Basi di Dati

Progetto A.A. 2023/2024

SISTEMA DI GESTIONE DI CORSI DI LINGUE STRANIERE

0315886

Greta Luna Ancora

Indice

1. Descrizione del Minimondo 2

2. Analisi dei Requisiti 3

3. Progettazione concettuale 6

4. Progettazione logica 10

5. Progettazione fisica 19

1. Descrizione del Minimondo

Si progetti un sistema informativo per la gestione dei corsi di lingua inglese, tenuti presso un istituto di insegnamento. I corsi sono organizzati per livelli. Ciascun livello è identificato dal nome del livello stesso (ad esempio Elementary, Intermediate, First Certificate, Advanced, Proficiency); inoltre è specificato il nome del libro di testo e se viene richiesto di sostenere un esame finale.

I corsi sono identificati univocamente dal nome del livello cui afferiscono e da un codice progressivo, necessario per distinguere corsi che fanno riferimento allo stesso livello. Per ciascun corso sono note la data di attivazione, il numero e le informazioni anagrafiche degli iscritti e l'elenco dei giorni ed orari in cui è tenuto.

Per gli insegnanti sono noti il nome, l'indirizzo, la nazione di provenienza, ed i corsi a cui sono stati assegnati. Ad un corso può essere assegnato più di un insegnante, assicurandosi che in una determinata fascia oraria un insegnante sia assegnato ad un solo corso.

Per gli allievi sono noti il nome, un recapito, il corso a cui sono iscritti, la data di iscrizione al corso e il numero di assenze fatte finora (è di interesse tenere traccia dei giorni specifici in cui un allievo è stato assente).

Il personale amministrativo della scuola deve poter inserire all'interno del sistema informativo tutte le informazioni legate ai corsi ed agli insegnanti e possono generare dei report indicanti, su base mensile, quali attività hanno svolto gli insegnanti. Il personale di segreteria gestisce le iscrizioni degli utenti della scuola ai corsi.
Gli insegnanti possono generare dei report indicanti la propria agenda, su base settimanale.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
2	Istituto di insegnamento	Scuola	Disambiguazione sinonimi.
10	Iscritti	Studenti	Disambiguazione sinonimi.
16, 18	Allievi	Studenti	Disambiguazione sinonimi.
23	Utenti	Studenti	Disambiguazione sinonimi.
16	Corso	Corsi	A valle del contatto con il cliente, si è capito che uno studente può seguire più corsi contemporaneamente.
20	Personale amministrativo	Personale di segreteria	Non c'è differenza semantica nel minimondo di riferimento.
22	Attività	Lezioni	Disambiguazione sinonimi.
24	Agenda	Lezioni	Disambiguazione sinonimi.

Specifica disambiguata

Si progetti un sistema informativo per la gestione dei corsi di lingua inglese, tenuti presso una scuola. I corsi sono organizzati per livelli. Ciascun livello è identificato dal nome del livello stesso (ad esempio Elementary, Intermediate, First Certificate, Advanced, Proficiency); inoltre è specificato il nome del libro di testo e se viene richiesto di sostenere un esame finale.

I corsi sono identificati univocamente dal nome del livello cui afferiscono e da un codice progressivo, necessario per distinguere corsi che fanno riferimento allo stesso livello. Per ciascun corso sono note la data di attivazione, il numero e le informazioni anagrafiche degli studenti e l'elenco dei giorni ed orari in cui è tenuto.

Per gli insegnanti sono noti il nome, l'indirizzo, la nazione di provenienza, ed i corsi a cui sono stati assegnati. Ad un corso può essere assegnato più di un insegnante, assicurandosi che in una determinata fascia oraria un insegnante sia assegnato ad un solo corso.

Per gli studenti sono noti il nome, un recapito, i corsi a cui sono iscritti, la data di iscrizione e il numero di assenze fatte per ogni corso (è di interesse tenere traccia dei giorni specifici in cui uno studente è stato assente).

Il personale di segreteria della scuola deve poter inserire all'interno del sistema informativo tutte le

informazioni legate ai corsi ed agli insegnanti.

Il personale di segreteria della scuola può generare dei report indicanti, su base mensile, quali lezioni hanno tenuto gli insegnanti.

Il personale di segreteria della scuola gestisce le iscrizioni degli studenti della scuola ai corsi.

Gli insegnanti possono generare dei report indicanti le proprie lezioni, su base settimanale.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Livello	Ogni corso afferisce ad un livello di inglese.		Corso
Corso	Insieme di lezioni fruite dagli studenti.		Insegnante, Lezione, Studente, Livello
Insegnante	Colui che tiene le lezioni.		Corso
Studente	Colui che segue le lezioni.	Utente, Iscritto	Corso, Lezione
Lezione	Attività didattica tenuta da un insegnante.	Attività, Agenda	Corso, Studente

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Livello
Ciascun livello è identificato dal nome del livello stesso (ad esempio Elementary, Intermediate, First Certificate, Advanced, Proficiency); inoltre è specificato il nome del libro di testo e se viene richiesto di sostenere un esame finale.

Frasi relative a Corso
I corsi sono organizzati per livelli. I corsi sono identificati univocamente dal nome del livello cui afferiscono e da un codice progressivo, necessario per distinguere corsi che fanno riferimento allo stesso livello. Per ciascun corso sono note la data di attivazione, il numero e le informazioni anagrafiche degli studenti e l'elenco dei giorni ed orari in cui è tenuto.

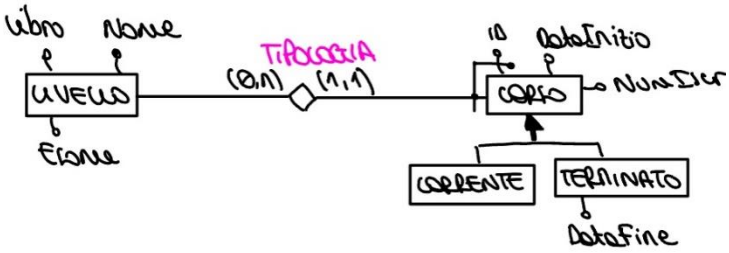
Frasi relative a Insegnante
Per gli insegnanti sono noti il nome, l'indirizzo, la nazione di provenienza, ed i corsi a cui sono stati assegnati. Ad un corso può essere assegnato più di un insegnante, assicurandosi che in una determinata fascia oraria un insegnante sia assegnato ad un solo corso. Gli insegnanti possono generare dei report indicanti le proprie lezioni, su base settimanale.

Frasi relative a Studente
Per gli studenti sono noti il nome, un recapito, i corsi a cui sono iscritti, la data di iscrizione e il numero di assenze fatte per ogni corso (è di interesse tenere traccia dei giorni specifici in cui uno studente è stato assente).

Frasi relative a Personale di segreteria
Il personale di segreteria della scuola deve poter inserire all'interno del sistema informativo tutte le informazioni legate ai corsi ed agli insegnanti. Il personale di segreteria della scuola può generare dei report indicanti, su base mensile, quali lezioni hanno tenuto gli insegnanti. Il personale di segreteria della scuola gestisce le iscrizioni degli studenti della scuola ai corsi.

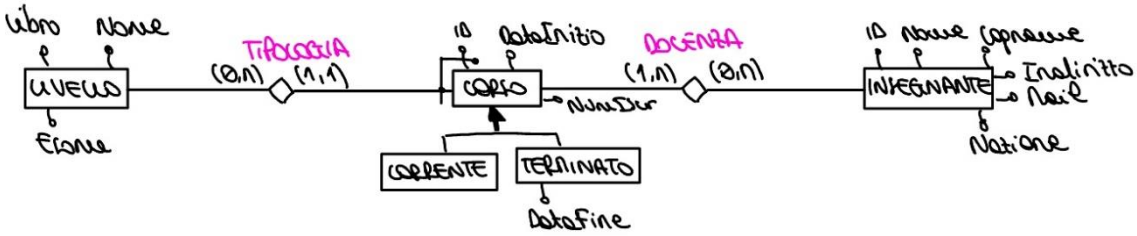
3. Progettazione concettuale

Costruzione dello schema E-R



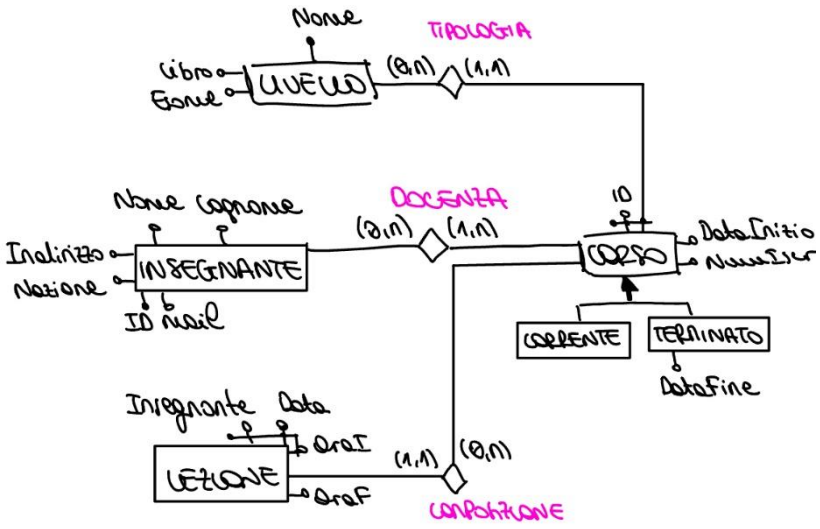
Si è iniziato definendo le due entità Livello e Corso, collegate tra loro tramite la relazione Tipologia.

