Documentazione database
Sistema di gestione di personale e progetti all'interno di un'azienda

Roberto Ingenito Simone Ingenito Lorenzo Sequino

20 gennaio 2023

Indice

1 Progettazione concettuale

1.1 Analisi dei requisiti

La base dati si deve occupare della gestione del personale di un'azienda:
la figura centrale di tutto lo schema è l'impiegato.

La prima classificazione di un generico impiegato viene fatta tra:

- impiegato assunto regolarmente
- impiegato assunto esclusivamente per lavorare ad un progetto

Gli impiegati assunti regolarmente, vengono assunti con un determinato ruolo e stipendio, inoltre hanno diverse specializzazioni in base sia al merito sia agli anni trascorsi all'interno dell'azienda.

Dal momento in cui viene assunto, l'impiegato diventa automaticamente un junior. Trascorsi 3 anni diventa middle, trascorsi altri 4 diventa senior. Inoltre, qualunque siano gli anni trascorsi all'interno dell'azienda, può essere promosso e diventare manager.

Si tiene traccia di tutti gli scatti di carriera all'interno dell'entità *career_log* dove sono memorizzati:

- ruolo precedente
- ruolo successivo
- data dello scatto

L'azienda è divisa in varie sedi dove sono presenti diversi laboratori.

Ogni impiegato afferisce ad un unico laboratorio.
Un impiegato può aver anche lavorato in più laboratori ma in periodi diversi.
Al momento dell'assunzione l'impiegato non ha ancora una collocazione ma viene stabilita successivamente.
Ad un impiegato senior può essere assegnata la gestione di laboratori e/o di progetti.

Ad ogni laboratorio è assegnato un manager scientifico che è un impiegato senior.

L'amministratore, a cui è affidata la gestione del database, si occupa di:

- inserire gli impiegati assunti
- monitorare gli scatti di carriera
- promuovere gli impiegati meritevoli a manager
- creare progetti e affidare la supervisione ad un manager e la gestione ad un referente scientifico

Per ogni progetto viene stabilita una data di inizio, una *deadline* per la conclusione del progetto e dei fondi.

Inoltre, si tiene traccia dell'effettiva data in cui il progetto è terminato *end_date*; può accadere che il progetto si protragga oltre la deadline

Ad un progetto possono prendere parte massimo 3 laboratori contemporaneamente.
Ogni impiegato che afferisce ad un laboratorio prende parte automaticamente a tutti i progetti a cui quel laboratorio sta lavorando.

Il manager scientifico di un laboratorio decide a quali progetti partecipare e abbandonare, e richiede attrezzatura.

Le richieste vengono memorizzate nell'entità *equipment_request*. Queste vengono valutate dal manager e dal referente scientifico del progetto a cui sono state fatte, i quali decidono, in base ai fondi del progetto, se soddisfare le richieste, con il vincolo che il costo totale delle attrezzature non può superare il 50% dei fondi del progetto.

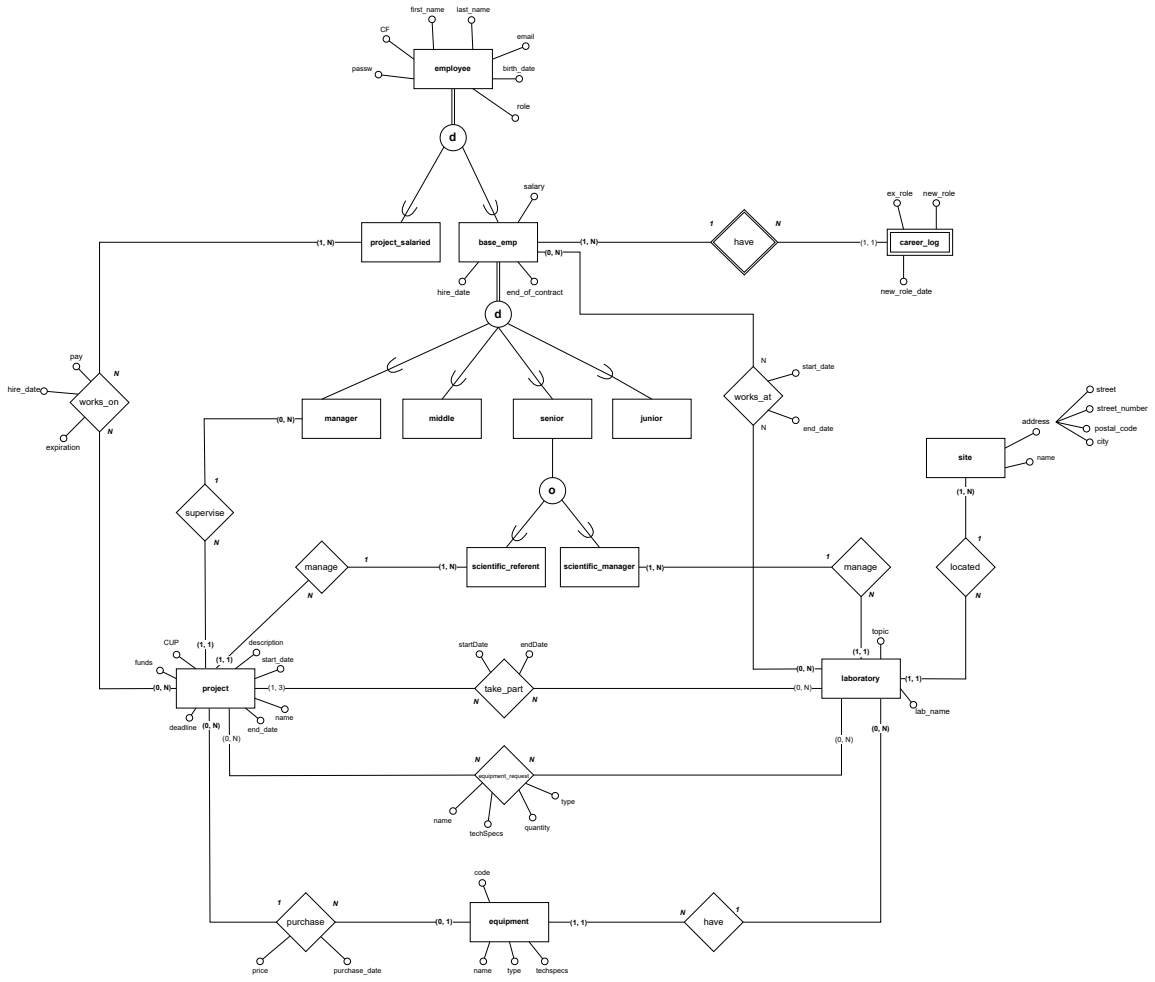
Quando viene acquistata dell'attrezzatura, si salva l'acquisto nell'entità *purchase* che tiene traccia della data e del prezzo d'acquisto.

Ogni attrezzatura di ogni laboratorio viene memorizzata nell'entità *equipment*.
Un laboratorio può possedere già dell'attrezzatura (acquistata precedentemente da altri progetti) e riceverne altra tramite richieste a progetti. Di conseguenza l'attrezzatura acquistata in un laboratorio, rimane anche dopo la fine del progetto.

Con il restante 50% dei fondi, il manager e il referente scientifico di un progetto possono assumere del personale che viene pagato per lavorare esclusivamente per quel progetto, e non viene considerato come impiegato regolare.

Lo stesso impiegato può essere assunto più volte (anche per lo stesso progetto ma in date differenti), per questo motivo ognuno di questi impiegati non viene rimosso una volta terminato il contratto. La relazione *works_on* infatti, tiene traccia di tutte le volte in cui un impiegato è stato assunto per lavorare ai progetti.

