Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

TITOLO DEL PROGETTO

0241098

Stefano Costanzo

Indice

1. Descrizione del Minimondo

Sistema di gestione di corsi di lingue straniere.

Si progetti un sistema informativo per la gestione dei corsi di lingua inglese, tenuti presso un instituto di insegnamento. Tutte le informazioni fanno riferimento ad un solo anno scolastico in corso, e non viene richiesto di mantenere le informazioni relative agli anni scolastici precedenti (è quindi necessario prevedere un'opportuna funzionalità per indicare che si vuole riconfigurare il sistema per l'avvio di un nuovo anno scolastico). La base dati deve avere le seguenti caratteristiche e mantenere le seguenti informazioni.

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). Gli allievi possono anche prenotare lezioni private, qualora vogliano approfondire alcuni aspetti della lingua inglese. Si vuole tener traccia di tutte le lezioni private eventualmente richieste da un allievo, in quale data e con quale insegnante. La prenotazione di una lezione individuale non può avvenire in concomitanza di un altro impegno di un insegnante.

La scuola organizza anche un insieme di attività culturali. Ciascuna attività è identificata da un codice progressivo, e sono noti il giorno e l'ora in cui verrà tenuta. Nel caso di proiezioni in lingua originale, sono noti il nome del film ed il nome del regista. Nel caso di conferenze, sono noti l'argomento che verrà trattato ed il nome del conferenziere. Per poter partecipare alle attività gli allievi devono iscriversi.

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

Specifica disambiguata

Il testo non presenta frasi o parole ambigue.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Corso	Corsi di lingue inglese		Livello, Insegnante, Allievo
Livello	Livello di competenza linguistica		Corso
Insegnante	Insegnante dell'istituto. Puo essere assegnato a corsi e lezioni private		Corso, Lezione Privata
Allievo	Allievo iscritto nell'istituto		Corso, Lezione Privata, Attività
Lezione Privata	Lezione individuale tenuta dall'allievo con l'insegnante		Allievo, Insegnante
Attività	Attività culturali tenute nell'istituto. Possono essere conferenze o proiezioni	Conferenza, proiezione	Allievo
Personale amministrativo	Personale lavorativo del sistema. Si interessa della gestione dei corsi e degli insegnanti	Amministratore	
Segreteria	Personale lavorativo del sistema. Si interessa della gestione degli allievi		

Raggruppamento dei requisiti in insiemi omogenei

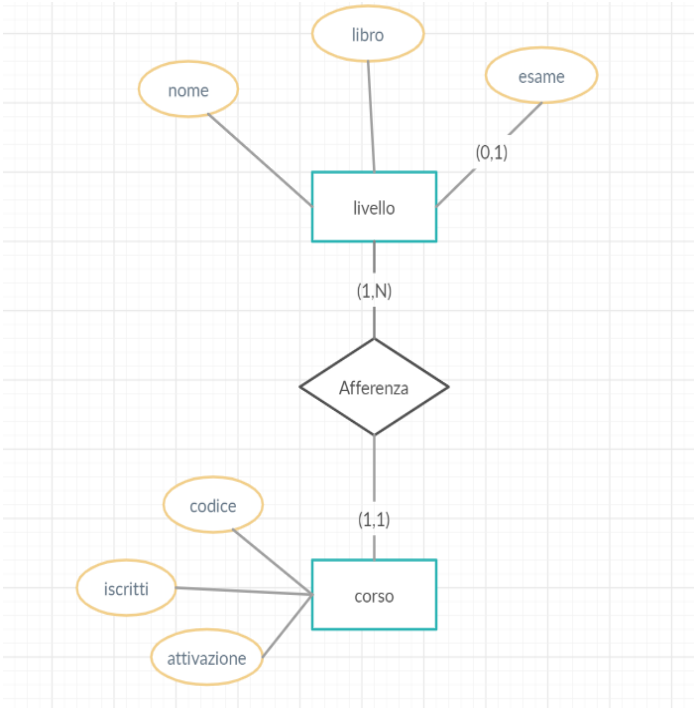
Frasi relative a Livello
Ciascun livello è identificato dal nome del livello stesso (ad esempio Elementary, Intermediate, First Certificate, Advanced, Proficiency); inoltre è specificato il nome del libro di testo e se viene richiesto di sostenere un esame finale (righe 7-9).
Frasi relative a Corso
I corsi sono organizzati per livelli. (riga 7) 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. (righe 10-13)
Frasi relative ad 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. (righe 14-16)
Frasi relative ad Allievo
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). Gli allievi possono anche prenotare lezioni private. (righe 17-19)
Frasi relative a Lezione privata
Si vuole tener traccia di tutte le lezioni private eventualmente richieste da un allievo, in quale data e con quale insegnante. La prenotazione di una lezione individuale non può avvenire in concomitanza di un altro impegno di un insegnante.
Frasi relative ad Attività
La scuola organizza anche un insieme di attività culturali. Ciascuna attività è identificata da un codice progressivo, e sono noti il giorno e l'ora in cui verrà tenuta. Nel caso di proiezioni in lingua originale, sono noti il nome del film ed il nome del regista. Nel caso di conferenze, sono noti l'argomento che verrà trattato ed il nome del conferenziere. Per poter partecipare alle attività gli allievi devono iscriversi. (righe 24-28)
Frasi relative a Personale Lavorativo

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. (righe 29-33)

3. Progettazione concettuale

Costruzione dello schema E-R

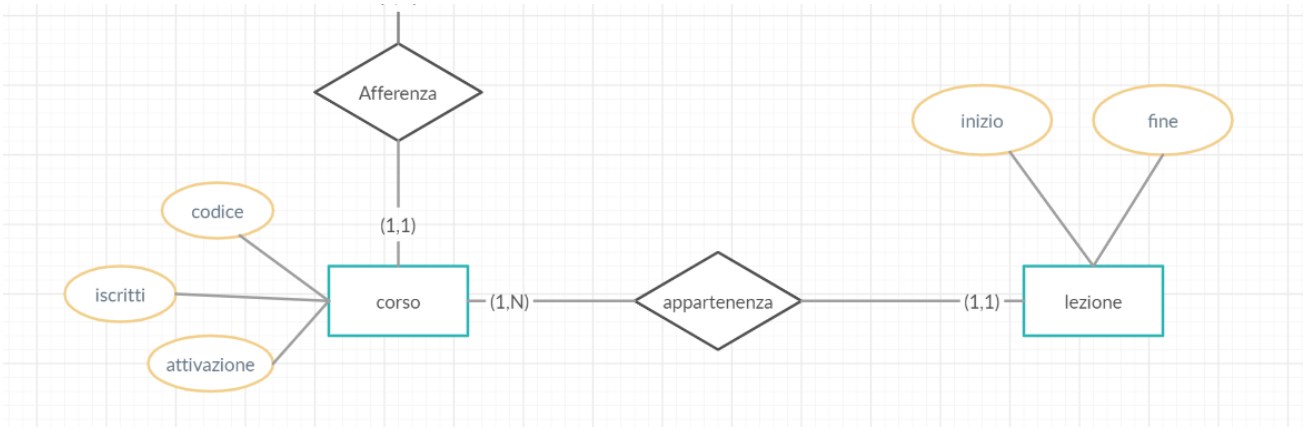
Per la costruzione dello schema E-R è stata seguita prevalentemente una strategia a macchia d'olio.



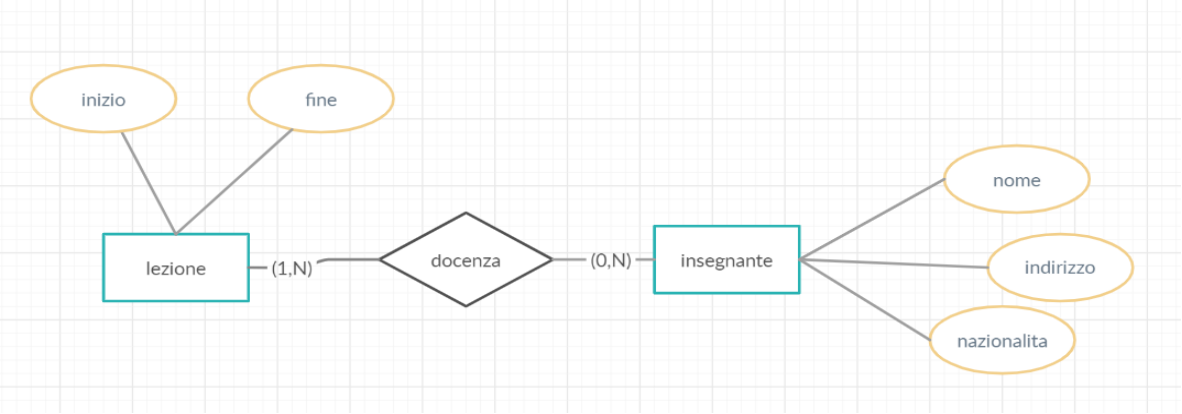
Si è partito riconoscendo l'entità *livello* con attributi *nome*, *libro*, *esame* e l'entità *corso* con attributi *attivazione*, *iscritti*, *codice* collegati fra loro dalla relazione "uno a molti" *afferenza* (FIG 1).

Per quanto riguarda le informazioni anagrafiche (riga 12) degli iscritti si è scelto di posticipare il loro inserimento all'interno dell'entità "allievo".