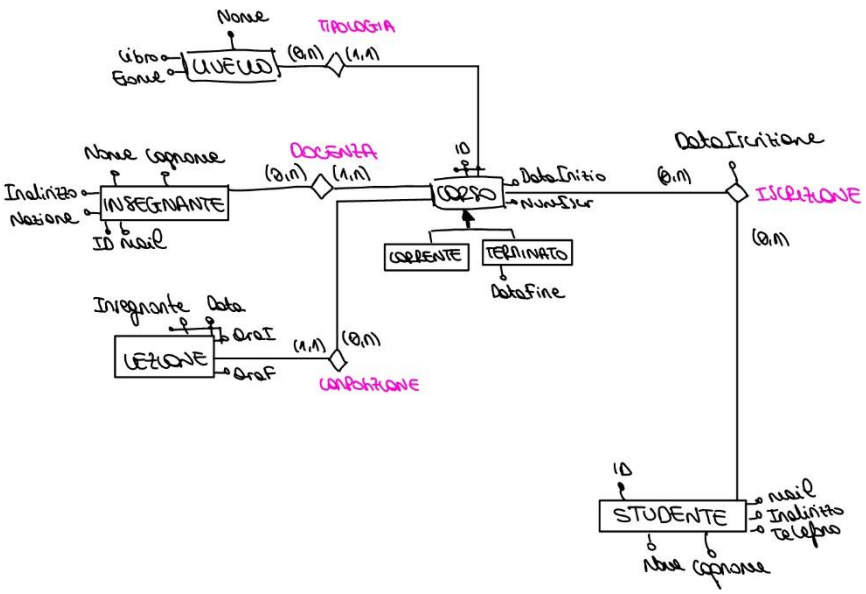
Per quanto riguarda gli attributi "le informazioni anagrafiche degli studenti e l'elenco dei giorni ed orari in cui è tenuto" relativi all'entità Corso, si è deciso di modellarli tramite ulteriori entità/relazioni, in quanto concetti importanti per il dominio di interesse.



L'entità Insegnante è stata connessa all'entità corso tramite la relazione Docenza.

Le informazioni concernenti i giorni e gli orari in cui un corso è tenuto vengono mantenute nell'entità Lezione, collegata all'entità Corso tramite la relazione Composizione. È stato inserito l'attributo Insegnante all'interno dell'entità Lezione, in quanto un corso può essere tenuto da più insegnanti, e così facendo si tiene traccia dell'insegnante che ha tenuto una specifica lezione.

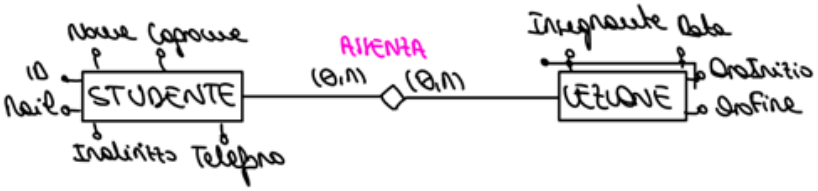




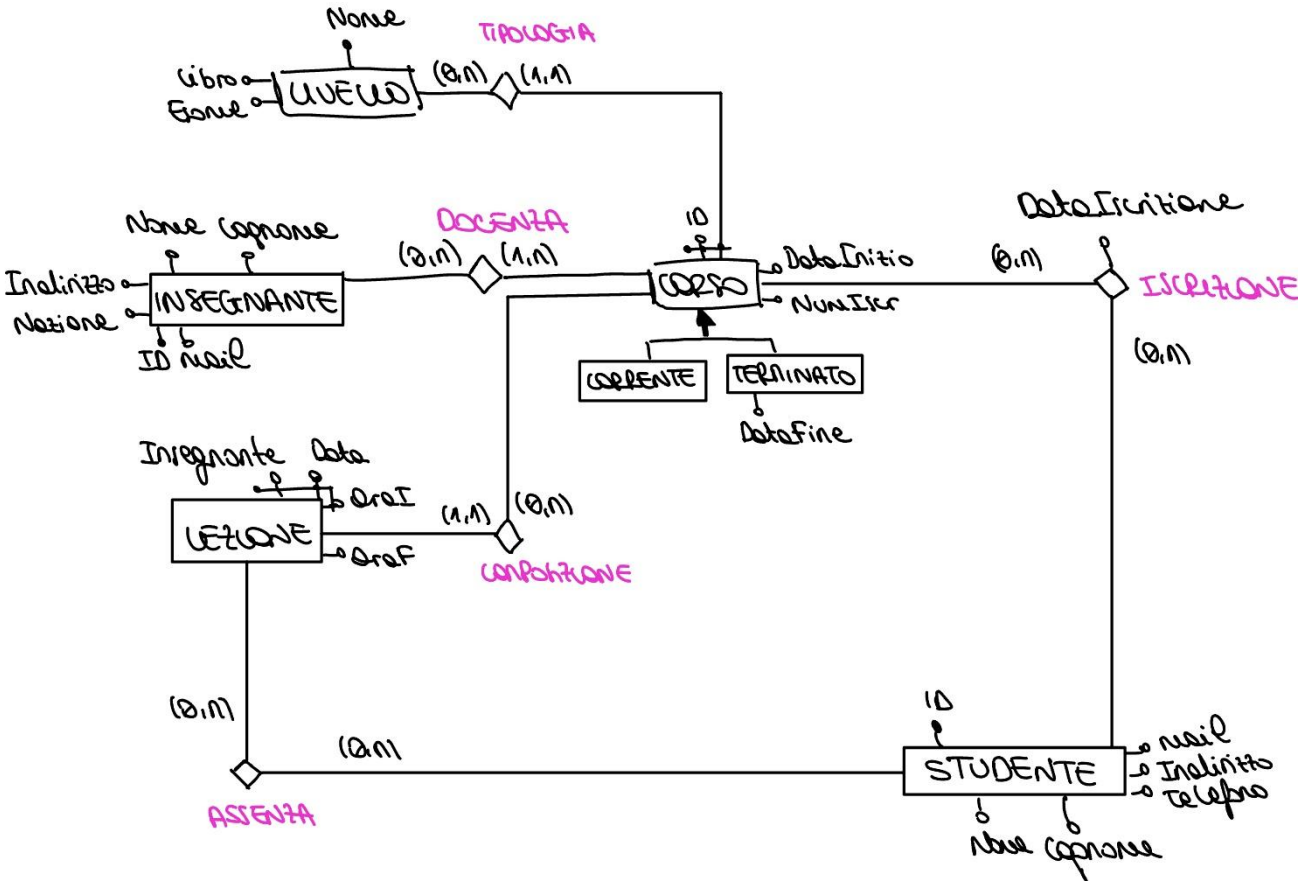
Per quanto riguarda l'entità Studente, viene connessa all'entità Corso tramite la relazione Iscrizione, la quale tiene traccia dei corsi a cui è iscritto lo studente e della relativa data di iscrizione.

Si è scelto di non reificare l'attributo della relazione Iscrizione, in quanto sembra logico che uno studente si iscriva al più una volta ad uno specifico corso.

Le informazioni relative al numero di assenze dello studente a un corso ed i relativi giorni si è scelto di modellarle tramite la relazione Assenza, che collega l'entità Studente all'entità Lezione.