1.2 Schema concettuale



1.3 Dizionario delle entità

Entità	Descrizione	Attributi
employee	Generico impiegato dell'azienda	**cf**: Codice fiscale, informazione utile all'azienda **first_name** **last_name** **birth_date** **role**: Professione (sviluppatore, designer, …) **email**: Email necessaria per l'accesso al gestionale **passw**: Password necessaria per l'accesso al gestionale
base_emp	Specializzazione di *employee*. Assunto a tempo indeterminato.	**salary**: Stipendio **hire_date**: Data assunzione
career_log	Entità che memorizza gli scatti di carriera degli impiegati *base_emp*	**ex_role**: Ruolo precedente (stringa vuota quando è assunto) **new_role**: Ruolo successivo (stringa vuota quando è licenziato) **new_role_date**: Data scatto di carriera
project_salaried	Specializzazione di *employee*. Assunto per lavorare esclusivamente ad un progetto.	
junior	Specializzazione di *base_emp*. Impiegato che lavora da meno di 3 anni.	
middle	Specializzazione di *base_emp*. Impiegato che lavora dai 4 ai 7 anni.	
senior	Specializzazione di *base_emp*. Impiegato che lavora da più di 7 anni.	
manager	Specializzazione di *base_emp*. Impiegato che ha ricevuto una promozione in base al merito.	
scientific_referent	Specializzazione di *senior*. Impiegato a cui è stata affidata la gestione di un progetto.	
scientific_manager	Specializzazione di *senior*. Impiegato a cui è stata affidata la gestione di un laborfatorio.	

project	Progetto a cui prendono parte i laboratori e i relativi impiegati.	**CUP**: Codice Univoco Progett **name**: Nome del progetto **description**: Descrizione del progetto **start_date**: Data di inizio del progetto **end_date**: Data di effettiva fine del progetto **deadline**: Data prevista per la fine del progetto **funds**: Somma di denaro disponibile per l'acquisto di attrezzatura e personale
laboratory	Luogo all'interno dell'azienda nel quale gli impiegati lavorano ai progetti.	**topic**: Campo di studi del laboratorio **lab_name**: Nome del laboratorio
site	Sede dell'azienda in cui possono essere presente i laboratori.	**name**: Nome della sede **address**: Indirizzo • **street**: Via • **street_number**: Numero civico • **postal_code**: CAP • **city**: Città
equipment	Attrezzatura presente all'interno di un laboratorio	**code**: Codice del prodotto **name**: Nome del prodotto **type**: Tipo di prodotto (computer, microscopio, …) **techspecs**: Specifiche tecniche del prodotto

1.4 Dizionario delle associazioni

Associazioni	Descrizione	Attributi
works_on	Associazione tra *project_salaried* e *project* **molti-a-molti**	**pay**: Paga stabilita per lavorare a quel progetto **hire_date**: Data di assunzione **expiration**: Data di fine contratto
works_at	Associazione tra *base_emp* e *laboratory* **molti-a-molti**.	**start_date**: Data di assegnazione di un impiegato ad un laboratorio **end_date**: Data di fine lavoro
supervise	Associazione tra *manager* e *project* **uno-a-molti**.	
manage	Associazione tra *scientific_referent* e *project* **uno-a-molti**.	
manage	Associazione tra *scientific_manager* e *laboratory* **uno-a-molti**.	
take_part	Associazione tra *project* e *laboratory* **molti-a-molti**.	**start_date**: Data in cui un laboratorio inizia a lavorare ad un progetto **end_date**: Data di fine lavoro
purchase	Associazione tra *project* e *equipment* **uno-a-molti**.	**purchase_date**: Data di acquisto **price**: Prezzo dell'attrezzatura
equipment_request	Associazione tra *project* e *laboratory* **molti-a-molti**.	
have	Associazione tra *equipment* e *laboratory* **uno-a-molti**.	
have	Associazione tra *base_emp* e *career_log* **uno-a-molti**.	
located	Associazione tra *site* e *laboratory* **uno-a-molti**.	

2 Ristrutturazione del modello concettuale

2.1 Analisi delle ridondanze

È presente una ridondanza nel campo *hire_date* di *base_emp* in quanto è possibile calcolarla dall'entità associata *career_log*

2.2 Eliminazione degli attributi multivalore

Non sono presenti attributi multivalore

2.3 Eliminazione degli attributi composti

L'attributo *address* di *site* è un attributo composto.

Siccome l'indirizzo è unico per ogni sede, si è deciso di accorpare gli attributi di *address* in *site*.

L'attributo *tech_specs* di *equipment* potrebbe essere scomposto in più attributi ma in questo caso non è necessario ai fini della traccia, quindi è più comodo memorizzarlo come un unico campo stringa.

2.4 Analisi delle generalizzazioni

Procediamo all'eliminazione delle generalizzazioni partendo dal basso:

1. Accorpamento di *scientific_referent* e *scientific_manager* in *senior* in quanto non avendo attributi non è possibile inserire valori nulli che occupano spazio inutile.
   Fatta tale scelta bisogna cambiare la cardinalità da $(1, n)$ a $(0, n)$ in entrambe le associazioni, poiché un *senior* può anche non essere nessuno dei due o uno soltanto (overlapping parziale)

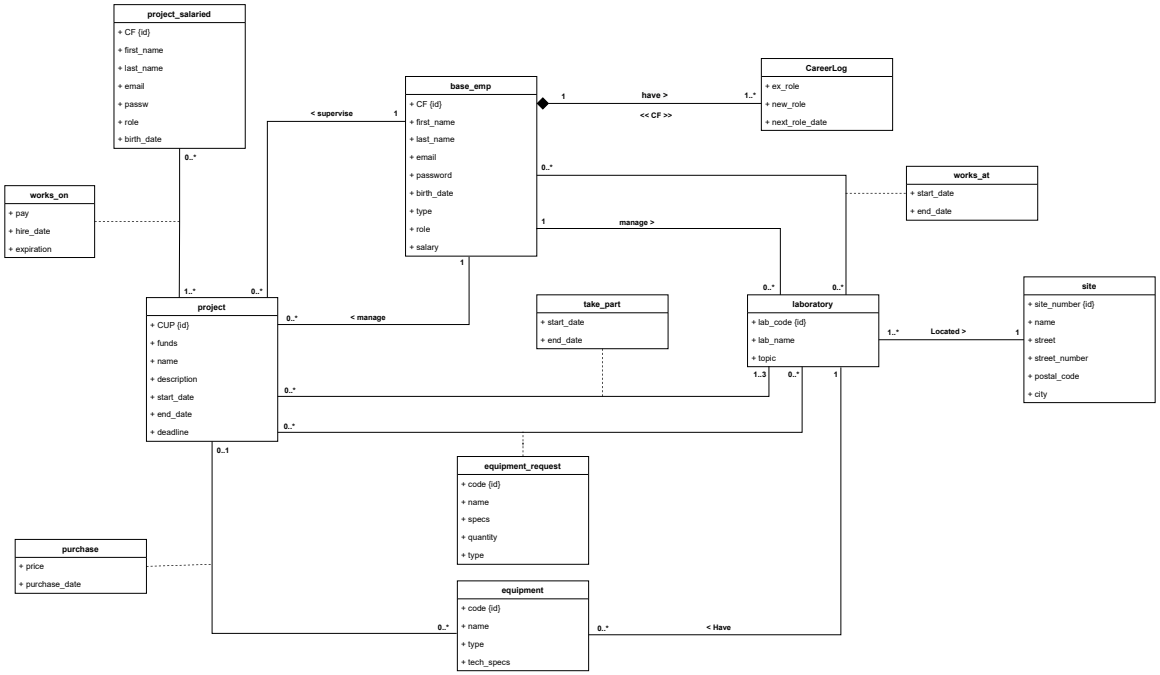
2. Accorpamento di *junior*, *middle*, *senior*, *manager* all'interno di *base_emp* aggiungendo un attributo che ne specifica il tipo.
   Notare che questa scelta aggiunge una **ridondanza** poiché il tipo è reperibile effettuando un'interrogazione in *career_log*. Nonostante ciò è più efficiente e immediato eseguire determinate operazioni avendo quest'attributo disponibile direttamente in *base_emp*