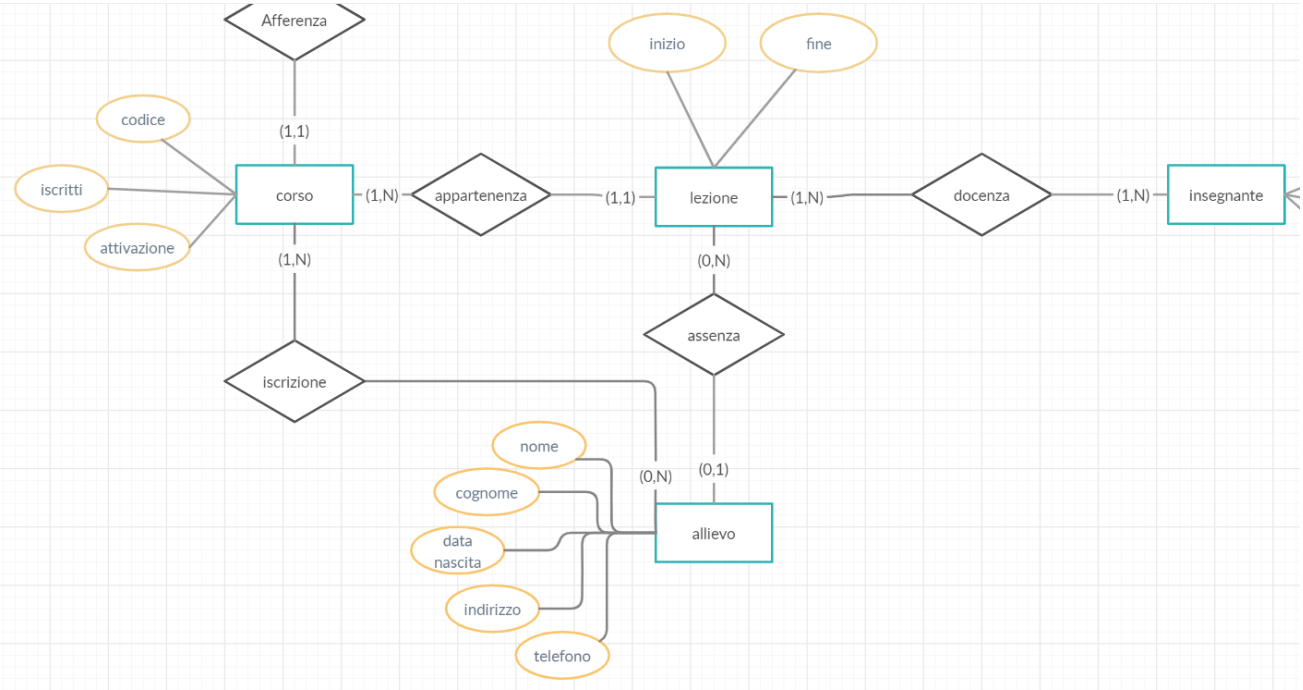
Per registrare giorni e orari in cui vengono impartiti i corsi (riga 13) è stata aggiunta l'entità *lezione* come "parte-di" *corso* con relazione "uno a molti". Invece di data e orario si è scelto *inizio* e *fine* per registrare anche la durata della lezione (FIG 2)



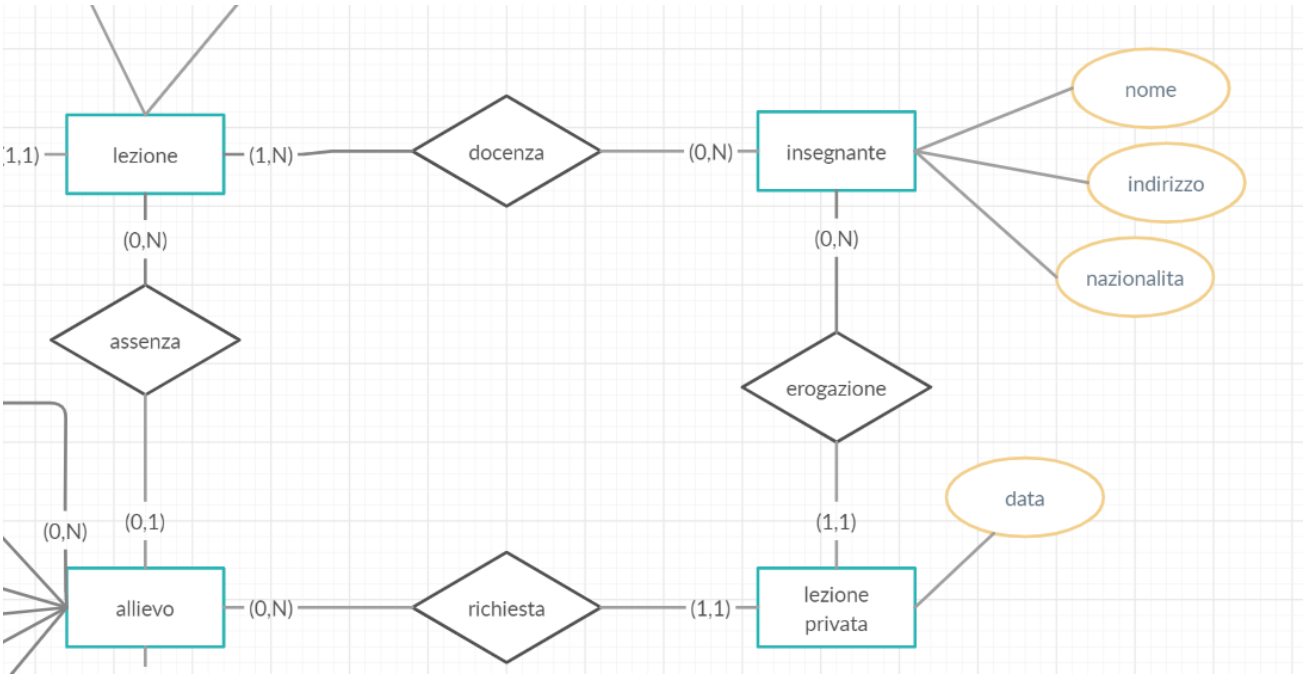
L'entità *insegnante* con attributi *nome*, *indirizzo* e *nazionalità* viene collegata con relazione "molti a molti" con *lezione.* (FIG 3)



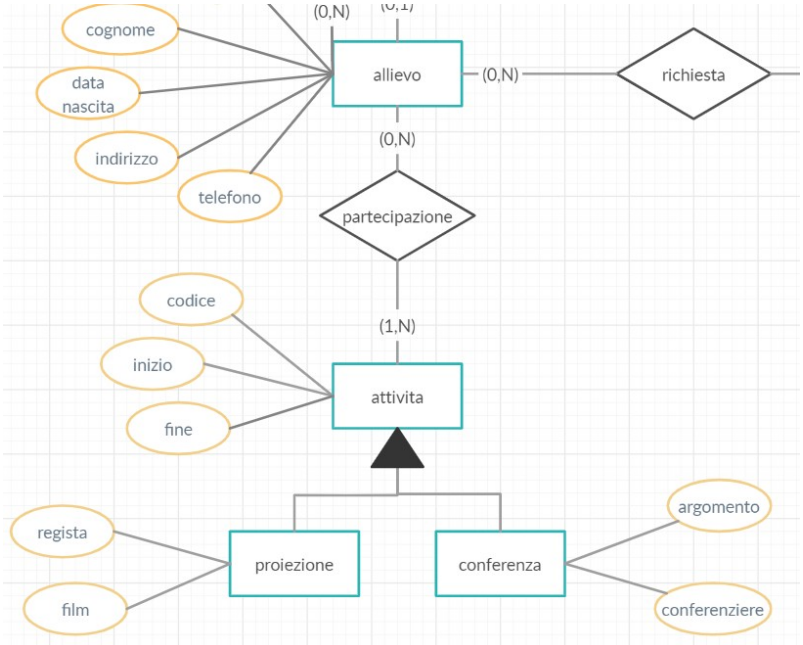
L'entità *allievo* con attributi *nome*, *cognome*, *data nascita*, *indirizzo* (ovvero informazioni anagrafiche richieste in precedenza, riga 13), e *telefono* è collegata a *corso* dalla relazione *iscrizione* e a *lezione* dalla relazione *assenza* per registrare una per una le lezioni a cui l'allievo non è stato presente (riga 17). (FIG 4)



L'entità *lezione privata* con attributo *data* viene collegata con relazioni uno a molti ad *allievo* ed *insegnante* tramite rispettivamente *richiesta* ed *erogazione*. (FIG 5)



L' entità *proiezione* con attributi *regista*, *film* e l'entità *conferenza* con attributi *argomento* e *conferenziere* si generalizzano totalmente nell'entità *attività* con attributi *codice*, *inizio*, *fine* legato ad *allievo* dalla relazione *partecipazione*. *(FIG 5)*



Si è scelta la generalizzazione per accorpare gli attributi comuni alle due entità *proiezione* e *conferenza*.