Integrazione finale



Regole aziendali

La data in cui è tenuta una lezione deve appartenere al periodo in cui il relativo corso è tenuto.

L'ora di inizio di una lezione deve essere precedente all'ora di fine della stessa.

Gli orari delle lezioni tenute dallo stesso insegnante non possono sovrapporsi.

Gli orari delle lezioni appartenenti allo stesso corso non possono sovrapporsi.

La data di inizio di un corso deve precedere la data di fine dello stesso.

La data di iscrizione di uno studente a un corso deve precedere la data di fine dello stesso.

Uno studente può risultare assente solo a lezioni appartenenti a corsi a cui è iscritto.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Livello	Livello di inglese coperto dai corsi della scuola.	Libro, Esame	Nome
Corso	Insieme di lezioni, erogate da uno o più professori, afferente a un livello.	DataInizio, DataFine, NumIscr	Livello, ID
Insegnante	Le informazioni sull'insegnante che tiene le lezioni.	Nome, Cognome, Indirizzo, Mail, Nazione	ID
Studente	Le informazioni sullo studente che è iscritto ai corsi.	Nome, Cognome, Indirizzo, Mail, Telefono	ID
Lezione	Le informazioni rispetto a giorni e orari in cui è tenuta una lezione.	OraFine	Insegnante, DataLezione, OraInizio

4. Progettazione logica

Volume dei dati

Nell'analisi, si ipotizza che il sistema mantenga i dati relativi a insegnanti, studenti e corsi per un periodo non superiore a dieci anni.

Si considera che ogni anno la scuola tenga circa 18 corsi, uno per ogni livello ogni trimestre, ognuno dei quali si compone di due lezioni a settimana.

Inoltre, si suppone che un insegnante tenga due corsi a trimestre, gli insegnanti impiegati all'anno siano circa tre, e che ogni anno vengano aggiunti due nuovi insegnanti (supplenti/di ruolo).

Si suppone che ogni corso abbia all'incirca 20 studenti, che ogni anno l'80% degli studenti sia un nuovo iscritto e che uno studente segua un corso all'anno.

Le informazioni legate ai livelli coperti, invece, sono considerate fisse, data la dimensione relativamente piccola della scuola.

Concetto nello schema	Tipo[1]	Volume atteso
Livello	E	6
Corso	E	180
Insegnante	E	21
Lezione	E	4.320
Studente	E	2.950
Assenza	R	23.895
Tipologia	R	180
Composizione	R	4.320
Iscrizione	R	7.965
Docenza	R	220

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
AM1	Aggiungi corso	18/anno
AM2	Aggiungi lezione	12/settimana
AM3	Aggiungi insegnante	2/anno
AM4	Aggiungi studente	288/anno
AM5	Assegna corso	22/anno
AM6	Registra iscrizione	360/anno
AM7	Registra assenza	1080/anno
AM8	Ottieni info corso	360/anno

[1] Indicare con E le entità, con R le relazioni

AM9	Ottieni info insegnante	20/anno
AM10	Ottieni info studente	360/anno
AM11	Lista assenze studente	360/anno
AM12	Lista iscrizioni studente	360/anno
AM13	Lista lezioni corso	54/anno
AM14	Lista corsi insegnante	54/anno
AM15	Aggiorna data fine corso	18/anno
AM16	Lista corsi	360/anno
AM17	Lista insegnanti	130/anno
AM18	Lista studenti	792/anno
AM19	Lista studenti corso	1.080/anno
AM20	Lista livelli	18/anno
R1	Report mensile lezioni insegnante	6/mese
R2	Report mensile lezioni	1/mese
R3	Report lezioni settimana corrente	12/settimana
R4	Report lezioni prossima settimana	6/settimana
L1	Login	20/giorno
E1	Elimina vecchie entries	3/anno

Costo delle operazioni

AM1 - Aggiungi corso

1 accesso in scrittura Corso, 1 accesso in scrittura Tipologia

Costo totale: 4

Numero accessi:72/anno

AM2 - Aggiungi lezione

1 accesso in scrittura Lezione, 1 accesso in scrittura Composizione

Costo totale: 4

Numero accessi: 48/settimana

AM3 - Aggiungi insegnante

1 accesso in scrittura Insegnate

Costo totale: 2

Numero accessi: 2/anno

AM4 – Aggiungi Studente

1 accesso in scrittura Studente

Costo totale: 2

Numero accessi: 576 /anno

AM5 - Assegna corso

1 accesso in lettura Insegnante, 1.5 accessi in scrittura Docenza

Costo totale: 3

Numero accessi: 81/anno

AM6 - Registra iscrizione

Caso non ridondante

1 accesso in lettura Studente, 1 accesso in scrittura Iscrizione

Costo totale: 3

Numero accessi: 1.080/anno

Caso ridondante

1 accesso in lettura Studente, 1 accesso in scrittura Iscrizione, 1 accesso in lettura Corso, 1 accesso in scrittura Corso

Costo totale: 6

Numero accessi: 2.160/anno

AM7 - Registra assenza

1 accesso in lettura Studente, 1 accesso in scrittura Assenza

Costo totale: 3

Numero accessi: 3.240/anno

AM8 - Ottieni info corso

Caso non ridondante

1 accesso in lettura Corso, 20 accessi in lettura Iscrizione

Costo totale: 21

Numero accessi: 7.560/anno

Caso ridondante

1 accesso in lettura Corso

Costo totale: 1

Numero accessi: 360/anno

AM9 - Ottieni info insegnante

1 accesso in lettura Insegnante

Costo totale: 1

Numero accessi: 20/anno

AM10 - Ottieni info studente

1 accesso in lettura Studente

Costo totale: 1

Numero accessi: 360/anno

AM11 - Lista assenze studente

1 accesso in lettura Studente, 3 accessi in lettura Assenza

Costo totale: 4

Numero accessi: 1.440/anno

AM12 - Lista iscrizioni studente

1 accesso in lettura Studente, 1 accesso in lettura Iscrizione

Costo totale: 2

Numero accessi: 720/anno

AM13 - Lista lezioni corso

1 accesso in lettura Corso, 24 accessi in lettura Composizione, 24 accessi in lettura Lezione

Costo totale: 49

Numero accessi: 2.646/anno

AM14 - Lista corsi insegnante

1 accesso in lettura Insegnante, 1 accesso in lettura Docenza, 6 accessi in lettura Corso

Costo totale: 8

Numero accessi: 432/anno

AM15 - Aggiorna data fine corso

1 accesso in scrittura Corso

Costo totale: 2

Numero accessi: 36/anno

AM16 - Lista corsi

Caso non ridondante

1 accesso in lettura Corsi, 20 accessi in lettura Iscrizioni

Costo totale: 21

Numero accessi: 7.560/anno

Caso ridondante

1 accesso in lettura Corsi

Costo totale: 1

Numero accessi: 360/anno

AM17 - Lista insegnanti

1 accesso in lettura Insegnante

Costo totale: 1

Numero accessi: 130/anno

AM18 - Lista studenti

1 accesso in lettura Studenti

Costo totale: 1

Numero accessi: 792/anno

AM19 - Lista studenti corso

1 accesso in lettura Corso, 20 accessi in lettura Iscrizione, 20 accessi in lettura Studente

Costo totale: 41

Numero accessi: 44.280/anno

AM20 - Lista livelli

1 accesso in lettura Livello

Costo totale: 1

Numero accessi: 18/anno

R1 - Report mensile lezioni insegnante

16 accessi in lettura Lezione

Costo totale: 16

Numero accessi: 48/mese

R2 - Report mensile lezioni
51 accessi in lettura Lezione
Costo totale: 51
Numero accessi: 51/mese

R3 – Report lezioni settimana corrente
4 accessi in lettura Lezione
Costo totale: 4
Numero accessi: 48/ settimana

R4 - Report lezioni settimana prossima
4 accesso in lettura Lezione
Costo totale: 4
Numero accessi: 24/settimana