3. Accorpamento di *employee* all'interno di *project_salaried* e *base_emp* in quanto la generalizzazione è totale.

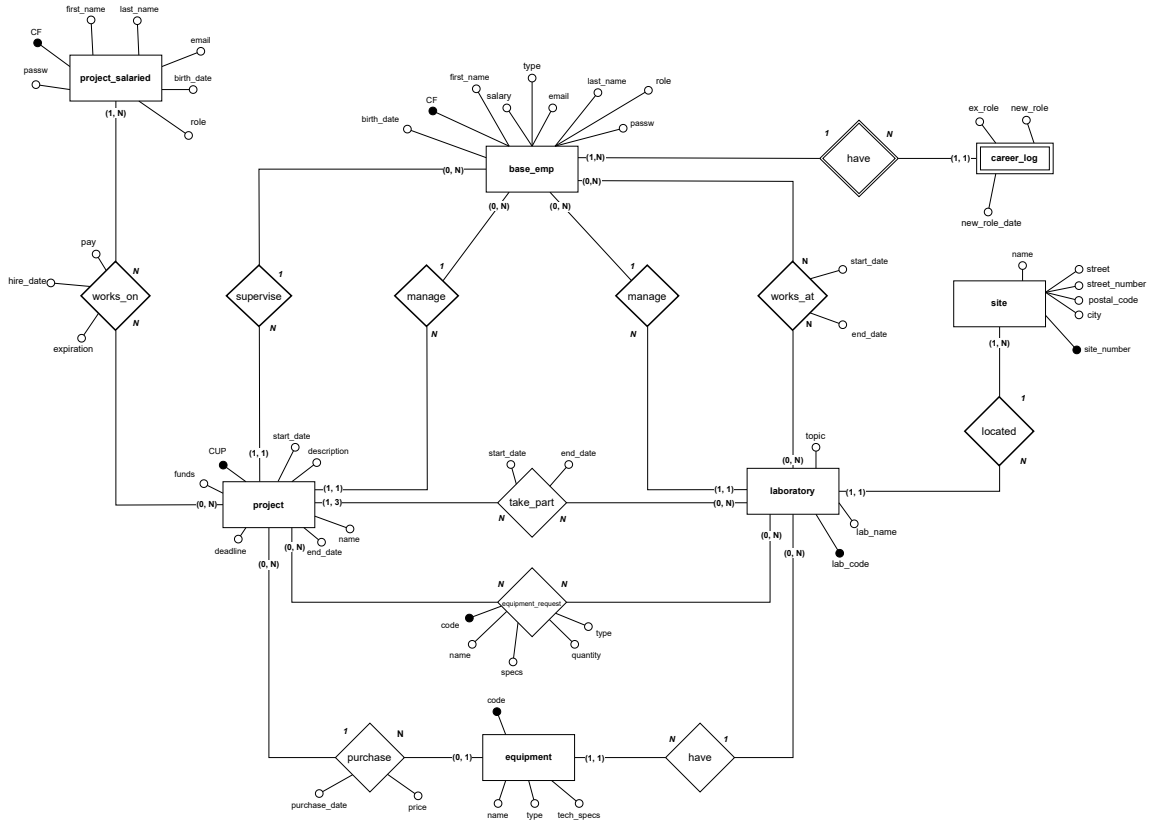
2.5 Identificazioni chiavi primarie

- In *base_emp* abbiamo l'attributo *CF* (Codice Fiscale), una stringa di 16 caratteri
- In *project_salaried* abbiamo l'attributo *CF* (Codice Fiscale), una stringa di 16 caratteri
- *project* è identificato dal *CUP* (Codice Univoco Progetto)
- *equipment* è identificato da *code* che è un codice generato randomicamente
- *equipment_request* è identificato da *code* che è un codice generato randomicamente
- *site* è identificato da un nuovo attributo *site_number* che si autoincrementa
- *laboratory* è identificato da un nuovo attributo *lab_code* che si autoincrementa

2.6 Schema ristrutturato UML



2.7 Schema ristrutturato ER



3 Traduzione al modello logico

3.1 Mapping associazioni

3.1.1 Associazioni 1-N

- *Site* - *Located* - *Laboratory*: inserimento chiave di *Site* in *Laboratory* come chiave esterna
- *base_emp* - *Supervise* - *Project*: inserimento chiave di *base_emp* all'interno di *Project* come chiave esterna
- *base_emp* - *Manage* - *Project*: inserimento chiave di *base_emp* all'interno di *Project* come chiave esterna
- *base_emp* - *Manage* - *Laboratory*: inserimento di chiave di *base_emp* all'interno di *Laboratory* come chiave esterna
- *Laboratory* - *Have* - *Equipment*: inserimento di chave di *Laboratory* in *Equipment* come chiave esterna
- *Project* - *Purchase* - *Equipment*: Equipment ha una partecipazione parziale, si procede come N-N
- *base_emp* - *Have* - *career_log*: relazione identificante, inserimento di chiave primaria di *base_emp* in *career_log* come chiave esterna

3.1.2 Associazioni N-N

Per ognuna di queste relazioni, si inseriscono le chiavi delle due associazioni come chiavi esterne.

Associazione	**Relazione**	**Associazione**
project_salaried	works_on	project
project	take_part	laboratory
project	equipment_request	laboratory
base_emp	works_at	laboratory

3.2 Modello logico

Gli attributi sottolineati sono chiavi primarie,
gli attributi con un asterisco * alla fine sono chiavi esterne

base_emp (CF, first_name, last_name, email, passw, birth_date, type, role, salary)

career_log (lab_code, lab_name, topic, cf_scientific_manager*, site_number*)
cf_scientific_manager ↦ base_emp.CF
site_number ↦ site.site_number

laboratory (lab_code, lab_name, topic, cf_scientific_manager*, site_number*)
cf_scientific_manager ↦ base_emp.CF
site_number ↦ site.site_number

take_part (start_date, end_date, CUP*, lab_code*)
CUP ↦ project.CUP
lab_code ↦ laboratory.lab_code

project_salaried (CF, first_name, last_name, email, passw, birth_date, role)

works_on (pay, hire_date, expiration, CF*, CUP*)
CF ↦ project_salaried.CF
CUP ↦ project.CUP

site (site_number, name, street, street_number, postal_code, city)

equipment (code, name, type, tech_specs, lab_code*)
lab_code ↦ laboratory.lab_code

purchase (purchase_date, price, CUP*, equipment_code*)
CUP ↦ project.CUP
equipment_code ↦ equipment.code

works_at (start_date, end_date, cf_base_emp*, lab_code*)
cf_base_emp ↦ base_emp.CF
lab_code ↦ laboratory.lab_code

equipment_request (code, type, name, specs, quantity, CUP*, lab_code*)
CUP ↦ project.CUP
lab_code ↦ laboratory.lab_code

project (CUP, funds, name, description, start_date, end_date,
deadline, cf_manager*,cf_scientific_referent*)
cf_manager ↦ base_emp.CF
cf_scientific_referent ↦ base_emp.CF

4 Progettazione fisica

4.1 Creazione del database

```
CREATE DATABASE company WITH OWNER postgres;
```

4.2 Creazione dello schema

```
CREATE SCHEMA projects_schema;
```

4.3 Attivazione estensioni

Le estensioni possono essere attivate solo da un superuser (*postgres* in questo caso); *project_admin* non può essere superuser altrimenti avrebbe gli stessi privilegi di *postgres*.

Le estensioni vengono create sullo schema *projects_schema* così che l'utente *project_admin* ne abbia l'accesso.

Estensione utilizzata per la cifratura delle password, usata per l'autenticazione lato client.

```
CREATE EXTENSION IF NOT EXISTS "pgcrypto" SCHEMA projects_schema;
```

Estensione utilizzata per generare codici univoci causali.

```
CREATE EXTENSION IF NOT EXISTS "uuid-ossp" SCHEMA projects_schema;
```

4.4 Gestione ruoli e permessi

All'utente *project_admin* è affidata la gestione dello schema *projects_schema*. Si occuperà di tutto, dalla creazione delle tabelle ai trigger …
Non essendo possessore dello schema, gli è impossibile cancellarlo.

Il numero massimo di connessioni è 1.

```
CREATE USER project_admin WITH
    PASSWORD 'proj_admin'
    CREATEROLE
    CONNECTION LIMIT 1;

GRANT ALL PRIVILEGES ON SCHEMA projects_schema TO project_admin;
```

Utente utilizzato esclusivamente per effettuare l'accesso lato client.
Ha accesso in lettura solo in *base_emp* e *project_salaried* perché ha bisogno di *email* e *password*

```
CREATE USER login_user WITH
        PASSWORD 'login';

GRANT SELECT ON base_emp, project_salaried TO login_user;
```

Ruolo generico che accorpa permessi necessari a tutti gli utenti che ne ”ereditano” il ruolo.

```
CREATE ROLE base_emp_role;

GRANT USAGE ON SCHEMA projects_schema TO base_emp_role;

GRANT SELECT ON base_emp, career_log, works_at, laboratory, take_part, equipment, site
             TO base_emp_role;

GRANT SELECT(CUP, name, description, start_date, end_date, deadline) ON project
             TO base_emp_role;
```

junior_user e *middle_user* non hanno ulteriori permessi specifici.

```
CREATE USER junior_user WITH
        PASSWORD 'junior'
        IN ROLE base_emp_role;

CREATE USER middle_user WITH
        PASSWORD 'middle'
        IN ROLE base_emp_role;
```

L'utente *senior_user* accorpa i permessi di *scientific_referent* e *scientific_manager*.

```
CREATE USER senior_user WITH
        PASSWORD 'senior'
        IN ROLE base_emp_role;

-- Possibilità di prendere parte ad un progetto e richiedere attrezzatura
GRANT INSERT ON take_part, equipment_request TO senior_user;

-- Possibilità di abbandonare un progetto
GRANT UPDATE(end_date) ON take_part TO senior_user;

-- Modificare la richiesta di attrezzatura
GRANT UPDATE(specs, quantity) ON equipment_request TO senior_user;

-- il manager di un progetto ne imposta la fine
GRANT UPDATE(end_date) ON project TO manager_user;
```

```
CREATE USER manager_user WITH
        PASSWORD 'manager'
        IN ROLE base_emp_role;
```

Questi permessi sono condivisi sia dallo *scientific_referent* che dal *manager*

```
GRANT SELECT ON project,
                purchase,
                equipment_request,
                project_salaried,
                works_on,
                remaining_project_funds
             TO manager_user, senior_user;

GRANT INSERT ON equipment,
                purchase,
                project_salaried,
                works_on
```

```
            TO manager_user, senior_user;

GRANT DELETE ON equipment_request TO manager_user, senior_user;
```