Integrazione finale

Regole aziendali

1) L'insegnante non deve erogare lezioni private o lezioni regolari – inerenti i corsi – nelle stesse fasce orarie.
2) L'allievo non deve essere assente alle lezioni dei corsi a cui non è iscritto.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Livello	Livello di conoscenze linguistiche del corso	Nome, Libro, Esame	Nome
Corso	Corso in inglese tenuto nell'istituto	Codice, Attivazione, iscritti	
Lezione	Lezione del corso	Inizio, Fine	
Insegnante	Docente dell'istituto. Eroga lezioni regolari e lezioni private	Nome, Indirizzo, Nazionalità	
Allievo	Studente dell'istituto	Nome, Cognome, DataNascita, Indirizzo, Telefono	
LezionePrivata	Lezione privata con insegnante richiesta dall'allievo	Data	
Attività	Attività tenuta nell'istituto	Codice, Inizio, Fine	Codice
Proiezione	E' una tipologia di attività. Proiezione filmografica	Regista, Film, (Codice, Inizio, Fine)	
Conferenza	E' una tipologia di attività. Conferenza pubblica di carattere culturale	Argomento, Conferenziere, (Codice, Inizio, Fine)	

Relazione	Descrizione	Entità Coinvolte	Attributi
Afferenza	Associa un corso al proprio livello linguistico	Corso (1,1), Livello (1,N)	
Appartenenza	Associa un corso alle proprie lezioni	Corso (1,N), Lezione (1,1)	
Iscrizione	Iscrizione dell'allievo ad un corso	Corso (1, N), Allievo (0, N)	
Assenza	Registra l'assenza di uno studente alla lezione	Lezione (0,N), Allievo (0,1)	
Docenza	Associa un insegnante con la lezione che deve erogare	Lezione (1, 1), Insegnante (0,N)	
Erogazione	Associa un insegnante con la lezione privata che deve erogare	Insegnante (0,N), LezionePrivata (1,1)	

Richiesta	Associa un allievo alla lezione privata richiesta	Allievo (0,N), LezionePrivata (1,1)	
Partecipazione	Associa un allievo all'attività da seguire	Allievo (0,N), Attivita (1,N)	

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo	Volume atteso
Livello	E	5
Corso	E	20
Lezione	E	350
Allievo	E	500
Insegnante	E	30
LezionePrivata	E	700
Attivita	E	50
Proiezione	E	40
Conferenza	E	10
Afferenza	R	20
Appartenenza	R	350
Assenza	R	1400
Iscrizione	R	800
Partecipazione	R	2000
Erogazione	R	700
Richiesta	R	700
Docenza	R	350

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
1	Iscrivi allievo indicando tutti i suoi dati	5 volte al giorno
2	Iscrivi insegnante indicando tutti i suoi dati	2 volte a settimana
3	Inserisci nuovo corso con il livello appropriato	1 volta a settimana
4	Inserisci nuova lezione indicandone il corso di riferimento	2 volte al giorno

5	Inserisci lezione private indicando allievo, insegnante e data.	10 volte al giorno
6	Inserisci attività indicando tutti i suoi dati	3 volte a settimana
7	Registra assenza dell'allievo alla lezione	20 volte al giorno
8	Registra iscrizione dell'allievo al corso	6 volte al giorno
9	Registra partecipazione dell'allievo all'attività	25 volte al giorno
10	Stampa agenda, su base settimanale, delle attività svolte dall'insegnante	5 volte al giorno
11	Stampa agenda, su base mensile, delle attività svolte da tutti gli insegnanti	1 volta a settimana
12	Stampa informazioni del corso e le lezioni a esso connesse	2 volte al giorno
13	Reset di tutte le informazioni per l'anno accademico passato del database	1 volte l'anno

Costo delle operazioni

1) Necessità solo di una scrittura in *Allievo* ripetuta 5 volte al giorno.
   Ha costo 10 [2 * 5].
2) Necessità solo di una scrittura in *Insegnante* ripetuta 2 volte a settimana.
   Ha costo 4/7 [2 * 2 /7].
3) Necessità di una scrittura in *Corso* e una in *Afferenza* ripetute 1 volta a settimana.
   Ha costo 4/7 [(2+2) * 1/7].
4) Necessità di una scrittura in *Lezione* e una in *Appartenenza* ripetute 2 volte al giorno.
   Ha costo 8 [(2+2) * 2].
5) Necessità di 3 scritture in *Richiesta*, *LezionePrivata* e *Erogazione*, il tutto ripetuto 10 volte al giorno.
   Ha costo 60 [6 * 10].
6) Necessità di due scritture in *Proiezione*/*Conferenza* e in *Attività* ripetute 3 volte a settimana.
   Ha valore 12/7 [4 * 3/7].
7) Necessità di una scrittura in *Assenza* ripetuta 20 volte al giorno.
   Ha valore 40 [2 * 20].
8) Necessità di una scrittura in *Iscrizione* e una in *Corso* ripetute 6 volte al giorno.

Ha valore 24 [(2 + 2) * 6].

9) Necessità di una scrittura in *Partecipazione* ripetuta 25 volte al giorno.
   Ha valore 50 [2 * 25].
10) Necessità di due letture in *Docenza* e *Lezione* per trovare tutte le informazioni riguardanti le lezioni regolari e altre due in *Erogazione* e *LezionePrivata* per le informazioni sulle Lezioni Private. Il tutto ripetuto 5 volte al giorno.
    Ha valore 20 [(2 + 2) * 5].
11) Necessità una lettura in *Insegnante* e per ogni risultato (30 in media) di due letture in *Docenza* e *Lezione* per trovare tutte le informazioni riguardanti le lezioni regolari e altre due in *Erogazione* e *LezionePrivata* per le informazioni sulle Lezioni Private. Il tutto ripetuto 1 volta a settimana.
    Ha valore 120/7 [30 * (2 + 2) * 1/7].
12) Necessità due letture in *Corso* e in *Afferenza* per le informazioni di carattere generale sul corso e due in *Appartenenza* e *Lezione* per rintracciare tutte le lezioni a esso connesse. Il tutto ripetuto 2 volte al giorno.
    Ha valore 8 [(2 + 2) *2].
13) Necessità di diverse scritture ma il suo costo è molto basso essendo utilizzata una volta l'anno.

Ristrutturazione dello schema E-R

1) Analisi delle ridondanze

Lo schema presenta un valore ridondante, l'attributo *Iscritti* dell'entità *Corso*. La presenza di questo attributo, ipotizzando di usare interi (4 byte per valore), aumenta la memoria di 80 byte essendo l'entità *Corso* presente in media 20 volte nel database [4 * 20].

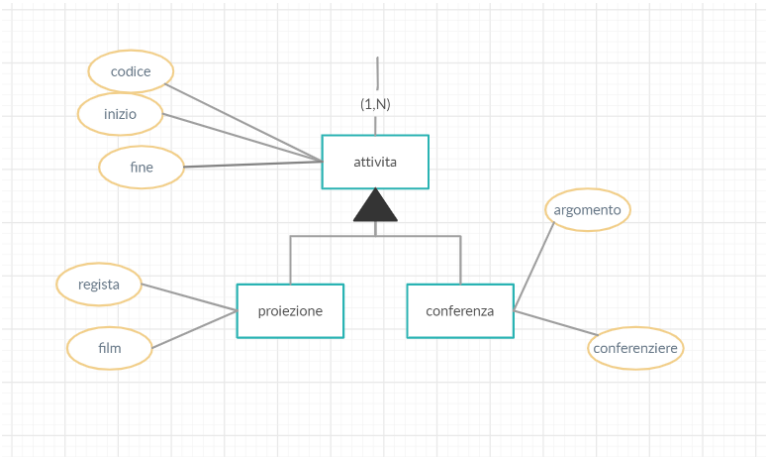
Le operazioni coinvolte sono la 8 e la 12 con un costo totale di 32 [8 + 24].

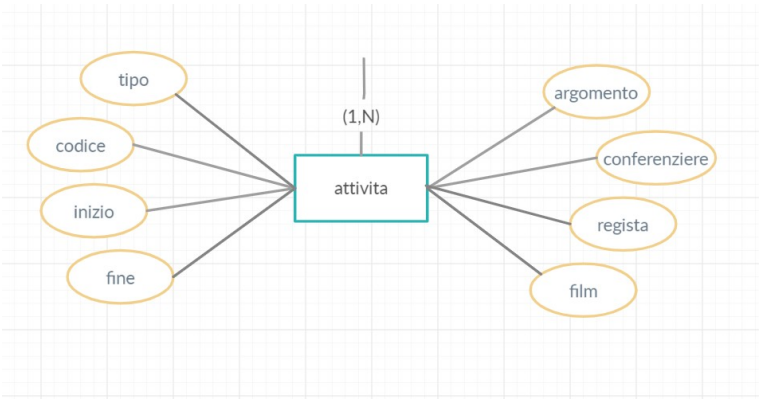
In caso di assenza dell'attributo l'operazione 12 richiederebbe un ulteriore lettura in *Iscrizione* aumentando a valore 10 [(2 + 2 + 1) * 2] e l'operazione 8 richiederebbe una scrittura in meno scendendo a valore 12 [2 * 6]. Perciò il valore totale sarebbe di 22 [12 + 10].

Comparando i risultati si è optato per la seconda opzione, ovvero l'omissione dell'attributo *Iscritti*, per migliorare le prestazioni delle operazioni.