L1 - Login
1 accesso lettura Utente
Costo totale: 1
Numero accessi: 20/giorno

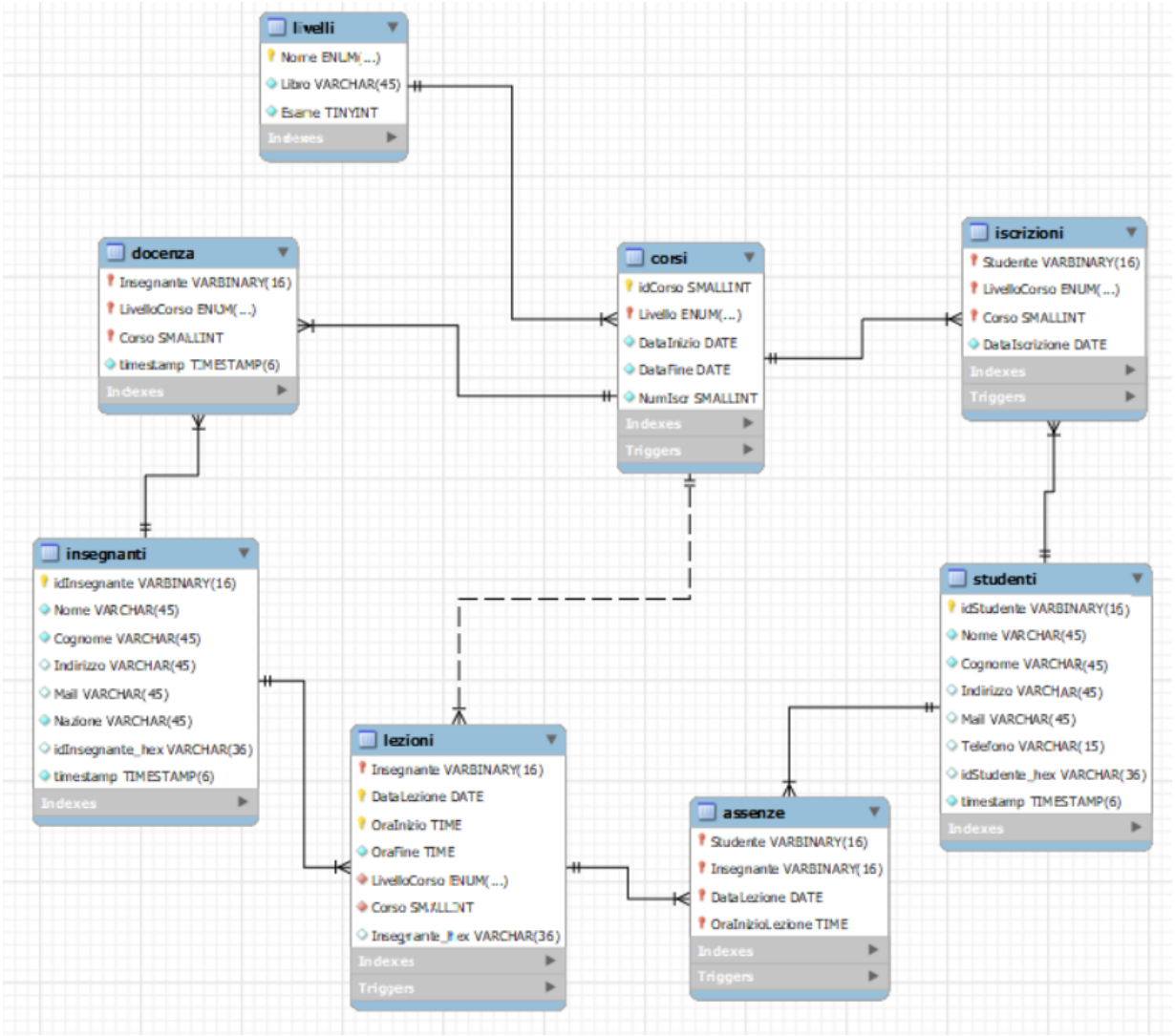
Ristrutturazione dello schema E-R

1) Analisi delle ridondanze: Non sono presenti ridondanze nello schema, in quanto si è scelto di non rappresentare il Numero di partecipanti in Corso come attributo, bensì tramite il numero di occorrenze della relazione Iscrizione.
   Questa decisione deriva dal seguente ragionamento:
   - Supponendo di utilizzare il tipo smallint (2 byte) per mantenere il numero di iscritti a un corso, considerando un volume medio di 90 corsi, sarebbero necessari 180 byte di memoria.
   - Le operazioni coinvolte sono: Registra iscrizione (AM6), ottieni info corso (AM8) e lista corsi (AM16).

Dall'analisi precedentemente svolta si evince che si ha un minor numero di accessi nel caso ridondante.

Si sceglie perciò di implementare il caso ridondante.

2) Eliminazione delle gerarchie: Si sceglie di accorpare le due entità figlie Corso Corrente e Corso Terminato nell'entità padre Corso, in quanto le operazioni sulla base di dati non fanno distinzione tra le occorrenze e tra gli attributi delle entità padre/figlie.
3) Scelta degli identificatori principali: Si sceglie di utilizzare un codice interno per l'identificazione delle entità Insegnante e Studente.



Traduzione di entità e associazioni

LIVELLO (Nome, Libro, Esame)

INSEGNANTE (idInsegnante, Nome, Cognome, Indirizzo, Mail, Nazione)

LEZIONE (Insegnante, DataLezione, OraInizio, OraFine, LivelloCorso, Corso)

STUDENTE (idStudente, Nome, Cognome, Indirizzo, Mail, Telefono)

CORSO (Livello, idCorso, DataInizio, DataFine, NumIscr)

ISCRIZIONE (Studente, LivelloCorso, Corso, DataIscrizione)

ASSENZA (Studente, Insegnante, DataLezione, OraInizioLezione)

DOCENZA (Insegnante, LivelloCorso, Corso)

Vincoli Di integrità:

Lezione (Insegnante) ⊆ Insegnante(idInsegnante)

Lezione (LivelloCorso, Corso) ⊆ Corso(Livello, idCorso)

Corso(Livello) ⊆ Livello(Nome)

Iscrizione (Studente) ⊆ Studente(idStudente)

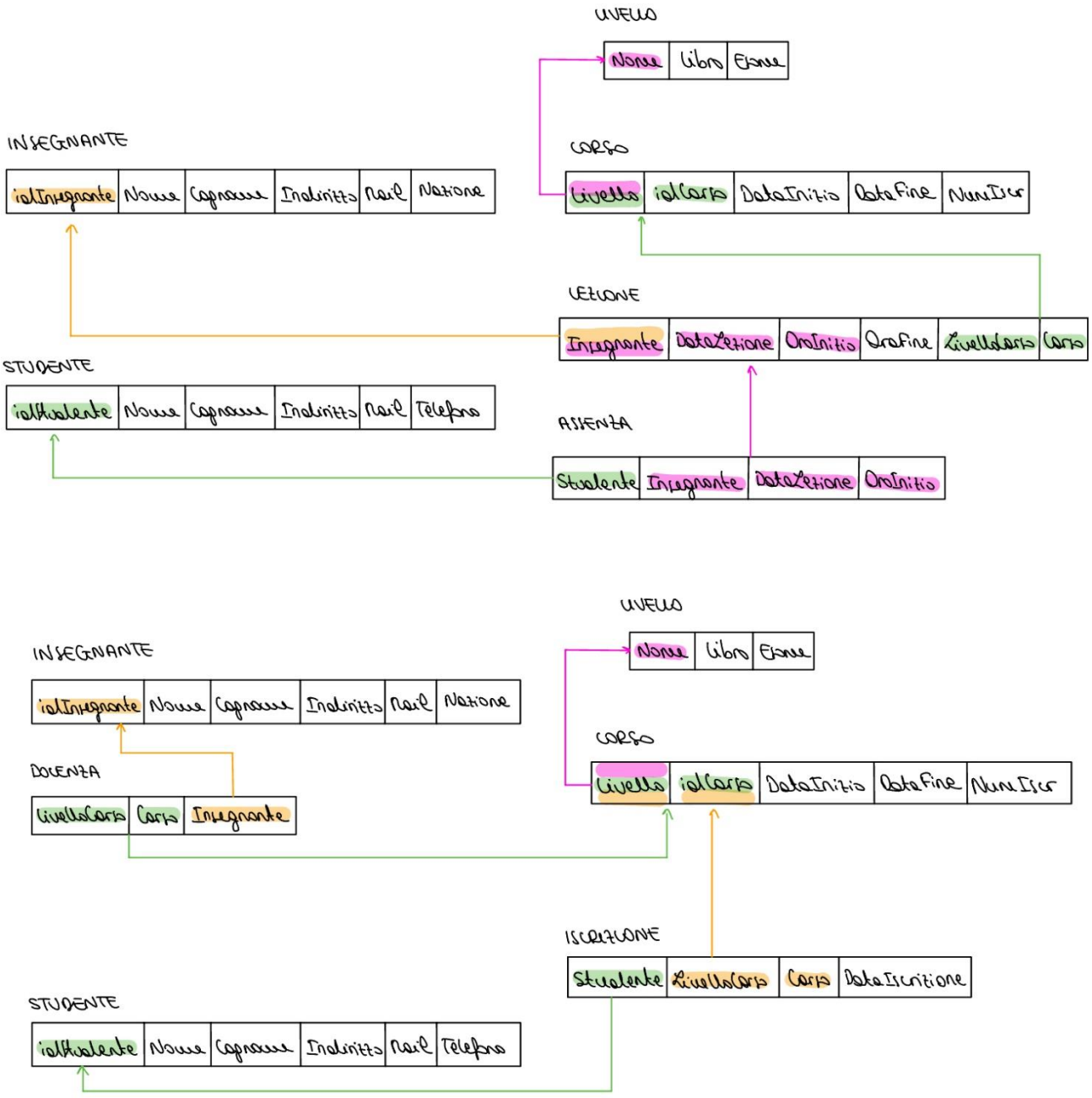
Iscrizione(LivelloCorso, Corso) ⊆ Corso(Livello, idCorso)

