Matricola	Nome	Cognome	Indirizzo e-mail istituzionale
N86003587	Agostino	Cesarano	ag.cesarano@studenti.unina.it
N86003884	Alessandro	Fortino	ale.fortino@studenti.unina.it

Sommario

Progettazione

Dopo un'accurata analisi della traccia, si è proseguito analizzando il problema posto ricercando l'implementazione **Basi di Dati Relazionale** che sia più adatta.

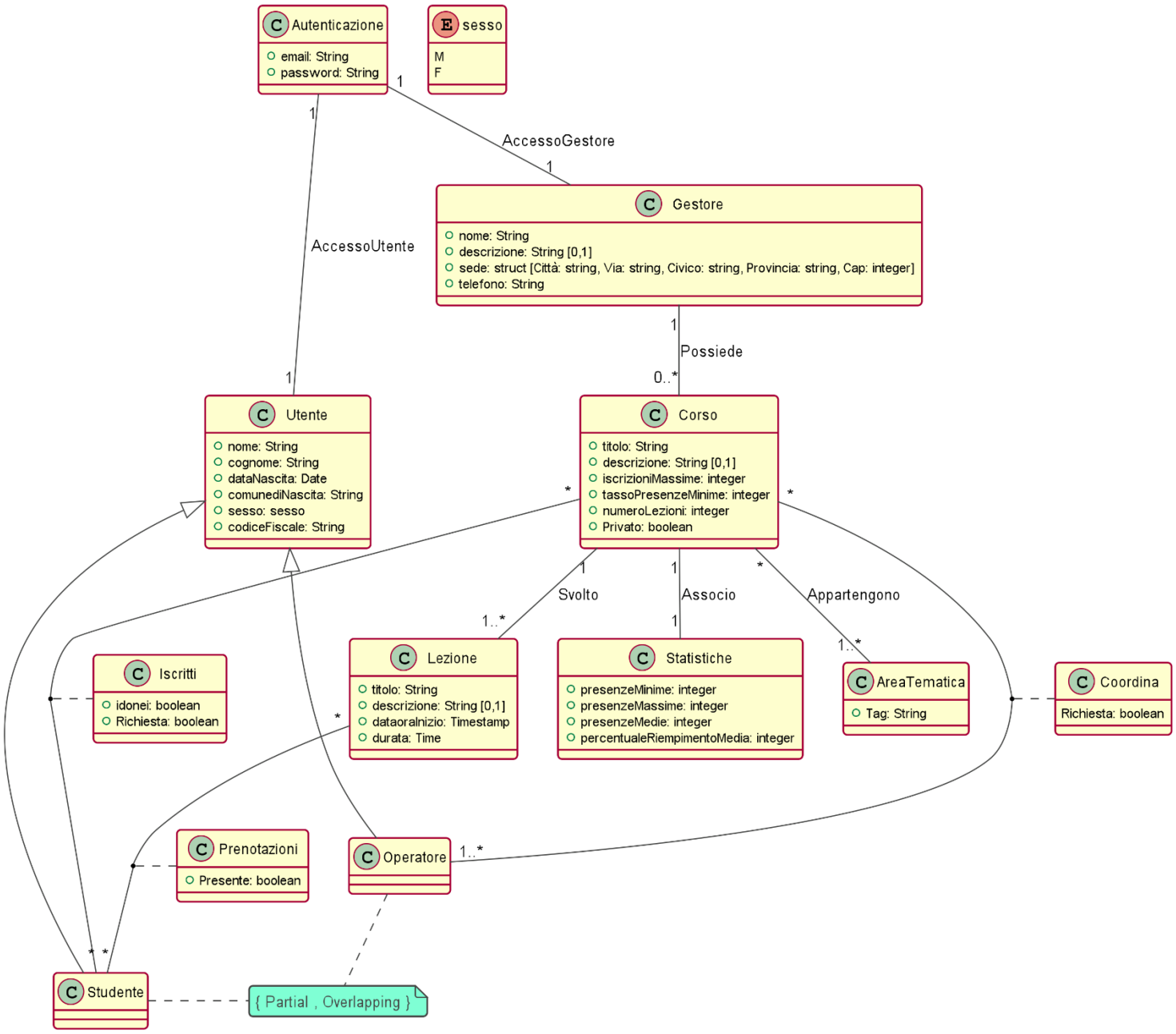
L'idea di base è di creare uno Schema quanto più semplice e intuitivo ma che sia a sua volta funzionale ed efficiente.

Progettazione Concettuale

Il primo passo della progettazione è la creazione di uno **schema concettuale** che definisca ciò che sono i principali aspetti delle basi di dati, **descrizione astratta** della Basi di Dati.

1. Definizione delle classi-entità.

2. Definizione delle associazioni.



Cardinalità
Tutti gli attributi, ad eccezioni di quelli in cui specificata, hanno cardinalità pari ad [1].

Enumerazioni
È presente un'enumerazione che definisce i valori di **sesso**, attributo presente nell'entità **Utente**.

Attributi Strutturati
È presente un attributo strutturato che è relativo alla **sede** dell'entità **Gestore**.

Specializzazioni
Sono presenti due Specializzazioni relative all'entità Utente, di tipo **parziale** e **overlapping**, infatti è stata decisione di progettazione rendere un Utente sia Operatore che Studente.

Prendendo ispirazione dalle classiche piattaforme di gestione corsi, un Utente può amministrare Corsi, ma allo stesso tempo prendere parte ad un Corso come Studente; **Overlapping**.

Allo stesso tempo un Utente può non essere iscritto a nessun Corso, come studente o come operatore, quindi essere un utente classico; **Partial**.

Classi di associazione
Utile è stato l'utilizzo di classi di associazione.

L'associazione Iscritti contiene gli attributi Idonei e Richiesta:

Idonei è un tipo semplice booleano che rappresenta **l'idoneità** dello studente, che come descritta dalla traccia avviene quando le **presenze a lezione** raggiungono il **tasso minimo**.

Richiesta è un tipo semplice booleano che rappresenta, nel caso in cui il Corso è di tipo Privato, l'accettazione o meno della richiesta di iscrizione al Corso.

L'associazione Prenotazione contiene gli attributi Presente:

Presente è un tipo semplice che rappresenta la presenza o meno dello Studente a Lezione.

L'associazione Coordina contiene gli attributi Richiesta:

Richiesta è un tipo semplice che rappresenta l'accettazione da parte dell'Utente della richiesta di coordinazione di un Corso, quindi di essere operatore del Corso.