Non essendo *base_emp*, non ne eredita il ruolo ma può visualizzare solo le informazioni dei progetti a cui lavora.

```
CREATE USER project_salaried_user WITH
       PASSWORD 'proj_sal';

GRANT USAGE ON SCHEMA projects_schema TO project_salaried_user;

GRANT SELECT ON project_salaried, works_on, project
            TO project_salaried_user;
```

4.5 Creazione domini

```
CREATE DOMAIN cf_type AS VARCHAR(16)
CONSTRAINT cf_type_length CHECK (length(VALUE) = 16);
```

```
CREATE DOMAIN cup_type AS VARCHAR(15)
CONSTRAINT check_cup_type CHECK ( length(VALUE) = 15);
```

```
CREATE DOMAIN emp_type AS VARCHAR(10)
CONSTRAINT emp_type_check CHECK (VALUE IN ('', 'junior', 'middle', 'senior', 'manager'
    ) AND VALUE IS NOT NULL);
```

```
CREATE DOMAIN equipment_name_type
AS VARCHAR(30);
```

```
CREATE DOMAIN name_type AS VARCHAR(30);
```

```
CREATE DOMAIN password_type AS VARCHAR(100);
```

```
CREATE DOMAIN salary_type AS DECIMAL(8,2);
```

4.6 Creazione tabelle

Tabella base_emp

```
CREATE TABLE base_emp(
    CF          cf_type,
    first_name  name_type     NOT NULL,
    last_name   name_type     NOT NULL,
    email       VARCHAR(100)  NOT NULL,
    passw       password_type NOT NULL,
    birth_date  DATE          NOT NULL,
    "type"      emp_type      NOT NULL,
    "role"      VARCHAR(30)   NOT NULL,
    salary      salary_type   NOT NULL,

    CONSTRAINT emp_email_unique UNIQUE (email),
    CONSTRAINT base_emp_pk PRIMARY KEY (CF)
);
```

Tabella career_log

```
CREATE TABLE career_log(
    ex_role         emp_type,
    new_role        emp_type,
    new_role_date   TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    CF              cf_type,

    -- foreign key constraint
    CONSTRAINT emp_career_fk FOREIGN KEY (CF) REFERENCES base_emp(CF)
        ON DELETE CASCADE
        ON UPDATE CASCADE,

    CONSTRAINT check_new_grade CHECK (
        (ex_role, new_role) IN (
            ('', 'junior'), -- assunto
            ('junior', 'middle'),
            ('middle', 'senior')
        ) OR

        -- licenziato
        ex_role <> '' AND new_role = ''
        OR

        -- avanzamento a manger
        ex_role <> '' AND ex_role <> 'manager' AND new_role = 'manager'
        OR

        -- declassato
        --
        -- un trigger verificherà che il new_role sia quello del log precedente
        -- se da middle è diventato manager e poi viene declassato,
        --    sarà declassato a middle
        ex_role = 'manager' AND new_role <> 'manager'
    )
```

Tabella laboratory

```
CREATE TABLE laboratory(
    lab_code    SERIAL,
    lab_name    VARCHAR(200)    NOT NULL,
    topic       VARCHAR(1000)   NOT NULL,

    -- foreign keys
    site_number             SERIAL  NOT NULL,
    cf_scientific_manager   cf_type NOT NULL,

    CONSTRAINT lab_pk                   PRIMARY KEY (lab_code),

    CONSTRAINT site_number_fk           FOREIGN KEY (site_number)
                                        REFERENCES site(site_number),

    CONSTRAINT cf_scientific_manager_fk FOREIGN KEY (cf_scientific_manager)
                                        REFERENCES base_emp(cf)
);
```

Tabella take_part

```
CREATE TABLE take_part(
    start_date  DATE    NOT NULL,
    end_date    DATE,

    -- foreign keys
    CUP         cup_type,
    lab_code    SERIAL,

    CONSTRAINT date_integrity CHECK (end_date IS NULL OR end_date > start_date),

    CONSTRAINT CUP_fk FOREIGN KEY (CUP) REFERENCES project(CUP)
    ON UPDATE CASCADE
    ON DELETE CASCADE,

    CONSTRAINT lab_code_fk FOREIGN KEY (lab_code) REFERENCES laboratory(lab_code)
    ON UPDATE CASCADE
    ON DELETE CASCADE
);
```

Tabella project_salaried

```
CREATE TABLE project_salaried(
    CF          cf_type,
    first_name  name_type       NOT NULL,
    last_name   name_type       NOT NULL,
    email       VARCHAR(100)    NOT NULL,
    passw       password_type   NOT NULL,
    "role"      VARCHAR(30)     NOT NULL,
    birth_date  DATE            NOT NULL,

    CONSTRAINT project_salaried_email_unique UNIQUE (email),
    CONSTRAINT project_salaried_pk PRIMARY KEY (CF)
);
```

Tabella works_on

```
CREATE TABLE works_on (
    pay         salary_type NOT NULL,
    hire_date   DATE        NOT NULL,
    expiration  DATE        NOT NULL,

    -- foreign keys
    CF          cf_type,
    CUP         cup_type,

    CONSTRAINT check_expiration_date CHECK ( expiration > hire_date ),

    CONSTRAINT fk_cf FOREIGN KEY (CF) REFERENCES project_salaried(CF)
    ON UPDATE CASCADE

    -- mi interessa salvare i contratti anche se
    -- un impiegato a progetto viene eliminato
    ON DELETE SET NULL,

    CONSTRAINT fk_cup FOREIGN KEY (CUP) REFERENCES project(CUP)
    ON UPDATE CASCADE
);
```

Tabella site

```
CREATE TABLE site(
    site_number     SERIAL,
    name            VARCHAR(40) NOT NULL,
    street          VARCHAR(30) NOT NULL,
    street_number   VARCHAR(10) NOT NULL,
    postal_code     VARCHAR(10) NOT NULL,
    city            VARCHAR(20) NOT NULL,

    CONSTRAINT site_pk PRIMARY KEY (site_number)
);
```

Tabella equipment

```
CREATE TABLE equipment(
    code        uuid                DEFAULT uuid_generate_v4(),
    name        equipment_name_type NOT NULL,
    type        VARCHAR(30)         NOT NULL,
    tech_specs  VARCHAR(100),

    -- foreign keys
    lab_code    SERIAL,

    CONSTRAINT equipment_pk PRIMARY KEY (code),

    CONSTRAINT lab_code_fk FOREIGN KEY (lab_code) REFERENCES laboratory(lab_code)
    ON UPDATE CASCADE
    ON DELETE SET NULL
);
```

Tabella purchase

```
CREATE TABLE purchase(
    price           DECIMAL(10,2)   NOT NULL,
    purchase_date   DATE            NOT NULL,

    -- foreign keys
    CUP             cup_type,
    equipment_code  uuid            DEFAULT uuid_generate_v4(),

    CONSTRAINT cup_type_fk FOREIGN KEY (CUP) REFERENCES project(CUP)
        ON UPDATE CASCADE
        ON DELETE SET NULL,

    CONSTRAINT equipment_code_fk    FOREIGN KEY (equipment_code)
                                    REFERENCES equipment(code)
        ON UPDATE CASCADE
        ON DELETE CASCADE,

    CONSTRAINT check_price CHECK ( price > 0)
);
```

Tabella works_at

```
CREATE TABLE works_at(
    start_date  DATE    NOT NULL,
    end_date    DATE,

    -- foreign keys
    cf_base_emp cf_type,
    lab_code    SERIAL,

    CONSTRAINT date_integrity CHECK (end_date IS NULL OR end_date > start_date),

    CONSTRAINT cf_base_emp_fk FOREIGN KEY (cf_base_emp) REFERENCES base_emp(cf)
        ON UPDATE CASCADE
        ON DELETE CASCADE,

    CONSTRAINT lab_code_fk    FOREIGN KEY (lab_code) REFERENCES laboratory(lab_code)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
```

Tabella equipment_request

```
CREATE TABLE equipment_request(
    code     uuid                    DEFAULT uuid_generate_v4(),

    name     equipment_name_type     NOT NULL,
    specs    VARCHAR(100)            NOT NULL,
    type     VARCHAR(30)             NOT NULL,

    quantity INTEGER                 NOT NULL   DEFAULT 1,

    -- foreign keys
    CUP      cup_type,
    lab_code SERIAL,

    CONSTRAINT equipment_request_pk PRIMARY KEY (code),

    CONSTRAINT CUP_fk       FOREIGN KEY (CUP)      REFERENCES project(CUP)
        ON UPDATE CASCADE
        ON DELETE CASCADE,

    CONSTRAINT lab_code_fk  FOREIGN KEY (lab_code) REFERENCES laboratory(lab_code)
        ON UPDATE CASCADE
        ON DELETE CASCADE,

    CONSTRAINT check_quantity CHECK ( quantity > 0 )
);
```

