW.codiceimpiegato,
'JUNIOR',
'MIDDLE',
(select dataAssunzione + interval '3 year'
```

```
from t_impiegati where
codiceImpiegato = NEW.codiceImpiegato));
end if;
if(NEW.ruolo = 'SENIOR') THEN
INSERT INTO t_carriera
(idcarriera,codImpiegato,oldRuolo, newRuolo, dataScatto)
VALUES(maxIdCarriera+3,
NEW.codiceimpiegato,
'MIDDLE',
'SENIOR',
(select dataAssunzione + interval '7 year'
from t_impiegati where
codiceimpiegato = NEW.codiceimpiegato));
end if;
elseif ( OLD.ruolo != NEW.ruolo) THEN
INSERT INTO t_carriera
(idcarriera,codImpiegato,oldRuolo, newRuolo, dataScatto)
VALUES(maxIdCarriera+1,
NEW.codiceimpiegato,
OLD.ruolo,
NEW.ruolo,
CURRENT_DATE);
elseif ( NEW."ruolo" = 'DIRIGENTE') THEN
INSERT INTO t_carriera
(idcarriera,codImpiegato,oldRuolo, newRuolo, dataScatto)
VALUES(maxIdCarriera+1,
NEW.codiceimpiegato,
OLD.ruolo,
'DIRIGENTE',
CURRENT_DATE);
end if;
RETURN new;
END $function$
;
```

4. 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) 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'.
b) Il responsabile dirigente deve essere un impiegato con ruolo 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 ha il ruolo di dirigente.

```
create trigger insertprogetto before
insert
or
update
on
public.t_progetti for each row execute function checkprogetto();

CREATE OR REPLACE FUNCTION public.checkprogetto()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
begin
if ( select ruolo from t_impiegati where codiceImpiegato =
new.refscientifico) !=  'SENIOR'
THEN raise exception 'Referente Scientifico non è SENIOR';
end if;
if ( select ruolo from t_impiegati where codiceImpiegato =
new.responsabile) !=  'DIRIGENTE'
THEN raise exception 'Responsabile dirigente non è DIRIGENTE';

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

5. 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) che il relativo progetto (CUP) non sia stato inserito in altri 3 diversi laboratori. In caso contrario la funzione ritornerà errore (Too many lab).
b) che il responsabile scientifico del laboratorio sia un impiegato con ruolo 'SENIOR'. In caso contrario la funzione ritornerà errore (Responsabile Scientifico non è SENIOR).

```
create trigger insertlaboratorio before
insert
or
update
on
public.t_laboratori for each row execute function checklaboratorio()

CREATE OR REPLACE FUNCTION public.checklaboratorio()
RETURNS trigger
LANGUAGE plpgsql
AS $function$
begin
if  (TG_OP = 'INSERT' or new.cup != old.cup) then
if ( select distinct count(idLab) from t_laboratori 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.respscientifico) != 'SENIOR'
THEN raise exception 'Responsabile Scientifico non è SENIOR';

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