2) Eliminazione delle generalizzazioni

Si è optato per l'accorpamento delle entità figlie *proiezione* e *conferenza* della generalizzazione nel genitore, *attività*. Questo perché le operazioni nel sistema non fanno molta distinzione fra le 3 entità. A tal proposito gli attributi dell'entità figlie sono stati portati nel padre ed è stato aggiunto l'attributo *tipo. (Vedere le due figure)*





3) Scelta degli identificatori primari

***** *Nuovo attributo usato come chiave primaria*

FK: Chiave esterna

ENTITA	CHIAVE
Livello	Username
Corso	Codice, FK(Livello)
Lezione	Numero*, FK(corso)
Insegnante	Codice*

Allievo	Matricola*
Lezione privata	Codice*
Attività	Codice

Trasformazione di attributi e identificatori

Traduzione di entità e associazioni

LIVELLO (NOME, LIBRO, ESAME)

CORSO (CODICE, LIVELLO, ATTIVAZIONE)

LEZIONE (NUMERO, CODICECORSO, LIVELLOCORSO, INIZIO, FINE)

ASSENZA (NUMEROLEZIONE, CODICECORSO, LIVELLOCORSO, ALLIEVO)

ISCRIZIONE (CODICECORSO, LIVELLOCORSO, ALLIEVO, ISCRIZIONE)

DOCENZA (NUMEROLEZIONE, CODICECORSO, LIVELLOCORSO, INSEGNANTE)

ALLIEVO (MATRICOLA, NOME, COGNOME, DATANASCITA, TELEFONO, INDIRIZZO)

INSEGNANTE (CODICE, NOME, INDIRIZZO, NAZIONALITA)

LEZIONEPRIVATA (CODICE, DATA, INSEGNANTE, ALLIEVO)

PARTECIPAZIONE (ALLIEVO, ATTIVITA)

ATTIVITA (CODICE, INIZIO, FINE, TIPO, REGISTA, FILM, ARGOMENTO, CONFERENZIERE)

Vincoli Di integrità:

Corso (Livello) $\subseteq$ Livello (Nome)

Lezione (CodiceCorso, LivelloCorso) $\subseteq$ Corso (Codice, Livello)

Assenza (NumeroLezione, CodiceCorso, LivelloCorso) $\subseteq$ Lezione (Numero, CodiceCorso, LivelloCorso)

Assenza (Allievo) $\subseteq$ Allievo (Matricola)

Iscrizione (CodiceCorso, LivelloCorso) $\subseteq$ Corso (Codice, Livello)

Iscrizione (Allievo) $\subseteq$ Allievo (Matricola)

Docenza (NumeroLezione, CodiceCorso, LivelloCorso) $\subseteq$ Lezione (Numero, CodiceCorso, LivelloCorso)

Docenza (insegnante) $\subseteq$ Insegnante (Codice)

LezionePrivata (Insegnante) $\subseteq$ Insegnante (Codice)

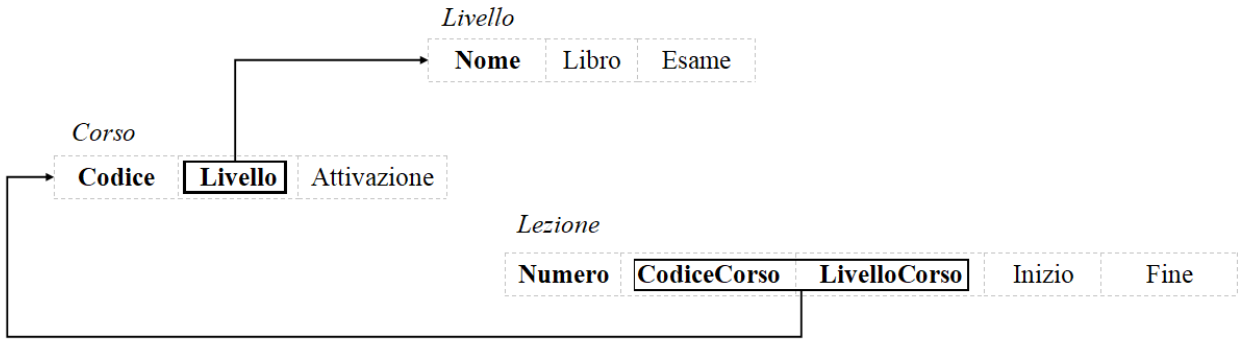
LezionePrivata (Allievo) $\subseteq$ Allievo (Matricola)

Partecipazione (Allievo) $\subseteq$ Allievo (Matricola)

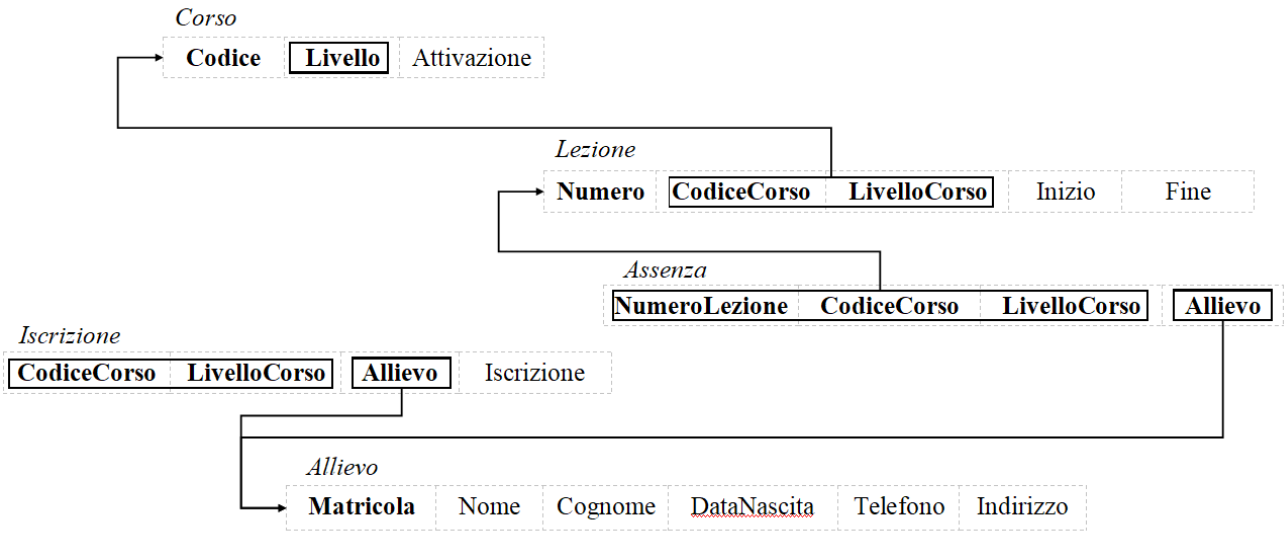
Partecipazione (Attivita) $\subseteq$ Attivita (Codice)

- Per la rappresentazione grafica del modello relazione si è preferito per leggibilità dividerlo in sottoporzioni.

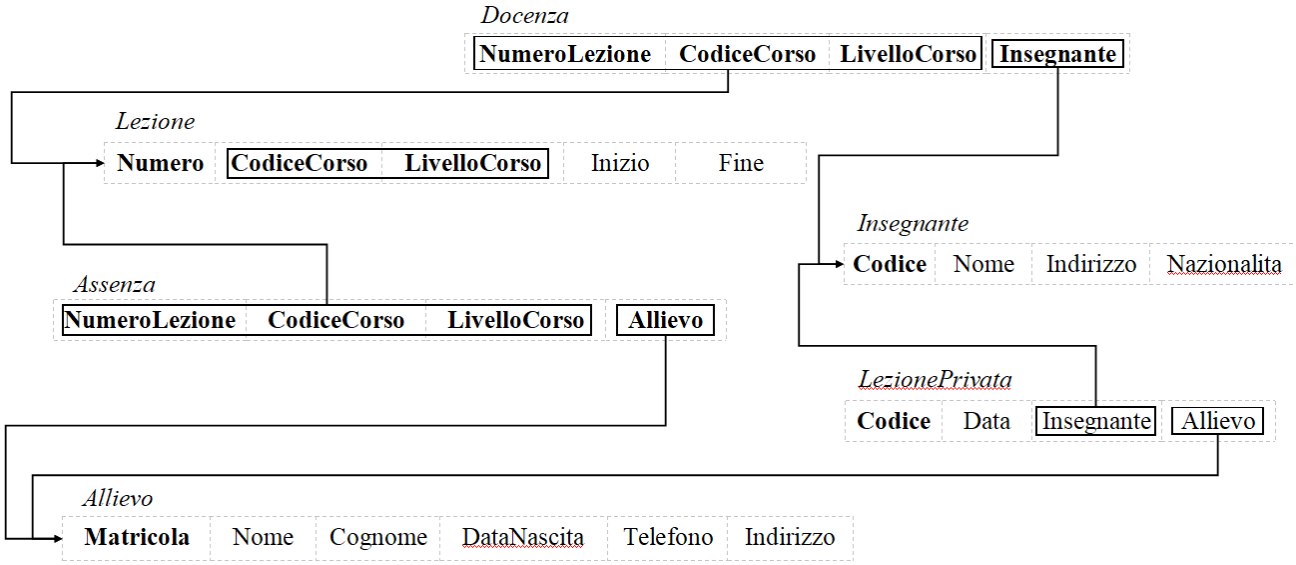
LIVELLO / CORSO / LEZIONE



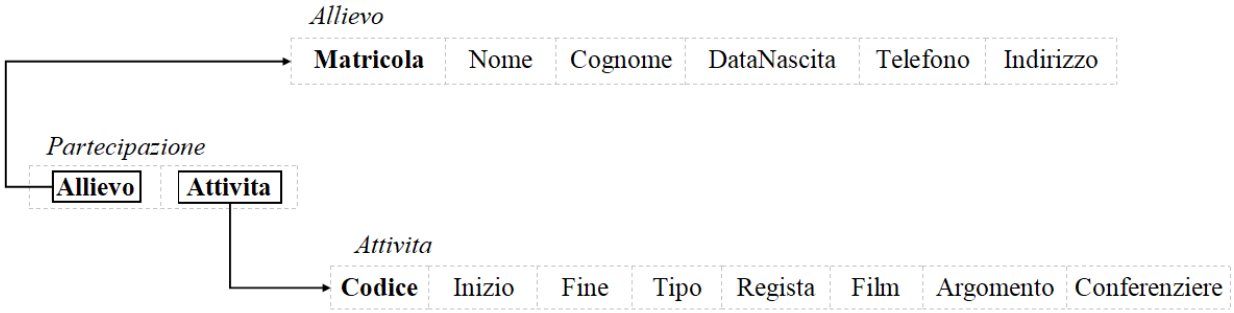
CORSO / LEZIONE / ASSENZA / ISCRIZIONE / ALLIEVO



CORSO / ISCRIZIONE / ALLIEVO / DOCENZA / INSEGNANTE / LEZIONEPRIVATA



ALLIEVO / PARTECIPAZIONE / ATTIVITA



Normalizzazione del modello relazionale

Il sistema non ha avuto bisogno di modifiche di normalizzazione, essendo ogni elemento conforme alle 3 Forme Normali(1NF, 2NF e 3NF).