Ristrutturazione

La **ristrutturazione** dello schema concettuale si divide in tre parti:

1. Eliminazione delle generalizzazioni/specializzazioni
2. Eliminazione degli attributi multipli e degli attributi strutturati
3. Scelta degli identificatori primari

Eliminazione delle specializzazioni

Il primo punto da affrontare è l'eliminazione delle specializzazioni, ci sono diversi modi per fronteggiare il problema. La scelta è ricaduta sul metodo **meno distruttivo**, che consiste nel **rimpiazzare le specializzazioni** con le **associazioni**.

Le specializzazioni verranno rimpiazzate con delle associazioni con cardinalità **[1] a [0..1]**, un **Utente** *può essere* **associato** o meno ad uno **Studente**, un **Utente** *può essere* **associato** o meno ad un **Operatore**.

Inoltre con questa ristrutturazione, le specializzazioni (**partial, overlapping**) non necessitano di vincoli aggiuntivi.

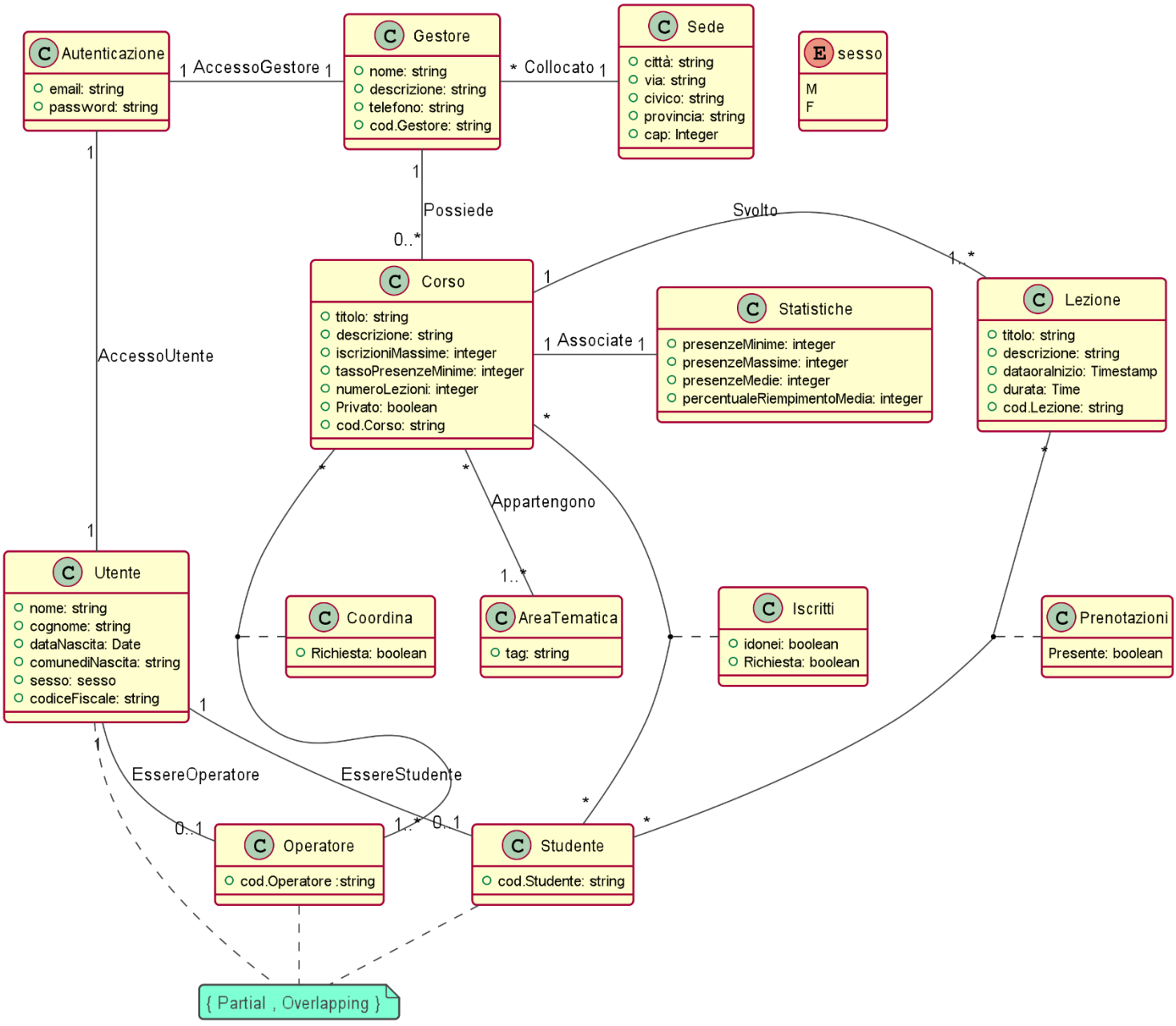
Eliminazione degli attributi strutturati

Il secondo punto consiste nell'eliminazione degli attributi strutturati, tra i due modi possibili è stato scelto di eliminare l'attributo strutturato mediante l'utilizzo di una **classe specifica**.

Nel nostro caso è stata creata una nuova classe "**Sede**" che contiene tutti gli attributi dell'attributo strutturato e viene collegato alla classe **Gestore** con un'associazione con cardinalità **[1] a [*]**, molti **Gestori** hanno **associati** *una* **Sede**.

Individuazione delle chiavi primarie

Alcune Classi devono essere individuate **univocamente** con uno o più attributi, gli attributi in questione possono essere già presenti nello schema concettuale oppure possono essere introdotte (**in generale codici**) ,la chiave primaria è necessaria per tradurre le associazioni nello **schema logico**.


Autenticazione
email: string
password: string
AccessoGestore
Gestore
nome: string
descrizione: string
telefono: string
cod.Gestore: string
Collocato
Sede
città: string
via: string
civico: string
provincia: string
cap: Integer
sesso
M
F
Possiede
Svolto
Corso
titolo: string
descrizione: string
iscrizioniMassime: integer
tassoPresenzeMinime: integer
numeroLezioni: integer
Privato: boolean
cod.Corso: string
Associate
Statistiche
presenzeMinime: integer
presenzeMassime: integer
presenzeMedie: integer
percentualeRiempimentoMedia: integer
Lezione
titolo: string
descrizione: string
dataoralnizio: Timestamp
durata: Time
cod.Lezione: string
AccessoUtente
Appartengono
Utente
nome: string
cognome: string
dataNascita: Date
comunediNascita: string
sesso: sesso
codiceFiscale: string
Coordina
Richiesta: boolean
AreaTematica
tag: string
Iscritti
idonei: boolean
Richiesta: boolean
Prenotazioni
Presente: boolean
EssereOperatore
EssereStudente
Operatore
cod.Operatore :string
Studente
cod.Studente: string
{ Partial , Overlapping }