Tabella project

```
CREATE TABLE project(
    CUP             cup_type,
    funds           DECIMAL(10,2)   NOT NULL,
    "name"          VARCHAR(50)     UNIQUE,
    description     VARCHAR(100)    NOT NULL,
    start_date      DATE            NOT NULL DEFAULT CURRENT_DATE,
    end_date        DATE,
    deadline        DATE,

    -- foreign key
    cf_manager              cf_type NOT NULL,
    cf_scientific_referent  cf_type NOT NULL,

    CONSTRAINT project_pk PRIMARY KEY (CUP),

    CONSTRAINT check_deadline CHECK ( deadline IS NULL OR deadline > start_date ),
    CONSTRAINT check_end_date CHECK ( end_date IS NULL OR end_date > start_date ),

    CONSTRAINT fk_cf_manager FOREIGN KEY (CF_manager) REFERENCES base_emp(CF)
        ON UPDATE CASCADE,

    CONSTRAINT fk_cf_scientific_referent    FOREIGN KEY (CF_scientific_referent)
                                            REFERENCES base_emp(CF)
        ON UPDATE CASCADE
);
```

4.7 Trigger e trigger function

La descrizione dei seguenti trigger è riportata nella sezione dei vincoli inter-relazionali

check_equipment_funds

```
CREATE OR REPLACE FUNCTION check_equipment_funds()
RETURNS TRIGGER
language 'plpgsql'
AS $$

DECLARE
    remaining_funds remaining_project_funds.funds_to_buy%TYPE;

BEGIN
    SELECT funds_to_buy INTO remaining_funds
    FROM remaining_project_funds
    WHERE CUP = NEW.CUP;

    IF remaining_funds - NEW.price < 0 THEN
        RAISE EXCEPTION 'not enough funds to buy equipment';
    END IF;

    RETURN NEW;
END $$;

CREATE OR REPLACE TRIGGER check_equipment_funds
BEFORE INSERT ON purchase
FOR EACH ROW
EXECUTE FUNCTION check_equipment_funds();
```

check_funds_to_hire

```
CREATE OR REPLACE FUNCTION check_funds_to_hire()
RETURNS TRIGGER
language 'plpgsql'
AS $$

DECLARE
    remaining_funds remaining_project_funds.funds_to_hire%TYPE;

BEGIN
    SELECT funds_to_hire INTO remaining_funds
    FROM remaining_project_funds
    WHERE CUP = NEW.CUP;

    IF remaining_funds - NEW.pay < 0 THEN
        RAISE EXCEPTION 'not enough funds to hire';
    END IF;

    RETURN NEW;
END $$;

CREATE OR REPLACE TRIGGER check_funds_to_hire
BEFORE INSERT ON works_on
FOR EACH ROW
EXECUTE FUNCTION check_funds_to_hire();
```

check_laboratory_assignment

```
CREATE OR REPLACE FUNCTION check_laboratory_assignment()
RETURNS trigger
language 'plpgsql'
AS $$

DECLARE
    scientific_manager_type base_emp.type%TYPE := ''; -- emp_type del manager di
        laboratorio

BEGIN
    SELECT "type" INTO scientific_manager_type
    FROM base_emp
    WHERE cf = NEW.cf_scientific_manager;

    IF scientific_manager_type <> 'senior' THEN
        RAISE EXCEPTION '% is not senior', NEW.cf_scientific_manager;
    END IF;

    RETURN NEW;

    EXCEPTION
        -- Questo sqlstate si riferisce ad una violazione di un vincolo
        -- in questo caso se una delle due select non trova valori, assegna a
            scientific_manager_type il valore NULL
        -- sono di tipo emp_type il quale ha un vincolo che gli impedisce di essere
            NULL
        -- viene dunque lanciata l'eccezione
        WHEN SQLSTATE '23514' THEN
            RAISE EXCEPTION 'cf_scientific_manager does not exist';
END; $$;

CREATE TRIGGER check_laboratory_assignment
BEFORE INSERT OR UPDATE OF cf_scientific_manager ON laboratory
FOR EACH ROW
EXECUTE FUNCTION check_laboratory_assignment();
```

check_project_assignment

```
CREATE OR REPLACE FUNCTION check_project_assignment()
RETURNS trigger
language 'plpgsql'
AS $$

DECLARE
    referent_type base_emp.type%TYPE := ''; -- emp_type del referente di progetto
    manager_type base_emp.type%TYPE := ''; -- emp_type del supervisore del progetto

BEGIN
    SELECT "type" INTO referent_type
    FROM base_emp
    WHERE cf = NEW.cf_scientific_referent;

    SELECT "type" INTO manager_type
    FROM base_emp
    WHERE cf = NEW.cf_manager;


    IF referent_type <> 'senior' THEN
        RAISE EXCEPTION '% is not senior', NEW.cf_scientific_referent;
    END IF;

    IF manager_type <> 'manager' THEN
        RAISE EXCEPTION '% is not manager', NEW.cf_manager;
    END IF;


    RETURN NEW;

    EXCEPTION
        -- Questo sqlstate si riferisce ad una violazione di un vincolo
        -- in questo caso se una delle due select non trova valori, assegna a
            referent_type (o manager_type) il valore NULL
        -- sono di tipo emp_type il quale ha un vincolo che gli impedisce di essere
            null
        -- viene dunque lanciata l'eccezione
        WHEN SQLSTATE '23514' THEN
            RAISE EXCEPTION 'cf_scientific_referent or cf_manager does not exist';
END; $$;

CREATE TRIGGER check_project_assignment
BEFORE INSERT OR UPDATE OF cf_scientific_referent, cf_manager ON project
FOR EACH ROW
EXECUTE FUNCTION check_project_assignment();
```

check_works_on_date

```
CREATE OR REPLACE FUNCTION check_works_on_date()
RETURNS trigger
language 'plpgsql'
AS $$
DECLARE
    max_expiration works_on.expiration%TYPE;

BEGIN
    SELECT MAX(expiration) INTO max_expiration
    FROM works_on
    WHERE cf = NEW.cf AND cup = NEW.cup;

    IF max_expiration IS NOT NULL   AND   NEW.hire_date < max_expiration THEN
        RAISE EXCEPTION '% sta gia lavorando per il progetto %', NEW.cf, NEW.cup;
    END IF;

    RETURN NEW;
END; $$;

CREATE TRIGGER check_works_on_date
BEFORE INSERT ON works_on
FOR EACH ROW
EXECUTE FUNCTION check_works_on_date();
```

crypt_password

```
CREATE OR REPLACE FUNCTION crypt_password()
RETURNS trigger
language 'plpgsql'
AS $$
BEGIN
    NEW.passw := crypt(NEW.passw, gen_salt('md5'));

    RETURN NEW;
END; $$;

CREATE TRIGGER crypt_password_project_salaried
BEFORE INSERT ON project_salaried
FOR EACH ROW
EXECUTE FUNCTION crypt_password();

CREATE TRIGGER crypt_password_base_emp
BEFORE INSERT ON  base_emp
FOR EACH ROW
EXECUTE FUNCTION crypt_password();
```

employee_hiring

```
CREATE OR REPLACE FUNCTION employee_hiring()
RETURNS trigger
language 'plpgsql'
AS $$
BEGIN
    INSERT INTO career_log (ex_role, new_role, new_role_date, CF)
    VALUES ('', 'junior', CURRENT_TIMESTAMP, NEW.cf);

    RETURN NEW;
END; $$;


CREATE TRIGGER employee_hiring
AFTER INSERT ON base_emp
FOR EACH ROW
EXECUTE FUNCTION employee_hiring();
```

fire_emp

```
CREATE OR REPLACE FUNCTION fire_emp()
RETURNS trigger
language 'plpgsql'
AS $$

BEGIN
    -- se l'impiegato da licenziare è un referente scientifico oppure manager di
        progetto
    -- e il progetto è in corso (end_date NULL)
    -- OPPURE
    -- è manager scientifico di un laboratorio
    -- non posso licenziarlo, ma bisognerà prima cambiare
    IF
    EXISTS(
        SELECT *
        FROM project
        WHERE (cf_manager = NEW.cf OR cf_scientific_referent = NEW.cf)
                AND end_date IS NULL
    ) OR
    EXISTS(
        SELECT *
        FROM laboratory
        WHERE cf_scientific_manager = NEW.cf
    )
    THEN
        RAISE EXCEPTION '% lavora come manager o referente in un progetto o
            laboratorio', NEW.cf;
    END IF;

    -- aggiorna l'end date dei laboratori in cui lavora, alla data del licenziamento
    -- ciò significa che non lavora più in quel laboratorio
    UPDATE works_at
    SET end_date = NEW.new_role_date
    WHERE cf_base_emp = NEW.cf AND end_date IS NULL;

    RETURN NEW;
END; $$;

CREATE TRIGGER fire_emp
BEFORE INSERT ON career_log
FOR EACH ROW
WHEN (NEW.new_role = '')
EXECUTE FUNCTION fire_emp();
```