5. Progettazione fisica

Utenti e privilegi

Si è optato per un controllo più specifico del database tramite le Stored Procedures, lasciando solo la funzionalità di "SELECT" per le varie tabelle.

Sono stati implementati 5 utenti con i rispettivi ruoli:

1) **Login:**

Descrizione: Utente iniziale per effettuare il login.

2) **Amministratore**:

Descrizione: E' la figura con più funzionalità: Si occupa della gestione dei corsi, degli insegnanti e delle attività.

Privilegi: "SELECT" delle tabelle *Allievo, Corso, Lezione, Attivita, Insegnante, Livello* per agevolare all'utente l'esecuzione delle operazioni principali.

Esegue le seguenti Stored Procedures:

- *Iscrivi_allievo:* Operazione 1 delle specifiche.
- *Iscrivi_insegnante:* Operazione 2 delle specifiche.
- *Crea_corso:* Operazione 3 delle specifiche.
- *Aggiungi_lezione*: Operazione 4 delle specifiche.
- *Aggiungi_lezioneprivata:* Operazione 5 delle specifiche.
- *Crea_attivita*: Operazione 6 delle specifiche.
- *Report_amministratore:* Operazione 11 delle specifiche.
- *Reset_anno:* Operazione 13 delle specifiche.
- *Verifica_interferenza_orari:* Per soddisfare la regola aziendale 1

3) **Segreteria**:

Descrizione: Si occupa della gestione degli allievi, registrando assenze, iscrizioni, lezioni private e partecipazioni ad attività.

Privilegi: “SELECT” delle tabelle *Allievo, Corso, Lezione, Attivita* per agevolare all’utente l’esecuzione delle operazioni principali.

Esegue le seguenti Stored Procedures:

- *Aggiungi_assenza:* Operazione 7 delle specifiche.
- *Iscrivi_allievo_corso:* Operazione 8 delle specifiche.
- *Partecipa_attivita:* Operazione 9 delle specifiche.
- *Verifica_interferenza_orari:* Per soddisfare la regola aziendale 1

4) **Insegnante**:

Descrizione: E’ una figura lavorativa dell’istituto. L’applicazione gli permette di ottenere il report delle sue attività di insegnamento con l’erogazione delle lezioni regolari - dei corsi – e di quelle private su base settimanale.

Privilegi:

Esegue le seguenti Stored Procedures:

- *Report_insegnante:* Operazione 10 delle specifiche.

5) **Allievo**:

Descrizione: E’ l’allievo dell’istituto. L’applicazione gli permette solo di avere le informazioni sulle sue attività all’interno dell’istituto

Privilegi: “SELECT” delle tabelle *Allievo, LezionePrivata, Partecipazione, Iscrizione, Assenza*.

Strutture di memorizzazione

Tabella Allievo		
Attributo	**Tipo di dato**	**Attributi**
Matricola	INT	PK, AI
Username	VARCHAR(45)	NN, UQ
Nome	VARCHAR(45)	NN

Cognome	VARCHAR(45)	NN
Telefono	BIGINT	
DataNascita	DATE	
Indirizzo	VARCHAR(45)	

Tabella Assenza		
Attributo	**Tipo di dato**	**Attributi**
CodiceCorso	INT	PK
LivelloCorso	VARCHAR(45)	PK
NumeroLezione	INT	PK
Allievo	INT	PK

Tabella Attivita		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK, AI
Inizio	TIMESTAMP	NN
Fine	TIMESTAMP	NN
Tipo	INT	NN
Regista	VARCHAR(45)	
Film	VARCHAR(45)	
Argomento	VARCHAR(45)	
Conferenziere	VARCHAR(45)	

Tabella Corso		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK, AI
Livello	VARCHAR(45)	PK

Attivazione	DATE	NN

Tabella Docenza		
Attributo	**Tipo di dato**	**Attributi**
CodiceCorso	INT	PK
LivelloCorso	VARCHAR(45)	PK
NumeroLezione	INT	PK
Insegnante	INT	PK

Tabella Insegnante		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK, AI
Username	VARCHAR(45)	NN, UQ
Nome	VARCHAR(45)	NN
Indirizzo	VARCHAR(45)	
Nazionalita	VARCHAR(45)	

Tabella Iscrizione		
Attributo	**Tipo di dato**	**Attributi**
CodiceCorso	INT	PK
LivelloCorso	VARCHAR(45)	PK
Allievo	INT	PK
Iscrizione	DATE	

Tabella Lezione		
Attributo	**Tipo di dato**	**Attributi**
CodiceCorso	INT	PK
LivelloCorso	VARCHAR(45)	PK
Numero	INT	PK, AI

Inizio	TIMESTAMP	NN
Fine	TIMESTAMP	NN

Tabella LezionePrivata		
Attributo	**Tipo di dato**	**Attributi**
Codice	INT	PK, AI
Inizio	TIMESTAMP	NN
Fine	TIMESTAMP	NN
Insegnante	INT	NN
Allievo	INT	NN

Tabella Livello		
Attributo	**Tipo di dato**	**Attributi**
Nome	VARCHAR(45)	PK
Libro	VARCHAR(45)	NN
Esame	TINYINT	NN

Tabella Partecipazione		
Attributo	**Tipo di dato**	**Attributi**
Allievo	INT	PK
Attivita	INT	PK

Tabella Utente		
Attributo	**Tipo di dato**	**Attributi**
Username	VARCHAR(45)	PK
Password	VARCHAR(45)	NN
Ruolo	ENUM	NN

Indici

Tabella Allievo	
Indice	**Tipo**
Matricola	PR
Username	IDX, UQ

Tabella Assenza	
Indice	**Tipo**
CodiceCorso, LivelloCorso, NumeroLezione, Allievo	PR
CodiceCorso, LivelloCorso, NumeroLezione	IDX
Allievo	IDX

Tabella Attivita	
Indice	**Tipo**
Codice	PR

Tabella Corso	
Indice	**Tipo**
Codice, Livello	PR
Livello	IDX

Tabella Docenza	
Indice	**Tipo**
CodiceCorso, LivelloCorso, NumeroLezione, Insegnante	PR

CodiceCorso, LivelloCorso, NumeroLezione	IDX
Insegnante	IDX

Tabella Insegnante	
Indice	**Tipo**
Codice	PR
Username	IDX, UQ

Tabella Iscrizione	
Indice	**Tipo**
CodiceCorso, LivelloCorso, Allievo	PR
Allievo	IDX

Tabella Lezione	
Indice	**Tipo**
CodiceCorso, LivelloCorso, Numero	PR
CodiceCorso, LivelloCorso	IDX

Tabella LezionePrivata	
Indice	**Tipo**
Codice	PR
Insegnante	IDX
Allievo	IDX

Tabella Livello	
Indice	**Tipo**
Nome	PR

Tabella Partecipazione

Indice	Tipo
Allievo, Attivita	PR
Attivita	IDX

Tabella Utente	
Indice	**Tipo**
Username	PR

Trigger

Sulla tabella *Assenza* viene implementato un Trigger BEFORE_INSERT per verificare con una SELECT se l'utente è realmente registrato al corso a cui fanno riferimento i valori da inserire (Regola aziendale 2).

```
CREATE DEFINER = CURRENT_USER TRIGGER `gestione-corsi`.`assenza_BEFORE_INSERT` BEFORE INSERT ON `assenza` FOR EACH ROW
BEGIN
    declare counter int;
    select count(*) from iscrizione where codicecorso = NEW.codicecorso
                                    AND livellocorso = NEW.livellocorso AND allievo = NEW.allievo
        into counter;
    if counter = 0 then
        signal sqlstate '45002'  set message_text = "The student is not registered to the course";
    end if;
END
```

Eventi

Non sono stati implementati eventi nel sistema.

Viste

Sono state implementate 4 View:

- *report_lezioni_mensile e report_lezioniprivate_mensile* utilizzate dalla Stored procedure *report_amministratore.*
- *report_lezioni_settimanale e report_lezioniprivate_settimanale* utilizzate dalla Stored procedure *report_insegnante*.