Dizionario delle classi

Classe	*Descrizione*	*Attributi*
Autenticazione	*Descrittore dei dati di Autenticazione*	**email** (*string*): Email di Autenticazione **password** (*string*): Password di Autenticazione
Gestore	*Descrittore dei gestori dei corsi di formazione*	**nome** (*string*): Nome del gestore **descrizione** (*string, opzionale*): Breve descrizione del gestore **sede** *(struct [Città: string, Via: string, Civico: string, Provincia: string, Cap: integer])* **telefono** (*string*): Numero di telefono del gestore **cod.Gestore** (*string*): Codice identificativo del Gestore
Corso	*Descrittore dei corsi di formazione*	**titolo** (*string*): Titolo del corso **descrizione** (*string, opzionale*): Breve descrizione del corso **iscrizioniMassime** (*integer*): Numero massimo di iscrizioni al corso **tassoPresenzeMinime** (integer): Tasso in percentuale di presenza minima richiesta dal corso **numeroLezioni** (integer): Numero di lezioni pianificate per corso **cod.Corso** (*string*): Codice identificativo del Corso
Statistiche	*Descrittore delle statistiche dei corsi*	**presenzeMinime** (*integer*): Valore minimo di presenze registrato ad una lezione del corso **presenzeMassime** (*integer*): Valore massimo di presenze registrate ad una lezione del corso **presenzeMedie** (*integer*): Valore medio di presenze a lezione del corso

		percentualeRiempimento (*integer*): Valore in percentuale del riempimento di un corso, iscritti su iscrizioni massime
Lezione	Descrittore delle lezioni dei corsi	**titolo** (*string*): Titolo della lezione **descrizione** (*string, opzionale*): Breve descrizione della lezione **dataorainizio** (*timestamp*): Data e ora d'inizio della lezione **durata** (*timestamp*): Durata di una lezione **cod.Lezione** (*string*): Codice identificativo della Lezione
Area tematica	*Descrittore delle aree tematiche di un corso*	**tag** (string): Tag che descrive le aree tematiche-parole chiavi.
Studente	*Descrittore degli utenti che sono studenti*	**cod.Studente** (*string*): Codice identificativo dello Studente
Operatore	*Descrittore degli utenti che sono operatori*	**cod.Operatore** (*string*): Codice identificativo del Operatore
Utente	*Descrittore degli utenti che sono operatori*	**nome** (*string*): Nome dell'utente **cognome** (*string*): Cognome dell'utente **dataNascita** (*date*): Data di nascita dell'utente **comunediNascita** (*string*): Comune in cui è nato l'utente **sesso** (sesso): Sesso dell'utente **codiceFiscale** (string): Identifica univocamente ogni Utente

Dizionario delle associazioni

Nome	***Descrizione***	***Classi coinvolte***
AccessoGestore	*Esprime l'appartenenza dei dati di accesso di uno specifico gestore*	**Gestore [1]** ruolo (**accede**): indica il gestore a cui sono associati i dati di autenticazione. **Autenticazione [1]** ruolo (**fa accedere**): indica i dati utili a un gestore per accedere.
AccessoUtente	*Esprime l'appartenenza dei dati di accesso di uno specifico utente*	**Utente [1]** ruolo(**accede**): indica l'utente a cui sono associati i dati di autenticazione. **Autenticazione [1]** ruolo (**fa accedere)**: Indica i dati utili a un utente per accedere.
Collocato	*Esprime l'appartenenza di più sedi a un gestore*	**Gestore [*]** ruolo (**possiede**): indica il gestore che può collocarsi in più sedi. **Sede [1]** ruolo (**possedut**a): Indica le varie sedi in cui un gestore può collocarsi.
Associo	*Esprime l'appartenenza di specifici dati statistici a un determinato corso*	**Corso [1]** ruolo (**caratterizzato**): indica il corso a cui sono associati dati statistici. **Statistiche [1]** ruolo (**caratterizzano**): indica i dati statistici di un determinato corso.
Possiede	*Esprime l'appartenenza dei corsi a un gestore*	**Gestore [1]** ruolo (**possiede):** Indica il gestore a cui sono associati i corsi. **Corso [0…*]** ruolo (**posseduti**): indica i corsi posseduti da un gestore.
Appartiene	*Esprime l'appartenenza di un corso a delle aree tematiche*	**Corso [*]** ruolo (**descritti):** indica i corsi ai quali sono associati aree tematiche. **AreaTematica [1..*]** ruolo (**descrivono**): indica le aree tematiche alle quali appartengono i corsi.
Iscritti	*Indica l'appartenenza degli studenti ai corsi*	**Studente [*]** ruolo (**frequenta**)**:** indica gli studenti iscritti ai corsi. **Corso [*]** ruolo (**frequentato**): indica i corsi alla quale si iscrivono gli studenti.
Prenotazioni	*Indica l'appartenenza degli studenti alle lezioni*	**Studente [*]** ruolo (**prenotano**)**:** indica gli studenti iscritti alle lezioni **Lezione [*]** ruolo (**prenotate**): indica le lezioni alla quale si iscrivono gli studenti
Svolto	*Esprime l'appartenenza delle lezioni a un corso*	**Corso [1]** ruolo (**costituiti**): indica il corso alla quale appartengono le lezioni. **Lezione [1..*]** ruolo (**costituiscono**): indica le lezioni delle quali è costituito un corso.
Coordina	*Esprime l'appartenenza di un operatore ai corsi*	**Operatore [1..*]** ruolo **(gestiscono):** indica gli operatori che coordinano i corsi **Corso [*]** ruolo (**gestiti**): indica i corsi coordinati dagli operatori

Progettazione Logica

Avendo uno schema iniziale su entità e associazioni, si può proseguire con la seconda parte della progettazione, che è specifica rispetta a quella che sarà il **tipo di base di dati** da utilizzare, cioè il **modello dei dati** che si vuole adottare.

Nella progettazione logica i punti da affrontare sono:

1. Scelta del modello di dati da adottare
2. Traduzione dello **schema concettuale** dei dati in uno **schema logico** che rispecchia il modello dei dati scelto.

In questo progetto il modello utilizzato è quello **relazionale** dei dati su cui si fondano le basi di dati relazionali.