max_lab_in_project

```
CREATE OR REPLACE FUNCTION max_lab_in_project()
RETURNS trigger
language 'plpgsql'
AS $$

DECLARE
labs_number INTEGER := 0;

BEGIN
    -- se end_date non è null, significa che un progetto non lavora più in quel
        laboratorio
    -- quind posso inserire nella tabella
    IF NEW.end_date IS NOT NULL THEN
        RETURN NEW;
    END IF;

    -- conto in quanti laboratori lavora il progetto che voglio inserire
    -- quando end_date è null, il progetto sta ancora lavorando in quel laboratorio
    SELECT COUNT(*) INTO labs_number
    FROM take_part
    WHERE NEW.cup = cup AND end_date IS NULL;

    -- essendo un BEFORE INSERT, se lavora in più di 2 laboratori,
    -- allora lancia un eccezione e impedisce l'inserimento
    IF labs_number > 2 THEN
        RAISE EXCEPTION 'Al progetto lavorano gia 3 laboratori';
    END IF;
    RETURN NEW;
END $$;

CREATE TRIGGER max_lab_in_project
BEFORE INSERT ON take_part
FOR EACH ROW
EXECUTE FUNCTION max_lab_in_project();
```

update_emp_type

```
CREATE OR REPLACE FUNCTION update_emp_type()
RETURNS trigger
language 'plpgsql'
AS $$

BEGIN
    UPDATE base_emp
    SET "type" = NEW.new_role
    WHERE cf = NEW.cf;

    RETURN NEW;
END; $$;

CREATE TRIGGER update_emp_type
AFTER INSERT ON career_log
FOR EACH ROW
EXECUTE FUNCTION update_emp_type();
```

take_part_no_duplicates

```
CREATE OR REPLACE FUNCTION take_part_no_duplicates()
RETURNS trigger
language 'plpgsql'
AS $$

DECLARE
labs_number INTEGER := 0;

BEGIN
    -- se end_date non è null, significa che un progetto non lavora più in quel
        laboratorio
    -- quind posso inserire nella tabella
    IF NEW.end_date IS NOT NULL THEN
        RETURN NEW;
    END IF;

    -- conto quante volte il laboratorio che si vuole inserire
    -- lavora nello stesso progetto (end_date is null)
    SELECT COUNT(*) INTO labs_number
    FROM take_part
    WHERE NEW.cup = cup AND NEW.lab_code = lab_code AND end_date IS NULL;

    -- se il laboratorio lavora gia in quel progetto, impedisco l'inserimento
    IF labs_number > 0 THEN
        RAISE EXCEPTION 'Il laboratorio % lavora già al progetto %', NEW.lab_code, NEW
            .cup;
    END IF;
    RETURN NEW;
END $$;

CREATE TRIGGER take_part_no_duplicates
BEFORE INSERT ON take_part
FOR EACH ROW
EXECUTE FUNCTION take_part_no_duplicates();
```

validate_equipment_request

```
CREATE OR REPLACE FUNCTION validate_equipment_request()
RETURNS TRIGGER
language 'plpgsql'
AS $$

BEGIN
    IF
        NOT EXISTS(
            SELECT *
            FROM take_part
            WHERE end_date IS NULL AND
                  CUP = NEW.CUP AND lab_code = NEW.lab_code
        )
    THEN
        RAISE EXCEPTION 'lab % non partecipa al progetto %', NEW.lab_code, NEW.CUP;
    END IF;

    RETURN NEW;
END $$;

CREATE OR REPLACE TRIGGER validate_equipment_request
BEFORE INSERT ON equipment_request
FOR EACH ROW
EXECUTE FUNCTION validate_equipment_request();
```

works_at_no_duplicates

```
CREATE OR REPLACE FUNCTION works_at_no_duplicates()
RETURNS trigger

language 'plpgsql'

AS $$

DECLARE
    emps_number INTEGER := 0;

BEGIN
    -- se end_date non è null, significa che non lavoro più in quel laboratorio
    -- quind posso inserire nella tabella
    IF NEW.end_date IS NOT NULL THEN
        RETURN NEW;
    END IF;

    -- conto quante volte l'impiegato che si vuole inserire
    -- lavora nello stesso laboratorio (end_date is null)
    SELECT COUNT(*) INTO emps_number
    FROM works_at
    WHERE NEW.cf_base_emp = cf_base_emp AND
          NEW.lab_code = lab_code AND
          end_date IS NULL;

    -- se l'impiegato lavora gia in quel laboratorio, impedisco l'inserimento
    IF emps_number > 0 THEN
        RAISE EXCEPTION 'L''impiegato % lavora già al laboratorio %', NEW.cf_base_emp,
             NEW.lab_code;
    END IF;
    RETURN NEW;
END $$;

CREATE TRIGGER works_at_no_duplicates
BEFORE INSERT ON works_at
FOR EACH ROW
EXECUTE FUNCTION works_at_no_duplicates();
```

end_project

```
CREATE OR REPLACE FUNCTION end_project()
RETURNS TRIGGER
language 'plpgsql'
AS $$

BEGIN
    UPDATE take_part
    SET end_date = NEW.end_date
    WHERE end_date IS NULL AND CUP = NEW.CUP;

    RETURN NEW;
END $$;

CREATE OR REPLACE TRIGGER end_project
AFTER UPDATE OF end_date ON project
FOR EACH ROW
EXECUTE FUNCTION end_project();
```

4.8 Procedure e funzioni

base_emp_login

Effettua il login di un *base_emp* con *email* e *password*.

Restituisce il tipo (junior, middle, senior, manager) se il login è avvenuto con successo, *NULL* altrimenti.

```
CREATE OR REPLACE FUNCTION base_emp_login(in_email IN VARCHAR(100), in_passw IN
    password_type)
RETURNS VARCHAR
language 'plpgsql'
AS $$

DECLARE
    emp_t VARCHAR;

BEGIN
    SELECT type INTO emp_t
    FROM base_emp
    WHERE email = in_email AND passw = crypt(in_passw, passw);

    RETURN emp_t;
END; $$;
```

project_salaried_login

Effettua il login di un *project_salaried* con *email* e *password*.

Restituisce vero o falso a seconda della riuscita del login.

```
CREATE OR REPLACE FUNCTION project_salaried_login(in_email IN VARCHAR(100), in_passw
    IN password_type)
RETURNS boolean
language 'plpgsql'

AS $$
BEGIN
RETURN EXISTS(
    SELECT *
    FROM project_salaried
    WHERE email = in_email AND passw = crypt(in_passw, passw)
);
END; $$;
```

buy_equipment

Prende in ingresso un codice di una richiesta attrezzatura e il prezzo per singolo prodotto.
Soddisfa la richiesta effettuando gli acquisti richiesti.

```
CREATE OR REPLACE PROCEDURE buy_equipment(
    request_code    IN equipment_request.code%TYPE,
    price           IN purchase.price%TYPE -- prezzo per la singola unità di prodotto
)
language 'plpgsql'
AS $$
DECLARE
    -- codice dell'attrezzatura
    equipment_code equipment.code%TYPE;

    request equipment_request%ROWTYPE;

    -- variabile contatore che scorre la quantità da acquistare
    quantity equipment_request.quantity%TYPE;

BEGIN
    -- prendo la richiesta
    SELECT * INTO request
    FROM equipment_request
    WHERE code = request_code;

    -- se non esiste nessuna richiesta con quel codice, blocca l'acquisto
    IF NOT FOUND THEN
        RAISE EXCEPTION 'request % not found', request_code;
    END IF;

    quantity := request.quantity;

    WHILE quantity > 0 LOOP
        quantity := quantity - 1;

        equipment_code := uuid_generate_v4();

        INSERT INTO equipment (code, name, type, tech_specs, lab_code) VALUES
        (equipment_code, request.name, request.type, request.specs, request.lab_code);

        INSERT INTO purchase (price, purchase_date, CUP, equipment_code) VALUES
        (price, CURRENT_DATE, request.CUP, equipment_code);
    END LOOP;

    DELETE FROM equipment_request
    WHERE code = request_code;
END; $$;
```

change_lab_afference

Prende in ingresso il *cf* di un impiegato e il codice di un laboratorio nel quale spostare l'impiegato.

Se l'impiegato sta lavorando in un laboratorio diverso rispetto al quale lo si vuole spostare, lo rimuove prima da quello vecchio (aggiornando la data di fine *end_date* a *CURRENT_DATE*).