Assenza(Studente) ⊆ Studente(idStudente)

Assenza(Insegnante, DataLezione, OraInizioLezione) ⊆ Lezione(Insegnante, DataLezione, OraInizio)

Docenza(Insegnante) ⊆ Insegnante(idInsegnante)

Docenza(LivelloCorso, Corso) ⊆ Corso(Livello, idCorso)



Normalizzazione del modello relazionale

Il modello relazionale non è in forma normale: la presenza dell'attributo NumIscr in Corsi può generare anomalie di inserimento, cancellazione e aggiornamento. Tuttavia, dato il significativo risparmio prestazionale e la tipologia di operazioni previste, si decide di mantenere il modello dei dati in questa forma.

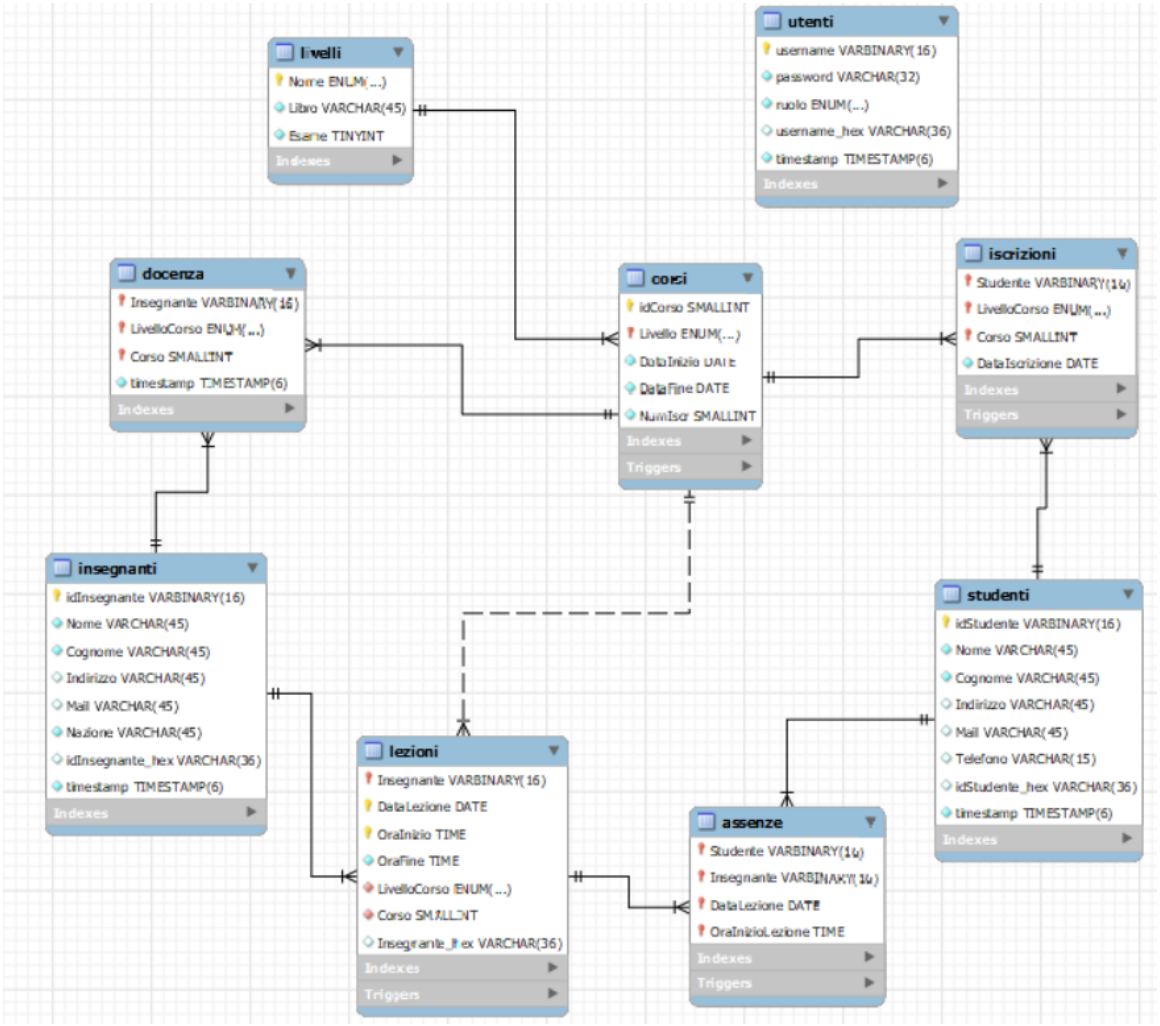
5. Progettazione fisica

Utenti e privilege

Si prevedono tre ruoli, per implementare il Principle of Least Privilege:

- Login:
  - Grant in esecuzione sull'operazione L1
- Segreteria:
  - Grant in esecuzione sulle operazioni R1, R2, AM1-AM20
- Insegnante:
  - Grant in esecuzione sull'operazione R3, R4

  Per identificare gli utenti si introduce una tabella Utenti per mantenerne le credenziali.



Strutture di memorizzazione

Tabella Utenti		
Colonna	**Tipo di dato**	**Attributi**[2]
username	Varbinary(16)	PK, NN
password	Varchar(32)	NN
ruolo	ENUM(‘segreteria’, ‘insegnante’)	NN
username_hex	Varchar(36)	G
timestamp	TIMESTAMP(6)	NN

Tabella Livelli		
Colonna	**Tipo di dato**	**Attributi**
nome	ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2')	PK, NN
libro	varchar(45)	NN
esame	Tinyint	NN

Tabella Corsi		
Colonna	**Tipo di dato**	**Attributi**
livello	ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2')	PK, NN
idCorso	smallint	PK, NN, UN
dataInizio	date	NN
dataFine	date	NN
numIscr	smallint	NN

Tabella Lezioni		
Colonna	**Tipo di dato**	**Attributi**
insegnante	Varbinary(16)	PK, NN
dataLezione	date	PK, NN
oraInizio	time	PK, NN
oraFine	time	NN
livelloCorso	ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2')	NN
Corso	smallint	NN, UN
Insegnante_hex	Varchar(36)	G

[2] PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

Tabella Insegnanti		
Colonna	**Tipo di dato**	**Attributi**
idInsegnante	Varbinary(16)	PK, NN
Nome	varchar(45)	NN
Cognome	varchar(45)	NN
Indirizzo	varchar(45)	
mail	varchar(45)	
nazione	varchar(45)	NN
idInsegnante_hex	Varchar(36)	G
timestamp	Timestamp(6)	NN

Tabella Studenti		
Colonna	**Tipo di dato**	**Attributi**
idStudente	Varbinary(16)	PK, NN
Nome	varchar(45)	NN
cognome	varchar(45)	NN
Indirizzo	varchar(45)	
mail	varchar(45)	
telefono	varchar(14)	
idInsegnante_hex	Varchar(36)	G
timestamp	Timestamp(6)	NN

Tabella Docenza		
Colonna	**Tipo di dato**	**Attributi**
insegnante	Varbinary(16)	PK, NN
livelloCorso	ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2')	PK, NN
Corso	smallint	PK, NN
timestamp	Timestamp(6)	NN