Schema Logico

Autenticazione (email, password)

Gestore (nome, descrizione, telefono, cod.Gestore, email)

email → Autentocazione.email

Sede (città, via, civico, provincia, Cap, cod.Gestore)

cod.Gestore → Gestore.cod.Gestore

Corso (titolo, descrizione, iscrizioneMassime, tassoPresenzeMinime, cod.Corso, cod.Gestore)

cod.Gestore → Gestore.cod.Gestore

Utente (nome, cognome, dataNascita, sesso, codiceFiscale, email)

email → Autentocazione.email

Statistiche(presenzeMinime, presenzeMassime, presenzeMedie, percentualeRiempimentoMedia, cod.Corso)

cod.Corso → Corso.cod.Corso

Lezione(titolo, descrizione, dataoraInizio, cod.Lezione, cod.Corso)

cod.Corso → Corso.cod.Corso

Studente(cod.Studente,codiceFiscale)

codiceFiscale → Utente.codiceFiscale

Prenotazioni(cod.Lezione, cod.Studente, Presente)

cod.Lezione → Lezione.cod.Lezione

cod.Studente → Studente.cod.Studente

Iscritti(cod.Corso, cod.Studenti, Idonei, Richiesta)

cod.Corso → Corso.cod.Corso

Operatore(cod.Operatore, codiceFiscale)

codiceFiscale → Utente.codiceFiscale

Coordina(cod.Corso, cod.Operatore, Richiesta)

cod.Corso → Corso.cod.Corso

cod.Operatore → Operatore.cod.Operatore

AreaTematica(tag)

Appartiene(tag, cod.Corso)

cod.AreaTematica → AreaTematica.cod.AreaTematica

Progettazione Fisica

L'ultima fase della progettazione di una base di dati è la **progettazione fisica**. Prima di iniziare la progettazione fisica occorre **scegliere un DBMS** che implementi il modello dei dati dello schema logico.

Il DBMS da noi scelto è PostgreSQL che è un **Sistema di gestione di basi di dati** relazionali.

La progettazione fisica consiste nella traduzione dello schema logico dei dati in uno **schema fisico dei dati** contenente le definizioni delle tabelle, dei relativi vincoli e delle viste espresse in SQL.

Definizione delle tabelle

```
--Definizione tabella Appartiene.

CREATE TABLE public."Appartiene" (
    "codCorso" character(8) NOT NULL,
    tag character varying(30) NOT NULL
);
```

```
--Definizione tabella AreaTematica.

CREATE TABLE public."AreaTematica" (
    tag character varying(30) NOT NULL
);
```

```
--Definizione tabella Autenticazione.

CREATE TABLE public."Autenticazione" (
    email character varying(60) NOT NULL,
    pASsword character varying(30) NOT NULL
);
```

```
--Definizione tabella Coordina.

CREATE TABLE public."Coordina" (
    "codCorso" character(8) NOT NULL,
    "codOperatore" character(8) NOT NULL,
    "Richiesta" boolean DEFAULT false NOT NULL
);
```

```
--Definizione tabella Corso.

CREATE TABLE public."Corso" (
    titolo character varying(50) NOT NULL,
    descrizione character varying(200) DEFAULT 'Nessuna descrizione'::character varying,
    "iscrizioniMassime" integer NOT NULL,
    "tassoPresenzeMinime" integer NOT NULL,
    "codCorso" character(8) NOT NULL,
    "codGestore" character(8) NOT NULL,
    "numeroLezioni" integer NOT NULL,
    "Privato" boolean NOT NULL
);
```

```
--Definizione tabella Gestore.

CREATE TABLE public."Gestore" (
    nome character varying(30) NOT NULL,
    descrizione character varying(200) DEFAULT 'Nessuna descrizione'::character varying,
    telefono character varying(15) NOT NULL,
    "codGestore" character(8) NOT NULL,
    email character varying(60) NOT NULL
);
```

```
--Definizione tabella Iscritti

CREATE TABLE public."Iscritti" (
    "codCorso" character(8) NOT NULL,
    "codStudente" character(8) NOT NULL,
    "Idoneo" boolean DEFAULT false NOT NULL,
    "Richiesta" boolean NOT NULL
);
```

```
--Definizione tabella Lezione

CREATE TABLE public."Lezione" (
    titolo character varying(50) NOT NULL,
    descrizione character varying(200) DEFAULT 'Nessuna descrizione'::character varying,
    "dataoraInizio" timestamp(6) without time zone NOT NULL,
    durata time(6) without time zone NOT NULL,
    "codLezione" character(8) NOT NULL,
    "codCorso" character(8) NOT NULL
);
```

```
--Definizione tabella Operatore.

CREATE TABLE public."Operatore" (
    "codOperatore" character varying(16) NOT NULL,
    "codiceFiscale" character varying(16) NOT NULL
);
```

```
--Definizione tabella Prenotazioni.

CREATE TABLE public."Prenotazioni" (
    "codLezione" character(8) NOT NULL,
    "codStudente" character(8) NOT NULL,
    "Presente" boolean DEFAULT false NOT NULL
);
```

```
--Definizione tabella Sede.

CREATE TABLE public."Sede" (
    "città" character varying(35) NOT NULL,
    via character varying(30) NOT NULL,
    civico character varying(5) NOT NULL,
    provincia character varying(22) NOT NULL,
    "codGestore" character(8) NOT NULL
);
```

```
--Definizione tabella Statistiche

CREATE TABLE public."Statistiche" (
    "presenzeMinime" integer NOT NULL,
    "presenzeMassime" integer NOT NULL,
    "presenzeMedie" real NOT NULL,
    "percentualeRempimento" real NOT NULL,
    "codCorso" character(8) NOT NULL
);
```

```
--Definizione tabella Studente

CREATE TABLE public."Studente" (
    "codStudente" character varying(8) NOT NULL,
    "codiceFiscale" character varying(16) NOT NULL
);
```

```
--Definizione tabella Utente

CREATE TABLE public."Utente" (
    email character varying(60) NOT NULL,
    nome character varying(30) NOT NULL,
    cognome character varying(30) NOT NULL,
    "dataNascita" date NOT NULL,
    "comunediNascita" character varying(35) NOT NULL,
    sesso public.sesso NOT NULL,
    "codiceFiscale" character varying(16) NOT NULL
);
```