Successivamente si inserisce in *works_at* l'impiegato per quel laboratorio.

```
CREATE OR REPLACE PROCEDURE change_lab_afference(
    cf base_emp.cf%TYPE,
    v_lab_code laboratory.lab_code%TYPE
)
LANGUAGE 'plpgsql'
AS $$

BEGIN
    -- verificare che l'impiegato stia lavorando in un laboratorio che non sia quello
        nuovo
    IF EXISTS (
        SELECT *
        FROM works_at
        WHERE lab_code <> v_lab_code AND cf_base_emp = cf AND end_date IS NULL
    ) THEN
        -- "sganciare" l'impiegato dal laboratorio attuale
        UPDATE works_at
        SET end_date = current_date
        WHERE cf_base_emp = cf AND end_date IS NULL;
    END IF;

    -- assegnare l'impiegato al nuovo laboratorio
    INSERT INTO works_at VALUES (current_date, null, cf, v_lab_code);
    RAISE NOTICE '% is now working into lab: %', cf, v_lab_code;
END; $$;
```

check_seniority

Controlla il grado di anzianità degli impiegati e li aggiorna nel caso. Si suppone che il controllo dell'avanzamento di grado verrà fatto una volta al mese dopo il giorno di paga.

```
CREATE OR REPLACE PROCEDURE check_seniority()
language 'plpgsql'
AS $$
DECLARE
difference_years INTEGER;
emp_current_role career_log.new_role%TYPE := '';

-- prendo l'ultimo log dove new_role = junior
-- escludendo quelli che attualmente sono manager o licenzianti
log_cursor CURSOR FOR
    SELECT *
    FROM career_log AS C

    -- escludo quelli attualmente manager o licenziati
    WHERE C.cf IN(

        -- prendo l'ULTIMO LOG di ogni impiegato
        -- escludendo i log dove new_role è manager oppure '' (licenziato)
        SELECT C1.cf
        FROM last_log AS C1
        WHERE
            C1.new_role <> 'manager' AND    -- escludo i manager
                C1.new_role <> ''   -- escludo i licenziati
    ) AND

        -- seleziono l'ultimo log dov'è stato assunto
    C.new_role_date = (
        SELECT MAX(C1.new_role_date)
        FROM career_log AS C1
        WHERE C.cf = C1.cf AND (C1.ex_role,C1.new_role) = ('','junior')
    );

BEGIN
    FOR item IN log_cursor LOOP
        difference_years := EXTRACT( YEAR FROM AGE(CURRENT_TIMESTAMP, item.
            new_role_date));

        -- prendo il ruolo attuale dell'impiegato
        SELECT new_role INTO emp_current_role
        FROM career_log
        WHERE cf = item.cf AND
                new_role_date = (
                        SELECT MAX(C2.new_role_date)
                        FROM career_log AS C2
                        WHERE C2.cf = item.cf
                );

        -- lavora da più di 7 anni e NON è già senior
        -- avanza a senior
        IF difference_years >= 7 AND emp_current_role <> 'senior' THEN
            INSERT INTO career_log (ex_role, new_role, new_role_date, CF)
            VALUES ('middle', 'senior', CURRENT_TIMESTAMP, item.cf);

        -- lavora da 3 anni (o più) ma meno di 7 e NON è già middle
        -- avanza a middle
        ELSIF difference_years >= 3 AND difference_years < 7 AND emp_current_role <> '
            middle' THEN
            INSERT INTO career_log (ex_role, new_role, new_role_date, CF)
            VALUES ('junior', 'middle', CURRENT_TIMESTAMP, item.cf);
        END IF;
    END LOOP;
END;$$;
```

fire

Licenzia l'impiegato passato come parametro, ossia inserisce una tupla in *career_log* con *new_role* = ' '

```
CREATE OR REPLACE PROCEDURE fire(fired_cf IN career_log.cf%TYPE)
language 'plpgsql'
AS $$

DECLARE
    new_role career_log.ex_role%TYPE := '';

BEGIN
    SELECT LL.new_role INTO new_role
    FROM last_log AS LL
    WHERE LL.cf = fired_cf;

    IF new_role = '' THEN
        RAISE EXCEPTION '% already fired', fired_cf;
    END IF;

    INSERT INTO career_log (ex_role, new_role, new_role_date, CF)
    VALUES (new_role, '', CURRENT_TIMESTAMP, fired_cf);

    EXCEPTION
        WHEN SQLSTATE '23514' THEN
            RAISE EXCEPTION '% non trovato', fired_cf;
END; $$;
```

hire_project_salaried

Assume un impiegato a progetto. Se questo è già stato inserito nella tabella *project_salaried*, verrà inserita solo la tupla in *workson*.

```
CREATE OR REPLACE PROCEDURE hire_project_salaried(
    emp_cf      IN project_salaried.CF%TYPE,
    first_name  IN project_salaried.first_name%TYPE,
    last_name   IN project_salaried.last_name%TYPE,
    email       IN project_salaried.email%TYPE,
    passw       IN project_salaried.passw%TYPE,
    role        IN project_salaried.role%TYPE,
    birth_date  IN project_salaried.birth_date%TYPE,
    pay         IN works_on.pay%TYPE,
    hire_date   IN works_on.hire_date%TYPE,
    expiration  IN works_on.expiration%TYPE,
    CUP         IN works_on.CUP%TYPE
)
language 'plpgsql'
AS $$
BEGIN
    IF
        NOT EXISTS(
            SELECT *
            FROM project_salaried
            WHERE cf = emp_cf
        )
    THEN
        INSERT INTO project_salaried (CF, first_name, last_name, email, passw, role,
            birth_date) VALUES
        (emp_cf, first_name, last_name, email, passw, role, birth_date);
    END IF;

    INSERT INTO works_on (pay, hire_date, expiration, CF, CUP) VALUES
    (pay, hire_date, expiration, emp_cf, CUP);
END; $$;
```

promotion_to_manager

Promuove l'impiegato preso in input a *manager*.

Se è gia *manager* oppure è *referente scientifico* o *manager scientifico* non può essere promosso. Altrimenti inserisce in *career_log* con *new_role* = 'manager'

```
CREATE OR REPLACE PROCEDURE promotion_to_manager( v_cf_manager IN base_emp.cf%TYPE )
language 'plpgsql'
AS $$

DECLARE
    emp_log last_log%ROWTYPE;

BEGIN
    -- recuperare l'utlimo log dell'impiegato in ingresso e verificare che non sia già
         manager
    SELECT * INTO emp_log
    FROM last_log
    WHERE cf = v_cf_manager AND new_role <> 'manager';

    IF NOT FOUND THEN
        RAISE EXCEPTION '% is already manager', v_cf_manager;

    -- se è senior ed è un referente scientifico (progetto) oppure un manager
        scientifico (laboratorio)
    ELSIF
        emp_log.new_role = 'senior' AND
        (
            EXISTS(
                SELECT *
                FROM project
                WHERE cf_scientific_referent = v_cf_manager AND end_date IS NULL
            )
        OR
            EXISTS(
                SELECT *
                FROM laboratory
                WHERE cf_scientific_manager = v_cf_manager
            )
        )
    THEN
        RAISE EXCEPTION '% cannot be promoted, is scientific manager and/or scientific
             referent', v_cf_manager;
    END IF;


    -- inserimento nuova tupla in carrer log
    INSERT INTO career_log(ex_role, new_role, new_role_date, CF)
    VALUES (emp_log.new_role, 'manager', CURRENT_TIMESTAMP, v_cf_manager);

    RAISE NOTICE '% promoted to manager', v_cf_manager;
END; $$;
```

revoke_manager

Declassa l'impiegato preso in input da *manager* al ruolo che aveva precedentemente.