Tabella Iscrizioni		
Colonna	**Tipo di dato**	**Attributi**
studente	Varbinary(16)	PK, NN
livelloCorso	ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2')	PK, NN
Corso	smallint	PK, NN
DataIscrizione	date	NN

Tabella Assenze		
Colonna	**Tipo di dato**	**Attributi**
studente	Varbinary(16)	PK, NN
insegnante	Varbinary(16)	PK, NN
dataLezione	date	PK, NN
oraInizioLezione	time	PK, NN

Indici

Operazioni indice iscrizioni:

AM19 - Lista studenti corso

1 accesso in lettura Corso, 20 accessi in lettura Iscrizione, 20 accessi in lettura Studente

Costo totale: 41

Numero accessi: 44.280/anno

Operazioni indice lezioni:

AM13 - Lista lezioni corso

1 accesso in lettura Corso, 24 accessi in lettura Composizione, 24 accessi in lettura Lezione

Costo totale: 49

Numero accessi: 2.646/anno

R1 - Report mensile lezioni insegnante

16 accessi in lettura Lezione

Costo totale: 16

Numero accessi: 48/mese

R2 - Report mensile lezioni

51 accessi in lettura Lezione

Costo totale: 51

Numero accessi: 51/mese

R3 – Report lezioni settimana corrente

4 accessi in lettura Lezione

Costo totale: 4

Numero accessi: 48/ settimana

R4 - Report lezioni settimana prossima

4 accesso in lettura Lezione

Costo totale: 4

Numero accessi: 24/settimana

Tabella Iscrizioni	
Indice in_livello_corso	**Tipo:**
LivelloCorso, Corso	IDX

Tabella Lezioni	
Indice in_data_ora	**Tipo:**
Insegnante, DataLezione, OraInizio	IDX

Trigger

Tabella Assenze: BEFORE INSERT

Il trigger implementa la seguente regola aziendale: uno studente può risultare assente solo a lezioni appartenenti a corsi a cui è iscritto.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `assenze_BEFORE_INSERT` BEFORE INSERT ON `assenze` FOR EACH ROW BEGIN
	declare var_studente varbinary(16);
   declare var_livello ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2');
   declare var_corso smallint;

   select LivelloCorso, Corso
   from lezioni
   where Insegnante = new.Insegnante and DataLezione = new.DataLezione and OraInizio = new.OraInizioLezione
   into var_livello, var_corso;

	select studente
   from iscrizioni
   where LivelloCorso = var_livello and Corso = var_corso
   into var_studente;

   if(!var_studente) then signal sqlstate '45014' set message_text = 'Studente non iscritto al corso a cui appartiene la lezione o lezione non esistente.';
   end if;

END
```

Tabella Iscrizioni: BEFORE INSERT

Il trigger implementa la seguente regola aziendale: la data di iscrizione di uno studente a un corso deve precedere la data di fine dello stesso.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `iscrizioni_BEFORE_INSERT` BEFORE INSERT ON `iscrizioni` FOR EACH ROW BEGIN
	declare var_data_fine_corso date;

   select DataFine
   from corsi
   where Livello = new.LivelloCorso and idCorso = new.Corso
   into var_data_fine_corso;

   if var_data_fine_corso < new.DataIscrizione then
		signal sqlstate '45013' set message_text = 'Corso già finito.';
	end if;
END
```

Tabella Corsi: BEFORE UPDATE

Il trigger implementa la seguente regola aziendale: la data di inizio di un corso deve precedere la data di fine dello stesso.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `corsi_BEFORE_UPDATE` BEFORE UPDATE ON `corsi` FOR EACH ROW BEGIN
	if new.DataInizio >= new.DataFine then
		signal sqlstate '45012' set message_text = 'Date di inizio e fine del corso incongruenti.';
	end if;
END
```

Tabella Corsi: BEFORE INSERT

Il trigger implementa la seguente regola aziendale: la data di inizio di un corso deve precedere la data di fine dello stesso.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `corsi_BEFORE_INSERT` BEFORE INSERT ON `corsi` FOR EACH ROW BEGIN
	if new.DataInizio >= new.DataFine then
		signal sqlstate '45011' set message_text = 'Date di inizio e fine del corso incongruenti.';
	end if;
END
```

Tabella Lezioni: BEFORE INSERT

Il trigger implementa le seguenti regole aziendali:

La data in cui è tenuta una lezione deve appartenere al periodo in cui il relativo corso è tenuto.

L'ora di inizio di una lezione deve essere precedente all'ora di fine della stessa.

Gli orari delle lezioni tenute dallo stesso insegnante non possono sovrapporsi.

Gli orari delle lezioni appartenenti allo stesso corso non possono sovrapporsi.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `lezioni_BEFORE_INSERT` BEFORE INSERT ON `lezioni` FOR EACH ROW BEGIN
	declare var_dataInizio date;
   declare var_dataFine date;
   DECLARE done boolean default false;
   declare var_ora_inizio time;
   declare var_ora_fine time;

   DECLARE cur_ins CURSOR FOR
   SELECT OraInizio, OraFine
   FROM lezioni
   WHERE Insegnante = new.Insegnante and DataLezione = new.DataLezione;

   DECLARE cur_cor CURSOR FOR
   SELECT OraInizio, OraFine
   FROM lezioni
   WHERE LivelloCorso = new.LivelloCorso and Corso = new.Corso and DataLezione = new.DataLezione;
```

```
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = true;

select DataInizio, DataFine
from corsi
where idCorso = new.Corso and Livello = new.LivelloCorso
into var_dataInizio, var_dataFine;

if(new.DataLezione < var_dataInizio or new.DataLezione > var_dataFine) then SIGNAL SQLSTATE '45010' SET MESSAGE_TEXT = 'Data lezione non appartiene al periodo in cui si tiene il corso.';
end if;

    IF new.oraInizio >= new.oraFine then signal sqlstate '45001' set message_text = 'Orari lezione incongruenti.';
END IF;

OPEN cur_ins;

loop_ins: LOOP

    FETCH cur_ins INTO var_ora_inizio, var_ora_fine;

    IF done = true THEN
      LEAVE loop_ins;
    END IF;

    IF (var_ora_inizio <= new.OraInizio and var_ora_fine >= new.OraInizio) THEN signal sqlstate '45002' set message_text = 'Orario lezione in conflitto con lezioni precedenti insegnante.';
    ELSEIF (var_ora_inizio >= new.OraInizio and var_ora_inizio <= new.OraFine) THEN signal sqlstate '45003' set message_text = 'Orario lezione in conflitto con lezioni successive insegnante.';
    ELSEIF (var_ora_inizio <= new.OraInizio and var_ora_fine >= new.OraFine) THEN signal sqlstate '45004' set message_text = 'Orario lezione in sovrapposizione a lezione precedente insegnante.';
```

ELSEIF (var_ora_inizio >= new.OraInizio and var_ora_fine <= new.OraFine) THEN signal sqlstate '45005' set message_text = 'Orario lezione in sovrapposizione a lezione precedente insegnante.';
END IF;

END LOOP loop_ins;

CLOSE cur_ins;

set done = false;
OPEN cur_cor;

loop_cor: LOOP

FETCH cur_cor INTO var_ora_inizio, var_ora_fine;

IF done = true THEN
LEAVE loop_cor;
END IF;

IF (var_ora_inizio <= new.OraInizio and var_ora_fine >= new.OraInizio) THEN signal sqlstate '45006' set message_text = 'Orario lezione in conflitto con lezioni precedenti corso.';
ELSEIF (var_ora_inizio >= new.OraInizio and var_ora_inizio <= new.OraFine) THEN signal sqlstate '45007' set message_text = 'Orario lezione in conflitto con lezioni successive corso.';
ELSEIF (var_ora_inizio <= new.OraInizio and var_ora_fine >= new.OraFine) THEN signal sqlstate '45008' set message_text = 'Orario lezione in sovrapposizione a lezione precedente corso.';
ELSEIF (var_ora_inizio >= new.OraInizio and var_ora_fine <= new.OraFine) THEN signal sqlstate '45009' set message_text = 'Orario lezione in sovrapposizione a lezione precedente corso.';
END IF;

END LOOP loop_cor;

END

Eventi

I seguenti eventi vengono instanziati in fase di configurazione del sistema:

```
CREATE EVENT IF NOT EXISTS `cancella_entries`
ON SCHEDULE EVERY 4 MONTH
STARTS CURRENT_DATE + INTERVAL 1 DAY
ON COMPLETION PRESERVE
DO CALL elimina_vecchie_entries();

CREATE EVENT IF NOT EXIST aggiorna_cred_segreteria
ON SCHEDULE EVERY 5 YEAR
DO
  UPDATE utenti
  SET timestamp = current_timestamp()
  WHERE ruolo = 'segreteria';
```

Viste

Non sono state usate viste.

Stored Procedures e transazioni

OPERAZIONE AM15

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiorna_data_fine_corso`(in var_livelloCorso ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2'), in var_corso smallint, in var_nuova_data date)
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
```

```
    resignal;
  end;

      set transaction isolation level repeatable read;
  start transaction;

            UPDATE corsi set DataFine = var_nuova_data where Livello = var_livelloCorso and
idCorso = var_corso;

  commit;
END
```

OPERAZIONE AM1

Si utilizza il livello di isolamento serializable per evitare aggiornamenti fantasma durante la creazione dell'id.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_corso`(in var_livello
ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2'), in var_dataInizio date, in var_dataFine date, out var_idCorso
smallint)
BEGIN
      declare var_id smallint;

      declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

      set transaction isolation level serializable;
  start transaction;

            select max(idCorso)
    from corsi
    where Livello = var_livello
    into var_id;
```

```
    set var_idCorso = var_id;

    if var_idCorso is null then set var_idCorso = 1;
            else set var_idCorso = var_idCorso + 1;
    end if;

            insert into corsi(idCorso, Livello, DataInizio, DataFine) values (var_idCorso, var_livello, var_dataInizio, var_dataFine);

  commit;

END
```

OPERAZIONE AM3

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_insegnante`(in var_nome varchar(45), in var_cognome varchar(45), in var_indirizzo varchar(45), in var_mail varchar(45), in var_nazione varchar(45), out var_idInsegnante varchar(36))
BEGIN
      declare var_idBin varbinary(16);

      declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

  set var_idInsegnante = uuid();
  set var_idBin = uuid_to_bin(var_idInsegnante, 1);

  set transaction isolation level repeatable read;
  start transaction;
```

insert into insegnanti(idInsegnante, Nome, Cognome, Indirizzo, Mail, Nazione) values (var_idBin, var_nome, var_cognome, var_indirizzo, var_mail, var_nazione);

insert into utenti(username, password, ruolo) values (var_idBin, md5(var_idInsegnante), 'insegnante');

commit;

END

OPERAZIONE AM2

Si utilizza il livello di isolamento serializable per evitare aggiornamenti fantasma che potrebbero causare sovrapposizioni tra lezioni dello stesso insegnante/corso.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_lezione`(in var_insegnante varchar(36), in var_dataLezione date, in var_oraInizio time, in var_oraFine time, in var_livelloCorso ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2'), in var_corso smallint)
BEGIN
    declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

    set transaction isolation level serializable;
  start transaction;

    insert into lezioni(Insegnante, DataLezione, OraInizio, OraFine, LivelloCorso, Corso) values (uuid_to_bin(var_insegnante,1), var_dataLezione, var_oraInizio, var_oraFine, var_livelloCorso, var_corso);

    commit;
END
```

OPERAZIONE AM4

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `aggiungi_studente`(in var_nome varchar(45), in var_cognome varchar(45), in var_indirizzo varchar(45), in var_mail varchar(45), in var_telefono varchar(15), out var_idStudente varchar(36))
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

   set var_idStudente = uuid();

        set transaction isolation level repeatable read;
   start transaction;

                insert into studenti(idStudente, Nome, Cognome, Indirizzo, Mail, Telefono) values (uuid_to_bin(var_idStudente, 1), var_nome, var_cognome, var_indirizzo, var_mail, var_telefono);
        commit;
END
```

OPERAZIONE AM5

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `assegna_corso`(in var_insegnante varchar(36), in var_livelloCorso ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2'), in var_corso smallint)
BEGIN
                declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

        set transaction isolation level repeatable read;
   start transaction;
```

```
        insert into docenza(Insegnante, LivelloCorso, Corso) values (uuid_to_bin(var_insegnante,1), var_livelloCorso, var_corso);
        update insegnanti set timestamp = current_timestamp() where idInsegnante = uuid_to_bin(var_insegnante,1);

    commit;
END
```

OPERAZIONE E1

Si suddivide l'eliminazione delle vecchie entries in transazioni diverse per entità figlie/genitori, mantenendo così un buon livello di coerenza ed efficienza.

Inoltre, si utilizza il livello di isolamento read committed, in quanto la stored procedure lavora unicamente a mezzanotte, orario in cui si prevede al più l'interazione con la base di dati da parte di insegnanti.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `elimina_vecchie_entries`()
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    SET SQL_SAFE_UPDATES = 0;
    SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

    START TRANSACTION;
        DELETE FROM assenze WHERE DataLezione < DATE_SUB(CURDATE(), INTERVAL 10 YEAR);
        DELETE FROM iscrizioni WHERE DataIscrizione < DATE_SUB(CURDATE(), INTERVAL 10 YEAR);
        DELETE FROM docenza WHERE timestamp < NOW() - INTERVAL 10 YEAR;
    COMMIT;

    START TRANSACTION;
```

```
        DELETE FROM lezioni WHERE DataLezione < DATE_SUB(CURDATE(), INTERVAL 10 YEAR);
        DELETE FROM corsi WHERE DataFine < DATE_SUB(CURDATE(), INTERVAL 10 YEAR);
    COMMIT;

    START TRANSACTION;
        DELETE FROM insegnanti WHERE timestamp < NOW() - INTERVAL 10 YEAR;
        DELETE FROM studenti WHERE timestamp < NOW() - INTERVAL 10 YEAR;
        DELETE FROM utenti WHERE timestamp < NOW() - INTERVAL 10 YEAR;
    COMMIT;

  SET SQL_SAFE_UPDATES = 1;
END
```

OPERAZIONE AM18

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista studenti`()
BEGIN
    declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

  set transaction isolation level repeatable read;
  set transaction read only;
  start transaction;

        select idStudente_hex, Nome, Cognome, Indirizzo, Mail, Telefono
        from studenti
    where timestamp >= DATE_SUB(CURDATE(), INTERVAL 5 YEAR);

    commit;
END
```

OPERAZIONE AM11

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista_assenze_studente`(in var_idStudente varchar(36), in var_livello ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2'), in var_corso smallint)
BEGIN
	declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

  set transaction isolation level repeatable read;
  set transaction read only;
  start transaction;

		select assenze.DataLezione, OraInizioLezione
		from assenze join lezioni on assenze.Insegnante=lezioni.Insegnante and assenze.DataLezione=lezioni.DataLezione and OraInizioLezione=OraInizio
		where Studente = uuid_to_bin(var_idStudente, 1) and LivelloCorso = var_livello and Corso = var_corso
	order by DataLezione, OraInizioLezione;

  commit;
END
```

OPERAZIONE AM16

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista_corsi`()
BEGIN
	declare exit handler for sqlexception
  begin
    rollback;
    resignal;
```

```
    end;

    set transaction isolation level repeatable read;
    set transaction read only;
    start transaction;

                select Livello, idCorso, DataInizio, DataFine, NumIscr
                from corsi
                where DataFine >= DATE_SUB(CURDATE(), INTERVAL 3 YEAR)
        order by DataInizio;

    commit;
END
```

OPERAZIONE AM14

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista_corsi_insegnante`(in
var_idInsegnante varchar(36))
BEGIN
        declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level repeatable read;
    set transaction read only;
    start transaction;

                select LivelloCorso, Corso, DataInizio, DataFine, NumIscr
                from docenza join corsi on LivelloCorso=Livello and Corso=idCorso
                where Insegnante = uuid_to_bin(var_idInsegnante, 1) ;

    commit;
END
```

OPERAZIONE AM17

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista_insegnanti`()
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

   set transaction isolation level repeatable read;
   set transaction read only;
   start transaction;

                select idInsegnante_hex, Nome, Cognome, Indirizzo, Mail, Nazione
                from insegnanti
      where timestamp >= DATE_SUB(CURDATE(), INTERVAL 5 YEAR);

        commit;
END
```

OPERAZIONE AM12

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista_iscrizioni_studente`(in var_studente varchar(36))
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

   set transaction isolation level repeatable read;
   set transaction read only;
   start transaction;
```

```
            select LivelloCorso, Corso, DataIscrizione
            from iscrizioni
            where Studente = uuid_to_bin(var_studente,1);

  commit;
END
```