Definizione delle viste

```
--Vista utile all'login Gestore.

CREATE VIEW public.logingestore AS
 SELECT "Autenticazione".email,
    "Autenticazione".password,
    "Gestore".nome,
    "Gestore".descrizione,
    "Gestore".telefono,
    "Gestore"."codGestore"
   FROM (public."Autenticazione"
     JOIN public."Gestore" USING (email));
```

```
--Vista utile all'login Utente.

CREATE VIEW public.loginutente AS
 SELECT "Autenticazione".email,
    "Autenticazione".password,
    "Utente".nome,
    "Utente".cognome,
    "Utente"."dataNascita",
    "Utente"."comunediNascita",
    "Utente".sesso,
    "Utente"."codiceFiscale"
   FROM (public."Autenticazione"
     JOIN public."Utente" USING (email));
```

```
--Vista utile all'individuazione degli Operatori dei Corsi.

CREATE VIEW public.operatoricorsi AS
 SELECT "codOperatore",
    "Utente"."codiceFiscale",
    "Utente".email,
    "Utente".nome,
    "Utente".cognome,
    "Utente"."dataNascita",
    "Utente"."comunediNascita",
    "Utente".sesso,
    "Coordina"."codCorso",
    "Coordina"."Richiesta"
   FROM ((public."Utente"
     JOIN public."Operatore" USING ("codiceFiscale"))
     JOIN public."Coordina" USING ("codOperatore"));
```

```
--Vista utile alla ricerca di Corsi.

CREATE VIEW public.parametriricerca AS
 SELECT "Corso"."codGestore",
    "Corso".descrizione,
    "Corso".titolo,
    "Corso"."codCorso",
    "Corso"."Privato",
    "Corso"."numeroLezioni",
    "Corso"."tassoPresenzeMinime",
    "Corso"."iscrizioniMassime",
    "Gestore".nome,
    "Gestore".telefono,
    "Sede"."città",
    "Sede".provincia
   FROM ((public."Corso"
     JOIN public."Gestore" USING ("codGestore"))
     JOIN public."Sede" USING ("codGestore"));
```

```
--Vista che ci restituisce il numero di presenti a Lezione.

CREATE VIEW public.presenzelezioni AS
 SELECT res."codCorso",
    res."codLezione",
    COALESCE(res1.numero_presenti, (0)::bigint) AS numero_presenti
   FROM (( SELECT "Lezione"."codCorso",
            "Prenotazioni"."codLezione"
           FROM (public."Prenotazioni"
             JOIN public."Lezione" USING ("codLezione"))
          GROUP BY "Lezione"."codCorso", "Prenotazioni"."codLezione") res
     LEFT JOIN ( SELECT "Lezione"."codCorso",
            "Prenotazioni"."codLezione",
            count(*) AS numero_presenti
           FROM (public."Prenotazioni"
             JOIN public."Lezione" USING ("codLezione"))
          WHERE ("Prenotazioni"."Presente" = true)
          GROUP BY "Lezione"."codCorso", "Prenotazioni"."codLezione") res1 USING
("codLezione"));
```

```
--Vista utile all'individuazione degli Studenti iscritti ai Corsi.

CREATE VIEW public.studenticorsi AS
 SELECT "codStudente",
    "Studente"."codiceFiscale",
    "Utente".email,
    "Utente".nome,
    "Utente".cognome,
    "Utente"."dataNascita",
    "Utente"."comunediNascita",
    "Utente".sesso,
    "Iscritti"."codCorso",
    "Iscritti"."Idoneo",
    "Iscritti"."Richiesta"
   FROM ((public."Studente"
     JOIN public."Utente" USING ("codiceFiscale"))
     JOIN public."Iscritti" USING ("codStudente"));
```

Definizione dei vincoli

```
--Vincolo di Chiave Primaria sul codiceFiscale,esso è univoco.

Utente utente_pkey CONSTRAINT
        ALTER TABLE ONLY public."Utente"
        ADD CONSTRAINT utente_pkey PRIMARY KEY ("codiceFiscale");
```

```
--Vincolo di Chiave primaria sui tag.

AreaTematica AreaTematica_pkey CONSTRAINT
        ALTER TABLE ONLY public."AreaTematica"
        ADD CONSTRAINT "AreaTematica_pkey" PRIMARY KEY (tag);
```

```
--Vincolo di Chiave Primaria su email,un utente non può iscriversi con una mail già registrata,email è univoca.

Autenticazione autenticazione_pkey CONSTRAINT
        ALTER TABLE ONLY public."Autenticazione"
        ADD CONSTRAINT autenticazione_pkey PRIMARY KEY (email);
```

```
--Vincolo di Chiave Primaria su codCorso,esso è univoco.

Corso corso_pkey CONSTRAINT
        ALTER TABLE ONLY public."Corso"
        ADD CONSTRAINT corso_pkey PRIMARY KEY ("codCorso");
```

```
--Vincolo di Chiave Primaria sul codGestore,esso è univoco.

Gestore gestore_pkey CONSTRAINT
        ALTER TABLE ONLY public."Gestore"
        ADD CONSTRAINT gestore_pkey PRIMARY KEY ("codGestore");
```

```
--Vincolo di Chiave Primaria sul codLezione,esso è univoco.

Lezione lezione_pkey CONSTRAINT
        ALTER TABLE ONLY public."Lezione"
        ADD CONSTRAINT lezione_pkey PRIMARY KEY ("codLezione");
```

```
--Vincolo di Chiave Primaria sul codOperatore,esso è univoco.

Operatore operatore_pkey CONSTRAINT
        ALTER TABLE ONLY public."Operatore"
        ADD CONSTRAINT operatore_pkey PRIMARY KEY ("codOperatore");
```

```
--Vincolo di Chiave Primaria sul codStudente,esso è univoco.

Studente studente_pkey CONSTRAINT
        ALTER TABLE ONLY public."Studente"
        ADD CONSTRAINT studente_pkey PRIMARY KEY ("codStudente");
```

```
--Vincolo di Unicità sul codGestore,un Gestore ha una sola Sede.

Sede unique_codGestore CONSTRAINT
        ALTER TABLE ONLY public."Sede"
        ADD CONSTRAINT "unique_codGestore" UNIQUE ("codGestore");
```

```
--Vincolo di Unicità sul codiceFiscale,esiste un unico Operatore con quel codiceFiscale.

Operatore unique_codiceFiscaleOperatore CONSTRAINT
        ALTER TABLE ONLY public."Operatore"
        ADD CONSTRAINT "unique_codiceFiscaleOperatore" UNIQUE ("codiceFiscale");
```

```
--Vincolo di Chiave Primaria sul codiceFiscale,esiste un unico Studente con quel codiceFiscale.

Studente unique_codiceFiscaleStudente CONSTRAINT
        ALTER TABLE ONLY public."Studente"
        ADD CONSTRAINT "unique_codiceFiscaleStudente" UNIQUE ("codiceFiscale");
```

```
--Vincolo di Unicità sulla Lezione.

Lezione unique_lezione CONSTRAINT
        ALTER TABLE ONLY public."Lezione"
        ADD CONSTRAINT unique_lezione UNIQUE ("codLezione", "codCorso");
```

```
--Vincolo di Unicità su Coordina.

Coordina unique_coordina CONSTRAINT
        ALTER TABLE ONLY public."Coordina"
        ADD CONSTRAINT "unique_coordina"
        UNIQUE ("codCorso", "codOperatore");
```

```
--Vincolo di Unicità su Iscritti.

Iscritti unique_iscrizione CONSTRAINT
        ALTER TABLE ONLY public."Iscritti"
        ADD CONSTRAINT "unique_iscrizione"
        UNIQUE ("codStudente", "codCorso");
```

```
--Vincolo di Unicità su Prenotazioni.

Prenotazioni unique_prenotazione CONSTRAINT
        ALTER TABLE ONLY public."Prenotazioni"
        ADD CONSTRAINT "unique_prenotazione"
        UNIQUE ("codLezione", "codStudente");
```

```
--Vincolo di Forgein Key su "Utente" codiceFiscale.

Operatore references codiceFiscale FK CONSTRAINT
        ALTER TABLE ONLY public."Operatore"
        ADD CONSTRAINT "references codiceFiscale" FOREIGN KEY ("codiceFiscale")
        REFERENCES public."Utente"("codiceFiscale") ON UPDATE CASCADE ON DELETE
CASCADE NOT VALID;
```

```
--Vincolo di Forgein Key su "Corso".codCorso.

Iscritti references_codCorso FK CONSTRAINT
        ALTER TABLE ONLY public."Iscritti"
        ADD CONSTRAINT "references_codCorso" FOREIGN KEY ("codCorso")
        REFERENCES public."Corso"("codCorso") ON UPDATE CASCADE ON DELETE
CASCADE;
```

```
--Vincolo di Forgein Key su "Corso".codCorso.

Lezione references_codCorso FK CONSTRAINT
        ALTER TABLE ONLY public."Lezione"
        ADD CONSTRAINT "references_codCorso" FOREIGN KEY ("codCorso")
        REFERENCES public."Corso"("codCorso") ON UPDATE CASCADE ON DELETE
CASCADE;
```

```
--Vincolo di Forgein Key su "Corso".codCorso.

Statistiche references_codCorso FK CONSTRAINT
        ALTER TABLE ONLY public."Statistiche"
        ADD CONSTRAINT "references_codCorso" FOREIGN KEY ("codCorso")
        REFERENCES public."Corso"("codCorso") ON UPDATE CASCADE ON DELETE
CASCADE;
```