1) *report_lezioni_mensile e report_lezioni_settimanale* hanno una struttura similare, l'unica differenza consiste nell'ordinamento su base mensile o settimanale. Si basano sul join fra le

tabelle *Docenza* e *Lezione*: Viene selezionato l'identificatore dell'insegnante da *Docenza* e tutte le informazioni della lezione aggiungendo un ulteriore attributo "allievo" con valore NULL per allinearsi rispettivamente alle strutture di *report_lezioniprivate_mensile e report_lezioniprivate_settimanale.*

```
CREATE VIEW `report_lezioni_mensile` AS
    Select D.insegnante as insegnante, year(L.inizio) as anno,
        month(L.inizio) as mese, "Lezione corso" as tipo,
        L.codicecorso as corso, L.livellocorso as livello,
        L.numero as lezione,    NULL as allievo, L.inizio as inizio, L.fine as fine
    from docenza as D join lezione as L on D.codicecorso = L.codicecorso
        AND D.livellocorso = L.livellocorso AND D.numerolezione = L.numero
    order by insegnante, anno, mese, inizio;
```

```
CREATE VIEW `report_lezioni_settimanale`AS
    Select D.insegnante as insegnante, year(L.inizio) as anno,
        week(L.inizio) as settimana, "Lezione corso" as tipo,
        L.codicecorso as corso, L.livellocorso as livello,
        L.numero as lezione,    NULL as allievo, L.inizio as inizio, L.fine as fine
    from docenza as D join lezione as L on D.codicecorso = L.codicecorso
        AND D.livellocorso = L.livellocorso AND D.numerolezione = L.numero
    order by insegnante, anno, settimana, inizio;
```

2) *report_lezioniprivate_mensile* e *report_lezioniprivate_settimanale* hanno una struttura similare, l'unica differenza consiste nell'ordinamento su base mensile o settimanale. Si basano sul join fra le tabelle Docenza e Lezione: Vengono selezionate tutte le informazioni della lezione aggiungendo gli attributi *corso, livello* e *lezione* con valore NULL per allinearsi *rispettivamente* alle strutture di *report_lezioni_mensile e report_lezioni_settimanale.*

```
CREATE VIEW `report_lezioniprivate_mensile` AS
    Select LP.insegnante as insegnante, year(LP.inizio) as anno,
        month(LP.inizio) as mese, "Lezione privata" as tipo,
        NULL as corso, NULL as livello, NULL as lezione, A.matricola as allievo,
        LP.inizio as inizio, LP.fine as fine
    from lezioneprivata as LP join allievo as A on LP.allievo = A.matricola
    order by insegnante, anno, mese, inizio;
```

```
CREATE VIEW `report_lezioniprivate_settimanale` AS
    Select LP.insegnante as insegnante, year(LP.inizio) as anno,
        week(LP.inizio) as settimana, "Lezione privata" as tipo,
        NULL as corso, NULL as livello, NULL as lezione, A.matricola as allievo,
        LP.inizio as inizio, LP.fine as fine
    from lezioneprivata as LP join allievo as A on LP.allievo = A.matricola
    order by insegnante, anno, settimana, inizio;
```

Stored Procedures e transazioni

- *Aggiungi_assenza*: Inserimento.

```
CREATE PROCEDURE `aggiungi_assenza` (IN var_cod int, IN var_liv varchar(45),
                                     IN var_num int, IN var_all int)
BEGIN
    insert into assenza(codicecorso, livellocorso, numerolezione, allievo) values(var_cod, var_liv, var_num, var_all);
END
```

- *Crea_attivita*: Inserimento con ritorno del codice auto-incrementativo.

```
CREATE PROCEDURE `crea_attivita` (IN var_inizio timestamp, IN var_fine timestamp, IN var_tipo int,
                                  IN var_reg varchar(45), IN var_film varchar(45), IN var_arg varchar(45),
                                  IN var_conf varchar(45), OUT var_codice int)
BEGIN
    insert into attivita (inizio, fine, tipo, regista, film, argomento, conferenziere)
        values(var_inizio, var_fine, var_tipo, var_reg, var_film, var_arg, var_conf);
    set var_codice = last_insert_id();
END
```

- *Iscrivi_allievo*: Inserimento.

```
CREATE PROCEDURE `iscrivi_allievo` (IN var_username varchar(45), IN var_pass varchar(45), IN var_nome varchar(45),
                                    IN var_cogn varchar(45), IN var_tel bigint,
                                        IN var_dat date, IN var_ind varchar(45), OUT var_matricola int)
BEGIN
    insert into utente(username, `password`, ruolo) values(var_username, var_pass, "allievo");
    insert into allievo(username, nome, cognome, telefono, datanascita, indirizzo)
        values (var_username, var_nome, var_cogn, var_tel, var_dat, var_ind);
    set var_matricola = last_insert_id();
END
```

- *Iscrivi_insegnante*: Inserimento

```
CREATE PROCEDURE `iscrivi_insegnante` (IN var_username varchar(45), IN var_pass varchar(45), IN var_nome varchar(45),
                                       IN var_ind varchar(45), IN var_naz varchar(45), OUT var_codice int)
BEGIN
    insert into utente(username, `password`, ruolo) values(var_username, var_pass, "insegnante");
    insert into insegnante(username, nome, indirizzo, nazionalita)
        values (var_username, var_nome, var_ind, var_naz);
    set var_codice = last_insert_id();
END
```

- *Partecipa_attivita*: Inserimento

```
CREATE PROCEDURE `partecipa_attivita` (IN var_all int, IN var_att int)
BEGIN
    insert into partecipazione(allievo, attivita) values(var_all, var_att);
END
```

- *Reset_anno*: Transazione con livello Serializable per evitare tutte le anomalie essendo l'operazione di reset totale del sistema.

```
CREATE PROCEDURE `reset_anno` ()
BEGIN
    set transaction isolation level serializable;
    start transaction;
        DELETE FROM assenza;
        DELETE FROM iscrizione;
        DELETE FROM partecipazione;
        DELETE FROM docenza;
        DELETE FROM lezione;
        DELETE FROM corso;
        DELETE FROM lezioneprivata;
        DELETE FROM attivita;
        DELETE FROM allievo;
        DELETE FROM insegnante;
        DELETE FROM utente WHERE username != "admin" AND username != "segreteria";
    commit;
END
```

- *Login*: In base al valore di user_name e var_pass viene restituito come intero il ruolo dell'utente.

```
CREATE PROCEDURE `login` (in var_username varchar(45), in var_pass varchar(45), out var_role INT)
BEGIN
    declare var_user_role ENUM('amministratore', 'insegnante', 'allievo', 'segreteria');

    select `ruolo` from `utente`
        where `username` = var_username
        and `password` = var_pass
        into var_user_role;

        if var_user_role = 'amministratore' then
            set var_role = 1;
        elseif var_user_role = 'segreteria' then
            set var_role = 2;
        elseif var_user_role = 'insegnante' then
            set var_role = 3;
        elseif var_user_role = 'allievo' then
            set var_role = 4;
        else
            set var_role = 5;
        end if;
END
```

- Verifica_interferenza_orari: Verifica che gli orari di *var_inizio* e *var_fine* non vadano in conflitto con altri orari dell'insegnante, passato in ingresso con *var_ins*.
  Vengono dichiarati due cursori *cur1* e *cur2* per selezionare rispettivamente informazioni su *Docenza* e *LezionePrivate* dell'insegnante.
  Mentre *Cur2* ricava subito gli orari, *Cur1* esegue prima il fetch di *CodiceCorso*, *LivelloCorso* e *NumeroLezione* che utilizza per selezionare gli orari da *Lezione*.

```
CREATE PROCEDURE `verifica_interferenza_orari` (IN var_ins int, IN var_inizio timestamp, IN var_fine timestamp, OUT var_out int)
BEGIN
    -- if the output is 0 then the lesson is in opposition with other private or normal teacher's lessons
    declare temp_corso int;
    declare temp_livello varchar(45);
    declare temp_numero int;
    declare temp_inizio timestamp;
    declare temp_fine timestamp;
    declare done int default false;
    declare cur1 cursor for select codicecorso, livellocorso, numerolezione from docenza where insegnante = var_ins;
    declare cur2 cursor for select inizio, fine from lezioneprivata where insegnante = var_ins;
    declare continue handler for not found set done = true;

    -- set output to default
    set var_out = 1;

    -- verify the interference with normal lessons
    open cur1;
    read_loop: loop
        fetch cur1 into temp_corso, temp_livello, temp_numero;
        if done then
            leave read_loop;
        end if;
        select inizio, fine from lezione
            where codicecorso = temp_corso and livellocorso = temp_livello and numero = temp_numero
                into temp_inizio, temp_fine;

        if ((var_fine > temp_inizio and var_fine < temp_fine)
            or (var_inizio <= temp_inizio and var_fine >= temp_fine)
            or (var_inizio > temp_inizio and var_inizio < temp_fine))
                then set var_out = 0;
        end if;
    end loop;
```

```
    end loop;
    close cur1;

    -- verify the interference with private lessons
    set done = false;
    open cur2;
    read_loop: loop
        fetch cur2 into temp_inizio, temp_fine;
        if done then
            leave read_loop;
        end if;

        if ((var_fine > temp_inizio and var_fine < temp_fine)
            or (var_inizio <= temp_inizio and var_fine >= temp_fine)
            or (var_inizio > temp_inizio and var_inizio < temp_fine))
                then set var_out = 0;
        end if;
    end loop;
    close cur2;
END
```