Se nel mentre che era *manager* è avanzato di grado per anzianità, non dovrà essere declassato al ruolo precedente ma al nuovo grado (in base all'anzianità).

```
CREATE OR REPLACE PROCEDURE revoke_manager( v_cf_manager IN base_emp.cf%TYPE )
language 'plpgsql'
AS $$

DECLARE
    emp_log last_log%ROWTYPE;
    v_ex_role career_log.ex_role%TYPE := '';
    hire_date career_log.new_role_date%TYPE;
    difference_years INTEGER;

BEGIN
    IF
        EXISTS(
            SELECT *
            FROM project
            WHERE cf_manager = v_cf_manager AND end_date IS NULL
        )
    THEN
        RAISE EXCEPTION '% cannot be declassed, is a project manager', v_cf_manager;
    END IF;

    -- recuperare l'utlimo log dell'impiegato in ingresso e verificare che sia manager
    SELECT * INTO emp_log
    FROM last_log
    WHERE cf = v_cf_manager AND new_role = 'manager';

    IF NOT FOUND THEN
        RAISE EXCEPTION '% is not manager', v_cf_manager;
    ELSE
        -- l'impiegato è un manager e deve essere declassato al ruolo precedente
        -- selezionare la data di assunzione
        SELECT new_role_date INTO hire_date
        FROM career_log
        WHERE (ex_role, new_role) = ('', 'junior') AND cf = v_cf_manager;

        -- calcolare gli anni di lavoro
        difference_years := EXTRACT( YEAR FROM AGE(CURRENT_TIMESTAMP, hire_date));


        -- assegnare ex_role in base agli anni di lavoro
        IF difference_years >= 7 THEN
            v_ex_role := 'senior';
        ELSIF difference_years >= 3 AND difference_years < 7 THEN
            v_ex_role := 'middle';
        ELSE
            v_ex_role := 'junior';
        END IF;

        -- inserimento nuova tupla in carrer log
        INSERT INTO career_log(ex_role, new_role, new_role_date, CF)
        VALUES ('manager', v_ex_role, CURRENT_TIMESTAMP, v_cf_manager);

        RAISE NOTICE '% declassed to %', v_cf_manager, v_ex_role;

    END IF;
END;$$;
```

4.9 Dizionario dei vincoli

4.9.1 Vincoli intra-relazionali

base_emp	
Vincolo	**Descrizione**
emp_email_unique	L'email è univoca
base_emp_pk	Vincolo di chiave primaria

laboratory	
Vincolo	**Descrizione**
lab_pk	Vincolo di chiave primaria
site_number_fk	Vincolo di chiave esterna
cf_scientific_manager_fk	Vincolo di chiave esterna

career_log	
Vincolo	**Descrizione**
emp_career_fk	Vincolo di chiave esterna. Se viene eliminato l'impiegato, verrà eliminata anche la tupla in *career_log*
check_new_grade	Controlla che gli scatti di carriera siano coerenti. Ad esempio non è possibile passare da *junior* a *senior* senza essere diventati prima *middle*. La stringa vuota come primo parametro significa che l'impiegato è stato assunto; come secondo invece, significa che è stato licenziato. **Casi possibili:** • (' ', 'junior') • ('junior', 'middle') • ('middle', 'senior') • ('junior', 'manager') • ('middle', 'manager') • ('senior', 'manager') • ('manager', 'junior') • ('manager', 'middle') • ('manager', 'senior') • ('junior', ' ') • ('middle', ' ') • ('senior', ' ') • ('manager', ' ')

equipment_request	
Vincolo	**Descrizione**
equipment_request_pk	Vincolo di chiave primaria
CUP_fk	Vincolo di chiave esterna. Se viene eliminato il progetto (o modificato il cup), verrà eliminata (o aggiornata) anche la tupla in *equipment_request*
lab_code_fk	Vincolo di chiave esterna. Se viene eliminato il laboratorio (o modificato il cup), verrà eliminata (o aggiornata) anche la tupla in *equipment_request*
check_quantity	Controlla che la quantità sia maggiore di 0

equipment	
Vincolo	**Descrizione**
equipment_pk	Vincolo di chiave primaria
lab_code_fk	Vincolo di chiave esterna. Se viene eliminato il laboratorio, la chiave esterna verrà posta a NULL. Se viene aggiornata la chiave di *laboratory*, verrà aggiornata anche la chiave esterna.

project_salaried	
Vincolo	**Descrizione**
project_salaried_pk	Vincolo di chiave primaria
emp_email_unique	L'email è univoca

project	
Vincolo	**Descrizione**
project_pk	Vincolo di chiave primaria
check_deadline	Verifica che la deadline sia dopo la data di inizio (nel caso in cui ci fosse).
check_end_date	Verifica che la data di fine progetto sia dopo la data di inizio.
fk_cf_manager	Vincolo di chiave esterna
fk_cf_scientific_referent	Vincolo di chiave esterna

purchase	
Vincolo	**Descrizione**
cup_type_fk	Vincolo di chiave esterna. Se viene eliminato il progetto, la chiave esterna viene posta a NULL, questo perché si vuole tenere traccia della data d'acquisto e del prezzo dell'attrezzatura.
equipment_code_fk	Vincolo di chiave esterna. Se viene eliminato l'attrezzatura, verrà eliminata anche la tupla in purchase, dato che diventa inutile tenere traccia dell'acquisto di un prodotto che non si conosce.
check_price	Controlla che il prezzo sia maggiore di 0

site	
Vincolo	**Descrizione**
site_pk	Vincolo di chiave primaria

take_part	
Vincolo	**Descrizione**
date_integrity	Verifica che la data in cui il laboratorio termina di lavorare al progetto, sia dopo la data di inizio.
CUP_fk	Vincolo di chiave esterna. Se viene eliminato il progetto (o modificato il cup), verrà eliminata (o aggiornata) anche la tupla in *take_part*
lab_code_fk	Vincolo di chiave esterna. Se viene eliminato il laboratorio (o modificato il lab_code), verrà eliminata (o aggiornata) anche la tupla in *take_part*

works_at	
Vincolo	**Descrizione**
date_integrity	Verifica che la data in cui l'impiegato termina di lavorare al laboratorio, sia dopo la data di inizio.
cf_base_emp_fk	Vincolo di chiave esterna. Se viene eliminato l'impiegato (o modificato il cf), verrà eliminata (o aggiornata) anche la tupla in *works_at*
lab_code_fk	Vincolo di chiave esterna. Se viene eliminato il laboratorio (o modificato il lab_code), verrà eliminata (o aggiornata) anche la tupla in *works_at*

works_on	
Vincolo	**Descrizione**
check_expiration_date	Verifica che la data di fine contratto sia dopo la data di assunzione.
fk_cf	Vincolo di chiave esterna. Se viene eliminato l'impiegato, verrà posta la chiave esterna a NULL. Questo perché si vuole salvare i contratti anche se un impiegato a progetto viene eliminato.
fk_cup	Vincolo di chiave esterna.

Domini	
Vincolo	**Descrizione**
emp_type_check	Il valore inserito dev'essere uno fra questi (' ', 'junior', 'middle', 'senior', 'manager') e non deve essere NULL
check_cup_type	Il valore inserito dev'essere lungo 15 caratteri
cf_type_length	Il valore inserito dev'essere lungo 16 caratteri

4.9.2 Vincoli inter-relazionali

L'implementazione SQL dei seguenti trigger è riportata nella sezione dei trigger e trigger functions

- **check_equipment_funds**
  Prima di acquistare un'attrezzatura, controlla che il progetto abbia i fondi.
- **check_funds_to_hire**
  Prima di assumere un impiegato a progetto, controlla che il progetto abbia i fondi.
- **check_laboratory_assignment**
  Prima di inserire un laboratorio o prima di aggiornare il manager scientifico, controlla che questo sia *senior*
- **check_project_assignment**
  Prima di inserire un progetto, o prima di aggiornare il referente scientifico o il supervisore, controlla che questi siano rispettivamente *senior* e *manager*
- **check_works_on_date**
  Controlla che lo stesso impiegato non lavori già allo stesso progetto. Può lavorare allo stesso progetto solo se la data di assunzione del nuovo contratto è superiore alla data di scadenza del suo ultimo contratto per quel progetto.
- **crypt_password**
  Prima di inserire un *impiegato/impiegato a progetto*, cripta la password.
- **employee_hiring**
  Quando viene assunto un impiegato (ovvero quando viene inserito nella tabella *base_emp*), viene inserito un log in *career_log*.
  (' ', 'junior')
- **fire_emp**
  Prima di licenziare un impiegato, ossia prima di inserire una tupla in *career_log* dove il campo *new_role* = ' ' , controlla che l'impiegato non sia referente scientifico o manager di progetto, oppure che non sia manager scientifico di un laboratorio.

  In quel caso non è possibile licenziarlo, bisognerà prima rimuoverlo da questi ruoli.
- **max_lab_in_project**
  Verifica che ad un progetto prendano parte al massimo 3 laboratori
- **take_part_no_duplicates**
  Verifica che un laboratorio non prenda parte più volte allo stesso progetto nello stesso momento.
- **update_emp_type**
  Quando viene inserita una tupla in *career_log*, aggiorna il campo *type* di *base_emp* in base al *new_role* appena inserito.
- **validate_equipment_request**
  Prima di inserire una richiesta di attrezzatura, controlla che il laboratorio richiedente partecipi al progetto.
- **works_at_no_duplicates**
  Prima di inserire un impiegato in *works_at*, ossia prima di farlo lavorare in un laboratorio, controlla che lo stesso impiegato non lavori già allo stesso laboratorio.
- **emp_works_at_one_lab**
  Controlla che un impiegato lavori ad un unico laboratorio
- **end_project**
  Quando un progetto termina, aggiorna l'*end_date* anche in *take_part*