--Vincolo di Forgein Key su "Corso".codCorso.

Appartiene references_codCorso FK CONSTRAINT
ALTER TABLE ONLY public."Appartiene"
ADD CONSTRAINT "references_codCorso" FOREIGN KEY ("codCorso")
REFERENCES public."Corso"("codCorso") ON UPDATE CASCADE ON DELETE CASCADE;

--Vincolo di Forgein Key su "Corso".codCorso.

Coordina references_codCorso FK CONSTRAINT
ALTER TABLE ONLY public."Coordina"
ADD CONSTRAINT "references_codCorso" FOREIGN KEY ("codCorso")
REFERENCES public."Corso"("codCorso") ON UPDATE CASCADE ON DELETE CASCADE;

--Vincolo di Forgein Key su "Gestore".codGestore.

Corso references_codGestore FK CONSTRAINT
ALTER TABLE ONLY public."Corso"
ADD CONSTRAINT "references_codGestore" FOREIGN KEY ("codGestore")
REFERENCES public."Gestore"("codGestore") ON UPDATE CASCADE ON DELETE CASCADE NOT VALID;

--Vincolo di Forgein Key su "Gestore".codGestore.

Sede references_codGestore FK CONSTRAINT
ALTER TABLE ONLY public."Sede"
ADD CONSTRAINT "references_codGestore" FOREIGN KEY ("codGestore")
REFERENCES public."Gestore"("codGestore") ON UPDATE CASCADE ON DELETE CASCADE NOT VALID;

--Vincolo di Forgein Key su "Lezione".codLezione.

Prenotazioni references_codLezione FK CONSTRAINT
ALTER TABLE ONLY public."Prenotazioni"
ADD CONSTRAINT "references_codLezione" FOREIGN KEY ("codLezione")
REFERENCES public."Lezione"("codLezione") NOT VALID;

```
--Vincolo di Forgein Key su "Operatore".codOperatore.

Coordina references_codOperatore FK CONSTRAINT
        ALTER TABLE ONLY public."Coordina"
        ADD CONSTRAINT "references_codOperatore" FOREIGN KEY ("codOperatore")
        REFERENCES public."Operatore"("codOperatore") ON UPDATE CASCADE ON
DELETE CASCADE;
```

```
--Vincolo di Forgein Key su "Studente".codStudente.

Prenotazioni references_codStudente FK CONSTRAINT
        ALTER TABLE ONLY public."Prenotazioni"
        ADD CONSTRAINT "references_codStudente" FOREIGN KEY ("codStudente")
        REFERENCES public."Studente"("codStudente") ON UPDATE CASCADE ON DELETE
CASCADE;
```

```
--Vincolo di Forgein Key su "Utente".codiceFiscale.

Studente references_codiceFiscale FK CONSTRAINT
        ALTER TABLE ONLY public."Studente"
        ADD CONSTRAINT "references_codiceFiscale" FOREIGN KEY ("codiceFiscale")
        REFERENCES public."Utente"("codiceFiscale") ON UPDATE CASCADE ON DELETE
CASCADE NOT VALID;
```

```
--Vincolo di Forgein Key su "Autenticazione".email.

Utente references_email FK CONSTRAINT
        ALTER TABLE ONLY public."Utente"
        ADD CONSTRAINT references_email FOREIGN KEY (email)
        REFERENCES public."Autenticazione"(email) ON UPDATE CASCADE ON DELETE
CASCADE NOT VALID;
```

```
--Vincolo di Forgein Key su "Autenticazione".email.

Gestore references_email FK CONSTRAINT
        ALTER TABLE ONLY public."Gestore"
        ADD CONSTRAINT references_email FOREIGN KEY (email)
        REFERENCES public."Autenticazione"(email) ON UPDATE CASCADE ON DELETE
CASCADE NOT VALID;
```

```
--Vincolo di Forgein Key su "AreaTematica". tag.

Appartiene references_tag FK CONSTRAINT
        ALTER TABLE ONLY public."Appartiene"
        ADD CONSTRAINT references_tag FOREIGN KEY (tag)
        REFERENCES public."AreaTematica"(tag) NOT VALID;
```

Definizione dei domini

```
--Dominio del valore di sesso in Utente

CREATE DOMAIN public.sesso AS character(1)
CONSTRAINT sesso_check CHECK (((VALUE = 'M'::bpchar) OR (VALUE = 'F'::bpchar)));
```

Dizionario dei vincoli

Vincolo	***Tipo***	***Descrizione***
unique_codiceFiscaleOperatore	*Intrarelazionale*	Il codice fiscale di un operatore è univoco
unique_codiceFiscaleStudente	*Intrarelazionale*	Il codice fiscale di uno studente è univoco
unique_codGestore	*Intrarelazionale*	Il codice Gestore in sede è unico.
unique_lezione	*Intrarelazionale*	Il codice Corso e il codice Lezione sono unici.
unique_prenotazione	*Intrarelazionale*	Il codice Lezione e il codice Studente sono unici.
unique_iscrizione	*Intrarelazionale*	Il codice Corso e il codice Studente sono unici.
unique_coordina	*Intrarelazionale*	Il codice Corso e il codice Operatore sono unici.
sesso_check	*Intrarelazionale*	Il valore di sesso può essere 'M' o 'F'.