OPERAZIONE AM13

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista_lezioni_corso`(in var_livelloCorso
ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2'), in var_corso smallint)
BEGIN
      declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

  set transaction isolation level repeatable read;
  set transaction read only;
  start transaction;

            select bin_to_uuid(Insegnante, 1), DataLezione, OraInizio, OraFine
            from lezioni
            where LivelloCorso = var_livelloCorso and Corso = var_corso;

  commit;
END
```

OPERAZIONI AM20

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista_livelli`()
BEGIN
      declare exit handler for sqlexception
  begin
```

```
    rollback;
    resignal;
  end;

  set transaction isolation level repeatable read;
  set transaction read only;
  start transaction;

            select *
            from livelli
    order by Nome;

  commit;
END
```

OPERAZIONE AM18

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista_studenti`()
BEGIN
      declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

  set transaction isolation level repeatable read;
  set transaction read only;
  start transaction;

            select idStudente_hex, Nome, Cognome, Indirizzo, Mail, Telefono
            from studenti
    where timestamp >= DATE_SUB(CURDATE(), INTERVAL 5 YEAR);

      commit;
END
```

OPERAZIONE AM19

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `lista_studenti_corso`(in var_livello ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2'), in var_corso smallint)
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

   set transaction isolation level repeatable read;
   set transaction read only;
   start transaction;

                select idStudente_hex, Nome, Cognome, Indirizzo, Mail, Telefono
      from iscrizioni join studenti on Studente = idStudente
      where LivelloCorso = var_livello and Corso = var_corso
      order by Cognome, Nome;

   commit;
END
```

OPERAZIONE L1

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `login`(in var_username varchar(36), in var_pass varchar(45), out var_role INT)
BEGIN
        declare var_user_role ENUM('segreteria', 'insegnante');

   select ruolo from Utenti
                where username = uuid_to_bin(var_username,1)
      and password = md5(var_pass)
      into var_user_role;
```

```
        -- See the corresponding enum in the client
                if var_user_role = 'segreteria' then
                        set var_role = 1;
                elseif var_user_role = 'insegnante' then
                        set var_role = 2;
                else
                        set var_role = 3;
                end if;
END
```

OPERAZIONE AM8

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ottieni_info_corso`(
   IN var_livelloCorso ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2'),
   IN var_corso smallint
)
BEGIN
        DECLARE var_iscr INT;

   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;
      RESIGNAL;
   END;

   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
   START TRANSACTION;

   SELECT DataInizio, DataFine, Libro, Esame, NumIscr, insegnanti.Nome, Cognome,
idInsegnante_hex
   FROM corsi
   JOIN docenza ON Livello = LivelloCorso AND idCorso = Corso
   JOIN insegnanti ON Insegnante = idInsegnante
   JOIN livelli on Livello = livelli.Nome
   WHERE LivelloCorso = var_livelloCorso AND Corso = var_corso;
```

```
   COMMIT;
END
```

OPERAZIONE AM9

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ottieni_info_insegnante`(in var_idInsegnante varchar(36))
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

   set transaction isolation level repeatable read;
   set transaction read only;
   start transaction;

                select Nome, Cognome, Indirizzo, Mail, Nazione
                from insegnanti
                where idInsegnante = uuid_to_bin(var_idInsegnante, 1) ;

   commit;
END
```

OPERAZIONE AM10

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `ottieni_info_studente`(in var_idStudente varchar(36))
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
```

```
    end;

    set transaction isolation level repeatable read;
    set transaction read only;
    start transaction;

                select Nome, Cognome, Indirizzo, Mail, Telefono
                from studenti
                where idStudente = uuid_to_bin(var_idStudente, 1) ;

    commit;
END
```

OPERAZIONE AM7

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_assenza`(in var_studente varchar(36), in var_insegnante varchar(36), in var_dataLezione date, in var_oraInizioLezione time)
BEGIN
        declare exit handler for sqlexception
    begin
        rollback;
        resignal;
    end;

    set transaction isolation level repeatable read;
    start transaction;

                insert into assenze(Studente, Insegnante, DataLezione, OraInizioLezione) values (uuid_to_bin(var_studente,1), uuid_to_bin(var_insegnante,1), var_dataLezione, var_oraInizioLezione);

        commit;
END
```

OPERAZIONE AM6

Si utilizza come livello di isolamento serializable per evitare aggiornamenti fantasma durante la registrazione di un'assenza, che potrebbero causare problemi nell'aggiornamento del numero di iscritti.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_iscrizione`(in var_studente varchar(36), in var_livelloCorso ENUM('A1', 'A2', 'B1', 'B2', 'C1', 'C2'), in var_corso smallint, in var_dataIscrizione date)
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

   set transaction isolation level serializable;
   start transaction;

            insert into iscrizioni(Studente, LivelloCorso, Corso, DataIscrizione) values (uuid_to_bin(var_studente,1), var_livelloCorso, var_corso, var_dataIscrizione);
            update studenti set timestamp = current_timestamp() where idStudente = uuid_to_bin(var_studente,1);
      update corsi set NumIscr = NumIScr + 1 where Livello = var_livelloCorso and idCorso = var_corso;

        commit;
END
```

Per i report si utilizza come livello di isolamento serializable, per evitare aggiornamenti fantasma durante il reperimento delle informazioni dalla base di dati.

OPERAZIONE R2

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `report_mensile_lezioni`(in var_date date)
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
```

```
    resignal;
  end;

  set transaction isolation level serializable;
  set transaction read only;
  start transaction;

            select Insegnante_hex, DataLezione, OraInizio, OraFine, LivelloCorso, Corso
            from lezioni
            where
                  DataLezione >= DATE_FORMAT(var_date, '%Y-%m-01')
                  and DataLezione < DATE_ADD(DATE_FORMAT(var_date, '%Y-%m-01'),
INTERVAL 1 MONTH)
            order by Insegnante_hex, DataLezione, OraInizio;

      commit;
END
```

OPERAZIONE R1

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `report_mensile_lezioni_insegnante`(in
var_insegnante varchar(36), in var_date date)
BEGIN
      declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

  set transaction isolation level serializable;
  set transaction read only;
  start transaction;

            select DataLezione, OraInizio, OraFine, LivelloCorso, Corso
            from lezioni
```

```
        where
            Insegnante = uuid_to_bin(var_insegnante,1)
            and DataLezione >= DATE_FORMAT(var_date, '%Y-%m-01')
            and DataLezione < DATE_ADD(DATE_FORMAT(var_date, '%Y-%m-01'),
INTERVAL 1 MONTH)
        order by DataLezione, OraInizio;

    commit;
END
```

OPERAZIONE R3

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `report_settimana_corrente_lezioni`(in
var_insegnante varchar(36))
BEGIN
    declare exit handler for sqlexception
  begin
    rollback;
    resignal;
  end;

  set transaction isolation level serializable;
  set transaction read only;
  start transaction;

        select DataLezione, OraInizio, OraFine, LivelloCorso, Corso
        from lezioni
        where
            Insegnante = uuid_to_bin(var_insegnante,1)
            and DataLezione >= DATE_SUB(CURDATE(), INTERVAL
WEEKDAY(CURDATE()) DAY)
            and DataLezione < DATE_ADD(DATE_SUB(CURDATE(), INTERVAL
WEEKDAY(CURDATE()) DAY), INTERVAL 7 DAY)
        order by DataLezione, OraInizio;
```

```
        commit;
END
```

OPERAZIONE R4

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `report_settimana_prossima_lezioni`(in var_insegnante varchar(36))
BEGIN
        declare exit handler for sqlexception
   begin
      rollback;
      resignal;
   end;

   set transaction isolation level serializable;
   set transaction read only;
   start transaction;

                select DataLezione, OraInizio, OraFine, LivelloCorso, Corso
                from lezioni
                where
                        Insegnante = uuid_to_bin(var_insegnante,1)
                        and DataLezione >= CURDATE() + INTERVAL (7 - WEEKDAY(CURDATE())) DAY
                        and DataLezione < CURDATE() + INTERVAL (14 - WEEKDAY(CURDATE())) DAY
                order by DataLezione, OraInizio;

        commit;
END
```