- *Aggiungi_lezione*: Verifica con una SELECT se esistono il corso e l'insegnante passati in ingresso e con la Stored procedure v*erifica_interferenza_orari()* se ci sono interferenze con altri orari dell'insegnante. Per questo motivo si è scelto un livello di transazione Read Committed per evitare letture sporche e inconsistenti.
  Vengono inseriti lezione e docenza e vengono restituiti i rispettivi ID auto-incrementativi.

```
CREATE PROCEDURE `aggiungi_lezione` (IN var_corso int, IN var_livello varchar(45),
                                     IN var_inizio timestamp, IN var_fine timestamp,
                                         IN var_insegnante int, OUT var_numero int)
BEGIN
    declare count_corso int;
    declare count_ins int;
    declare verif int;
    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;
    set transaction isolation level read committed;
    start transaction;
        select count(*) from corso where codice = var_corso and livello = var_livello into count_corso;
        select count(*) from insegnante where codice = var_insegnante into count_ins;
        call verifica_interferenza_orari(var_insegnante, var_inizio, var_fine, verif);
        if count_corso != 1 then
            signal sqlstate '45000' set message_text = "The course doesn't exist";
        elseif count_ins != 1 then
            signal sqlstate '45000' set message_text = "the teacher doesn't exist";
        elseif verif = 0 then
            signal sqlstate '45001' set message_text = "Time interference with teacher's lessons";
        end if;
        insert into lezione(codicecorso, livellocorso, inizio, fine) values (var_corso, var_livello, var_inizio, var_fine);
        set var_numero = last_insert_id();
        insert into docenza (codicecorso, livellocorso, numerolezione, insegnante) values (var_corso, var_livello, var_numero, var_insegnante);
        set var_numero = last_insert_id();
    commit;
END
```

- *Aggiungi_lezioneprivata:* Viene inserita e restituito l'ID autoincrementativo di LezionePrivata.

  Ha una funzione parallela ad *Aggiungi_lezione().* Infatti si verifica prima la presenza di corso e insegnante, e la mancanza di interferenze con altri orari adottando un transazione di livello Read Committed.

```
CREATE PROCEDURE `aggiungi_lezioneprivata` (IN var_inizio timestamp, IN var_fine timestamp, In var_ins int, IN var_all int, OUT var_codice int)
BEGIN
    declare verif int;
    declare count_all int;
    declare count_ins int;
    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;
    set transaction isolation level read committed;
    start transaction;
        select count(*) from insegnante where codice = var_ins into count_ins;
        select count(*) from allievo where matricola = var_all into count_all;

        set verif = 1;
        call verifica_interferenza_orari(var_ins, var_inizio, var_fine, verif);

        if count_ins != 1 then
            signal sqlstate '45000' set message_text = "The teacher doesn't exist";
        elseif count_all != 1 then
            signal sqlstate '45000' set message_text = "The student doesn't exist";
        elseif verif = 0 then
            signal sqlstate '45001' set message_text = "Time interference with teacher's lessons";
        end if;

        insert into lezioneprivata(inizio, fine, insegnante, allievo) values (var_inizio, var_fine, var_ins, var_all);
        set var_codice = last_insert_id();
    commit;
END
```

- Crea_corso: Inserisce e restituisce l'ID auto-incrementativo di Corso.
  Utilizza un livello di isolamento Read Committed per garantire una lettura corretta con la SELECT di verifica in Livello.

```
CREATE PROCEDURE `crea_corso` (IN var_livello varchar(45), OUT var_codice int)
BEGIN
    declare counter int;
    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;
    set transaction isolation level read committed;
    start transaction;
        select count(*) from livello where nome = var_livello into counter;
        if counter != 1 then
            signal sqlstate '45000' set message_text = "Level doesn't exist";
        end if;
        insert into corso (livello, attivazione) values(var_livello, curdate());
        set var_codice = last_insert_id();
    commit;
END
```

- Iscrivi_allievo_corso: Ha un funzionamento similare a *crea_corso. Infatti* utilizza un livello di isolamento Read Committed per garantire una lettura corretta con la SELECT di verifica in Corso.
  Esegue un INSERT dei valori in ingresso in iscrizione.

```
CREATE PROCEDURE `iscrivi_allievo_corso` (IN var_corso int, IN var_liv varchar(45), IN var_all int)
BEGIN
    declare counter int;
    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;
    set transaction isolation level read committed;
    start transaction;
        select count(*) from corso where codice = var_corso and livello = var_liv into counter;
        if counter != 1 then
            signal sqlstate '45000' set message_text = "The course doesn't exist";
        end if;
        insert into iscrizione (codicecorso,livellocorso,allievo,iscrizione) values (var_corso, var_liv, var_all, curdate());
    commit;
END
```

- *Report_amministratore*: Esegue il report delle attività svolte dall'insegnante su base mensile. Utilizza il cursore *cur* per selezionare i codici di tutti gli insegnanti. Per ogni insegnante viene fatta una SELECT sull'UNION delle viste report_lezioni_mensile e report_lezioniprivate_mensile, che presentano gli stessi attributi.
  Per evitare interferenze con altre procedure viene applicata una transazione con livello Serializable.

```
CREATE PROCEDURE `report_amministratore` ()
BEGIN
    declare done int default false;
    declare temp_ins int;
    declare cur cursor for select codice from insegnante order by codice;
    declare continue handler for not found set done = true;
    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;
    set transaction isolation level serializable;
    start transaction;
    -- verify the interference with normal lessons
        select * from insegnante order by codice;
        open cur;
        read_loop: loop
            fetch cur into temp_ins;
            if done then
                leave read_loop;
            end if;
            select anno, mese, tipo, corso, livello, lezione, allievo, inizio, fine from (select * from report_lezioni_mensile
                UNION select * from report_lezioniprivate_mensile) as result
                    where insegnante = temp_ins order by insegnante, anno, mese, inizio;
        end loop;
        close cur;
    commit;

END
```