Definizione delle sequenze

```
--Sequence per la generazione di codCorso univoco.

CREATE SEQUENCE public.seqcorso
    START WITH 0
    INCREMENT BY 1
    MINVALUE 0
    MAXVALUE 9999999
    CACHE 1;
```

```
--Sequence per la generazione di codGestore univoco.

CREATE SEQUENCE public.seqgestore
    START WITH 0
    INCREMENT BY 1
    MINVALUE 0
    MAXVALUE 9999999
    CACHE 1;
--Sequence per la generazione di codLezione univoco.

CREATE SEQUENCE public.seqlezione
    START WITH 0
    INCREMENT BY 1
    MINVALUE 0
    MAXVALUE 9999999
    CACHE 1;
```

```
--Sequence per la generazione di codOperatore univoco.

CREATE SEQUENCE public.seqoperatore
    START WITH 0
    INCREMENT BY 1
    MINVALUE 0
    MAXVALUE 9999999
    CACHE 1;
```

```
--Sequence per la generazione di codStudente univoco.

CREATE SEQUENCE public.seqstudente
    START WITH 0
    INCREMENT BY 1
    MINVALUE 0
    MAXVALUE 9999999
    CACHE 1;
```

Definizione dei Trigger

```
--Trigger Function che,in seguito all'UPDATE sulla tabella "Prenotazioni", calcola le Statistiche sulle Presenze.

CREATE FUNCTION public.calcolASstatistiche() RETURNS trigger
LANGUAGE plpgsql
DECLARE
   ris_stat "Statistiche"%RowType;
BEGIN
SELECT MIN(coalesce(numero_presenti, 0)),
          MAX(coalesce(numero_presenti, 0)),
          avg(coalesce(numero_presenti, 0)) into ris_stat
FROM (SELECT numero_presenti
   FROM (SELECT "codLezione"
      FROM "Prenotazioni" NATURAL JOIN "Lezione"
      WHERE "codCorso" IN(SELECT "codCorso"
                  FROM "Lezione"
                  WHERE "codLezione"=new."codLezione")
   GROUP BY "codLezione") AS res NATURAL LEFT JOIN
   (SELECT "codLezione",count(*) AS "numero_presenti"
    FROM "Prenotazioni" NATURAL JOIN "Lezione"
    WHERE "codCorso" IN(SELECT "codCorso"
                  FROM "Lezione"
                  WHERE "codLezione"=new."codLezione")
   AND "Presente"=true
    GROUP BY "codLezione"
) AS res1)AS results;


   UPDATE "Statistiche" SET "presenzeMASsime"=ris_stat."presenzeMassime",
                  "presenzeMinime"=ris_stat."presenzeMinime",
                  "presenzeMedie"=ris_stat."presenzeMedie"
   WHERE "codCorso" IN(SELECT "codCorso"
                FROM "Lezione"
                WHERE "codLezione"=NEW."codLezione");
   RETURN new;
END;
```

--Trigger Function che, in seguito all'UPDATE, DELETE, INSERT sulla tabella "Iscritti", calcola la "percentuale di riempimento" di un Corso e la inserisce nelle Statistiche.

```
CREATE FUNCTION public.calcoloiscritti() RETURNS trigger
LANGUAGE plpgsql
DECLARE
        percentuale integer;
        numeroiscritti integer;
BEGIN
   SELECT count(*) into numeroiscritti
   FROM "Iscritti"
   WHERE "codCorso"=new."codCorso" AND "Richiesta"=true;
   IF numeroiscritti=0 THEN
   percentuale=0;
   ELSE
      SELECT 100/("iscrizioniMassime"/numeroiscritti) into percentuale
      FROM "Corso"
      WHERE "codCorso"=new."codCorso";
   END IF;

   UPDATE "Statistiche"
   SET "percentualeRempimento"=percentuale
   WHERE "codCorso"= new."codCorso";

   RETURN new;
END;
```

--Trigger Function che,in seguito all'INSERT sulla tabella "Iscritti",setta il valore di Richiesta in base al tipo di Corso,se è Privato la Richiesta sarà FALSE se è Pubblico la Richiesta sarà TRUE.

```
CREATE FUNCTION public.checkcorsopublic() RETURNS trigger
LANGUAGE plpgsql
DECLARE
   corso "Corso"%rowtype;
BEGIN
   SELECT *
   FROM "Corso"
   INTO corso
   WHERE "codCorso"= NEW."codCorso";
   if(corso."Privato"=false)then
   NEW."Richiesta"=true;
   RETURN NEW;
   else
      NEW."Richiesta"=false;
      RETURN NEW;
END if;
END;
```

```
--Trigger Function che,in seguito all'UPDATE sulla tabella "Prenotazioni",setta il valore di Idoneo
a TRUE se si è raggiunto il tasso di presenze minimo.

CREATE FUNCTION public.checkidoneo() RETURNS trigger
LANGUAGE plpgsql
DECLARE
        presenze integer;
        lezioniminime real;
        tasso numeric;

BEGIN
SELECT "tassoPresenzeMinime"*"numeroLezioni into tasso
FROM "Corso"
WHERE "codCorso" IN (Select "codCorso"
                FROM "Lezione"
                WHERE "Lezione"."codLezione"= new."codLezione");

lezioniminime=round(tasso/100);

SELECT count(*) into presenze
FROM "Prenotazioni" NATURAL JOIN "Lezione"
WHERE "codStudente"=NEW."codStudente"
AND "codCorso" IN (Select "codCorso"
   FROM "Lezione"
   WHERE "Lezione"."codLezione"=new."codLezione")
AND "Presente"=true;

IF lezioniminime <= presenze THEN
UPDATE "Iscritti" SET "Idoneo"=true
WHERE "codStudente"=NEW."codStudente" AND "codCorso" IN (Select "codCorso"
   FROM "Lezione"
   WHERE "Lezione"."codLezione"=new."codLezione");
ELSE
UPDATE "Iscritti" SET "Idoneo"=false
WHERE "codStudente"=NEW."codStudente" AND "codCorso" IN (Select "codCorso"
   FROM "Lezione"
   WHERE "Lezione"."codLezione"=new."codLezione");
END IF;
RETURN new;
END;
```

```
--Trigger Function che non permette l'iscrizione nel caso in cui è stato raggiunto il numero massimo di Iscritti.

CREATE FUNCTION public.checkiscritti() RETURNS trigger
LANGUAGE plpgsql
DECLARE
        conteggio integer;
        massimo integer;
BEGIN
SELECT count(*) into conteggio
FROM "Iscritti"
WHERE "codCorso"=new."codCorso";

SELECT "iscrizioniMassime" into massimo
FROM "Corso"
WHERE "codCorso"=new."codCorso";
IF conteggio<massimo THEN
RETURN new;
else
RETURN null;
END IF;
END;
```

```
--Trigger Function che non permette l'inserimento di una Lezione nel caso in cui è stato raggiunto il numero di Lezioni programmate.

CREATE FUNCTION public.checklezione() RETURNS trigger
LANGUAGE plpgsql
DECLARE
        conteggio integer;
        massimo integer;
BEGIN
SELECT count(*) into conteggio
FROM "Lezione"
WHERE "codCorso"=new."codCorso";

SELECT "numeroLezioni" into massimo
FROM "Corso"
WHERE "codCorso"=new."codCorso";

IF conteggio<massimo THEN
RETURN new;
else
RETURN null;
END IF;
END;
```

```
--Trigger Function che, nel caso in cui è presente un Operatore che non coordina Corsi, elimina l'operatore dalla tabella Operatore.

CREATE FUNCTION public.checkoperatore() RETURNS trigger
LANGUAGE plpgsql
BEGIN
DELETE FROM "Operatore"
WHERE "codOperatore" NOT IN(SELECT "codOperatore"
                    FROM "Coordina");
RETURN OLD;
END;
```

```
--Trigger Function che, nel caso in cui è presente uno Studente che non coordina Corsi,elimina lo studente dalla tabella Studente.

CREATE FUNCTION public.checkstudente() RETURNS trigger
LANGUAGE plpgsql
DELETE FROM "Studente"
WHERE "codStudente" NOT IN(SELECT "codStudente"
                    FROM "Iscritti");
RETURN OLD;
END;
```

```
--Trigger Function che genera le statistiche di Default per un nuovo Corso.

CREATE FUNCTION public.creaStatistiche() RETURNS trigger
LANGUAGE plpgsql
BEGIN
INSERT INTO "Statistiche" VALUES(0,0,0,0,NEW."codCorso");
RETURN new;
END
```

```
--Trigger Function che Setta la descrizione nel caso in cui è " uguale a "Nessuna Descrizione".

CREATE FUNCTION public."defaultvalueDescrizione"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
IF NEW."descrizione" =" THEN
NEW."descrizione"='Nessuna Descrizione';
END IF;
RETURN NEW;
END;
```

```
--Trigger Function che genera un codCorso univoco.

CREATE FUNCTION public."generatecodCorso"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
   LOOP
        NEW."codCorso" = ('C' || CAST (nextval('seqcorso') AS char));
        if NOT EXISTS (select "codCorso"  FROM "Corso"
                        WHERE "codCorso"=new."codCorso") then
           RETURN NEW;
        END if;
   END LOOP;
END;
```

```
--Trigger Function che genera un codGestore univoco.

CREATE FUNCTION public."generatecodGestore"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
   LOOP
        NEW."codGestore" = ('G' || CAST (nextval('seqgestore') AS char));
        if NOT EXISTS (select "codGestore" FROM "Gestore"
                WHERE "codGestore"= new."codGestore”) then
           RETURN NEW;
        END if;
   END LOOP;
END;
```

```
--Trigger Function che genera un codLezione univoco.

CREATE FUNCTION public."generatecodLezione"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
   LOOP
        NEW."codLezione" = ('L' || CAST (nextval('seqlezione') AS char));
        if NOT EXISTS (select "codLezione" FROM "Lezione"
                        WHERE "codLezione"= new."codLezione" ) then
           RETURN NEW;
        END if;
   END LOOP;
END;
```

```
--Trigger Function che genera un codOperatore univoco.

CREATE FUNCTION public."generatecodOperatore"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
   LOOP
        NEW."codOperatore" = ('O' || CAST (nextval('seqoperatore') AS char));
        if NOT EXISTS (select "codOperatore" FROM "Operatore"
              WHERE "codOperatore"= new."codOperatore”) then
           RETURN NEW;
        END if;
   END LOOP;
END;
```

```
--Trigger Function che genera un codStudente univoco.

CREATE FUNCTION public."generatecodStudente"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
   LOOP
        NEW."codStudente" =('S' || CAST(nextval('seqstudente') AS char));
        if NOT EXISTS (select "codStudente" FROM "Studente" WHERE "codStudente"=
new."codStudente" ) then
           RETURN NEW;
        END if;
   END LOOP;

END;
```

```
--Trigger Function che non permette l'inserimento di un tag già presente.

CREATE FUNCTION public.noprimarykeyviolation() RETURNS trigger
LANGUAGE plpgsql
BEGIN
        if NOT EXISTS (select "tag" FROM "AreaTematica" WHERE "tag"= new."tag" ) then
           RETURN NEW;
        END if;
RETURN null;
END;
```

```
--Trigger Function che resetta la Sequence che permette il calcolo di un codCorso univoco.

CREATE FUNCTION public."resetseqCorso"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
ALTER SEQUENCE seqcorso
RESTART WITH 0;
RETURN NEW;
END;
```

```
--Trigger Function che resetta la Sequence che permette il calcolo di un codGestore univoco.

CREATE FUNCTION public."resetseqGestore"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
ALTER SEQUENCE seqgestore
RESTART WITH 0;
RETURN NEW;
END;
```

```
--Trigger Function che resetta la Sequence che permette il calcolo di un codLezione univoco.

CREATE FUNCTION public."resetseqLezione"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
ALTER SEQUENCE seqlezione
RESTART WITH 0;
RETURN NEW;
END;
```

```
--Trigger Function che resetta la Sequence che permette il calcolo di un codOperatore univoco.

CREATE FUNCTION public."resetseqOperatore"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
ALTER SEQUENCE seqoperatore
RESTART WITH 0;
RETURN NEW;
END;
```

```
--Trigger Function che resetta la Sequence che permette il calcolo di un codStudente univoco.

CREATE FUNCTION public."resetseqStudente"() RETURNS trigger
LANGUAGE plpgsql
BEGIN
ALTER SEQUENCE seqstudente
RESTART WITH 0;
RETURN NEW;
END;
```