- Report_insegnate: Ritorna il calendario delle attività di un insegnante su base settimanale.
  Esegue una SELECT sulla UNION fra le viste *report_lezioni_settimanale* e *report_lezioniprivate_settimanale* filtrando le tuple con codice insegnante uguale al paramentro in ingresso.
  Per evitare interferenze con altre procedure viene applicata una transazione con livello Serializable.

```
CREATE PROCEDURE `report_insegnante` (IN var_username varchar(45))
BEGIN
    declare temp_ins int;
    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;
    set transaction isolation level serializable;
    start transaction;
    -- verify the interference with normal lessons
        select * from insegnante where username = var_username;
        select codice from insegnante where username = var_username into temp_ins;
        select anno, settimana as sett, tipo, corso, livello, lezione, allievo, inizio, fine from (select * from report_lezioni_settimanale
            UNION select * from report_lezioniprivate_settimanale) as result
                where insegnante = temp_ins order by insegnante, anno, settimana, inizio;
    commit;
END
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
set @@session.explicit_defaults_for_timestamp=on;

DROP SCHEMA IF EXISTS `gestione-corsi`;

CREATE SCHEMA IF NOT EXISTS `gestione-corsi`;

USE `gestione-corsi`;


CREATE TABLE IF NOT EXISTS `gestione-corsi`.`livello` (
  `nome` VARCHAR(45) NOT NULL,
  `libro` VARCHAR(45) NOT NULL,
  `esame` TINYINT NULL,
  PRIMARY KEY (`nome`))
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `gestione-corsi`.`corso` (
  `codice` INT NOT NULL AUTO_INCREMENT,
  `livello` VARCHAR(45) NOT NULL,
  `attivazione` DATE NOT NULL,
  PRIMARY KEY (`codice`, `livello`),
  INDEX `fk_corso_livello_idx` (`livello` ASC),
  CONSTRAINT `fk_corso_livello`
    FOREIGN KEY (`livello`)
    REFERENCES `gestione-corsi`.`livello` (`nome`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `gestione-corsi`.`lezione` (
  `codicecorso` INT NOT NULL,
  `livellocorso` VARCHAR(45) NOT NULL,
  `numero` INT NOT NULL AUTO_INCREMENT,
  `inizio` TIMESTAMP NOT NULL,
  `fine` TIMESTAMP NOT NULL,
  PRIMARY KEY (`numero`, `codicecorso`, `livellocorso`),
  INDEX `fk_lezione_corso_idx` (`codicecorso` ASC, `livellocorso` ASC),
  CONSTRAINT `fk_lezione_corso`
    FOREIGN KEY (`codicecorso` , `livellocorso`)
    REFERENCES `gestione-corsi`.`corso` (`codice` , `livello`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `gestione-corsi`.`utente` (
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `ruolo` ENUM('amministratore', 'allievo', 'insegnante', 'segreteria') NOT NULL,
  PRIMARY KEY (`username`))
```

```
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `gestione-corsi`.`allievo` (
  `matricola` INT NOT NULL AUTO_INCREMENT,
  `username` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `cognome` VARCHAR(45) NOT NULL,
  `telefono` BIGINT NULL,
  `datanascita` DATE NULL,
  `indirizzo` VARCHAR(45) NULL,
  PRIMARY KEY (`matricola`),
  INDEX `fk_allievo_utente_idx` (`username` ASC),
  UNIQUE INDEX `username_UNIQUE` (`username` ASC),
  CONSTRAINT `fk_allievo_utente`
    FOREIGN KEY (`username`)
    REFERENCES `gestione-corsi`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `gestione-corsi`.`assenza` (
  `codicecorso` INT NOT NULL,
  `livellocorso` VARCHAR(45) NOT NULL,
  `numerolezione` INT NOT NULL,
  `allievo` INT NOT NULL,
```

```
  PRIMARY KEY (`codicecorso`, `allievo`, `livellocorso`, `numerolezione`),
  INDEX `fk_assenza_allievo_idx` (`allievo` ASC),
  INDEX `fk_assenza_lezione_idx` (`codicecorso` ASC, `livellocorso` ASC, `numerolezione` ASC),
  CONSTRAINT `fk_assenza_lezione`
    FOREIGN KEY (`codicecorso` , `livellocorso` , `numerolezione`)
    REFERENCES `gestione-corsi`.`lezione` (`codicecorso` , `livellocorso` , `numero`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_assenza_allievo`
    FOREIGN KEY (`allievo`)
    REFERENCES `gestione-corsi`.`allievo` (`matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `gestione-corsi`.`insegnante` (
  `codice` INT NOT NULL AUTO_INCREMENT,
  `username` VARCHAR(45) NOT NULL,
  `nome` VARCHAR(45) NOT NULL,
  `indirizzo` VARCHAR(45) NULL,
  `nazionalita` VARCHAR(45) NULL,
  PRIMARY KEY (`codice`),
  INDEX `ff_insegnante_utente_idx` (`username` ASC),
  UNIQUE INDEX `username_UNIQUE` (`username` ASC),
```

```
  CONSTRAINT `ff_insegnante_utente`
    FOREIGN KEY (`username`)
    REFERENCES `gestione-corsi`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE TABLE IF NOT EXISTS `gestione-corsi`.`docenza` (
  `codicecorso` INT NOT NULL,
  `livellocorso` VARCHAR(45) NOT NULL,
  `numerolezione` INT NOT NULL,
  `insegnante` INT NOT NULL,
  PRIMARY KEY (`codicecorso`, `insegnante`, `livellocorso`, `numerolezione`),
  INDEX `fk_docenza_insegnante_idx` (`insegnante` ASC),
  INDEX `fk_docenza_lezione_idx` (`codicecorso` ASC, `livellocorso` ASC, `numerolezione` ASC),
  CONSTRAINT `fk_docenza_lezione`
    FOREIGN KEY (`codicecorso` , `livellocorso` , `numerolezione`)
    REFERENCES `gestione-corsi`.`lezione` (`codicecorso` , `livellocorso` , `numero`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_docenza_insegnante`
    FOREIGN KEY (`insegnante`)
    REFERENCES `gestione-corsi`.`insegnante` (`codice`)
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `gestione-corsi`.`iscrizione` (
  `codicecorso` INT NOT NULL,
  `livellocorso` VARCHAR(45) NOT NULL,
  `allievo` INT NOT NULL,
  `iscrizione` DATE NOT NULL,
  PRIMARY KEY (`codicecorso`, `livellocorso`, `allievo`),
  INDEX `fk_iscrizione_allievo_idx` (`allievo` ASC),
  CONSTRAINT `fk_iscrizione_corso`
    FOREIGN KEY (`codicecorso` , `livellocorso`)
    REFERENCES `gestione-corsi`.`corso` (`codice` , `livello`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_iscrizione_allievo`
    FOREIGN KEY (`allievo`)
    REFERENCES `gestione-corsi`.`allievo` (`matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `gestione-corsi`.`lezioneprivata` (
  `codice` INT NOT NULL AUTO_INCREMENT,
```

```
  `inizio` TIMESTAMP NOT NULL,
  `fine` TIMESTAMP NOT NULL,
  `insegnante` INT NOT NULL,
  `allievo` INT NOT NULL,
  PRIMARY KEY (`codice`),
  INDEX `fk_lezioneprivata_insegnante_idx` (`insegnante` ASC),
  INDEX `fk_lezioneprivata_allievo_idx` (`allievo` ASC),
  CONSTRAINT `fk_lezioneprivata_insegnante`
    FOREIGN KEY (`insegnante`)
    REFERENCES `gestione-corsi`.`insegnante` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_lezioneprivata_allievo`
    FOREIGN KEY (`allievo`)
    REFERENCES `gestione-corsi`.`allievo` (`matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `gestione-corsi`.`attivita` (
  `codice` INT NOT NULL AUTO_INCREMENT,
  `inizio` TIMESTAMP NOT NULL,
  `fine` TIMESTAMP NOT NULL,
  `tipo` INT NOT NULL,
```

```
  `regista` VARCHAR(45) NULL,
  `film` VARCHAR(45) NULL,
  `argomento` VARCHAR(45) NULL,
  `conferenziere` VARCHAR(45) NULL,
  PRIMARY KEY (`codice`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `gestione-corsi`.`partecipazione` (
  `allievo` INT NOT NULL,
  `attivita` INT NOT NULL,
  PRIMARY KEY (`allievo`, `attivita`),
  INDEX `fk_partecipazione_attivita_idx` (`attivita` ASC),
  CONSTRAINT `fk_partecipazione_allievo`
    FOREIGN KEY (`allievo`)
    REFERENCES `gestione-corsi`.`allievo` (`matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_partecipazione_attivita`
    FOREIGN KEY (`attivita`)
    REFERENCES `gestione-corsi`.`attivita` (`codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
DELIMITER ;
SET SQL_MODE = '';
DROP USER IF EXISTS allievo;
SET SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
CREATE USER 'allievo' IDENTIFIED BY 'allievo';

GRANT SELECT ON TABLE `gestione-corsi`.`allievo` TO 'allievo';
SET SQL_MODE = '';
DROP USER IF EXISTS insegnante;
SET SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
CREATE USER 'insegnante' IDENTIFIED BY 'insegnante';

GRANT EXECUTE ON procedure `gestione-corsi`.`report_insegnante` TO 'insegnante';
SET SQL_MODE = '';
DROP USER IF EXISTS segreteria;
SET SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
CREATE USER 'segreteria' IDENTIFIED BY 'segreteria';

GRANT EXECUTE ON procedure `gestione-corsi`.`iscrivi_allievo_corso` TO 'segreteria';
GRANT EXECUTE ON procedure `gestione-corsi`.`aggiungi_assenza` TO 'segreteria';
GRANT EXECUTE ON procedure `gestione-corsi`.`partecipa_attivita` TO 'segreteria';
GRANT EXECUTE ON procedure `gestione-corsi`.`report_insegnante` TO 'segreteria';
GRANT EXECUTE ON procedure `gestione-corsi`.`verifica_interferenza_orari` TO 'segreteria';
```

```
GRANT SELECT ON TABLE `gestione-corsi`.`corso` TO 'segreteria';

GRANT SELECT ON TABLE `gestione-corsi`.`allievo` TO 'segreteria';

GRANT SELECT ON TABLE `gestione-corsi`.`lezione` TO 'segreteria';

GRANT SELECT ON TABLE `gestione-corsi`.`attivita` TO 'segreteria';

SET SQL_MODE = '';

DROP USER IF EXISTS amministratore;

SET SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

CREATE USER 'amministratore' IDENTIFIED BY 'amministratore';


GRANT EXECUTE ON procedure `gestione-corsi`.`aggiungi_lezione` TO 'amministratore';

GRANT EXECUTE ON procedure `gestione-corsi`.`crea_attivita` TO 'amministratore';

GRANT EXECUTE ON procedure `gestione-corsi`.`crea_corso` TO 'amministratore';

GRANT EXECUTE ON procedure `gestione-corsi`.`iscrivi_allievo` TO 'amministratore';

GRANT EXECUTE ON procedure `gestione-corsi`.`iscrivi_insegnante` TO 'amministratore';

GRANT EXECUTE ON procedure `gestione-corsi`.`verifica_interferenza_orari` TO 'amministratore';

GRANT EXECUTE ON procedure `gestione-corsi`.`report_amministratore` TO 'amministratore';

GRANT EXECUTE ON procedure `gestione-corsi`.`aggiungi_lezioneprivata` TO 'amministratore';

GRANT EXECUTE ON procedure `gestione-corsi`.`reset_anno` TO 'amministratore';

GRANT SELECT ON TABLE `gestione-corsi`.`corso` TO 'amministratore';

GRANT SELECT ON TABLE `gestione-corsi`.`lezione` TO 'amministratore';

GRANT SELECT ON TABLE `gestione-corsi`.`attivita` TO 'amministratore';

GRANT SELECT ON TABLE `gestione-corsi`.`insegnante` TO 'amministratore';

GRANT SELECT ON TABLE `gestione-corsi`.`livello` TO 'amministratore';

GRANT SELECT ON TABLE `gestione-corsi`.`allievo` TO 'amministratore';
```

```
SET SQL_MODE = '';
DROP USER IF EXISTS login;
SET SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
CREATE USER 'login' IDENTIFIED BY 'login';

GRANT EXECUTE ON procedure `gestione-corsi`.`login` TO 'login';

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Codice del Front-End

Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include "defines.h"
typedef enum {
	AMMINISTRATORE = 1,
	SEGRETERIA,
	INSEGNANTE,
	ALLIEVO,
```

```
        FAILED_LOGIN
} role_t;
struct configuration conf;
static MYSQL *conn;

static role_t attempt_login(MYSQL *conn, char *username, char *password) {
        MYSQL_STMT *login_procedure;
        MYSQL_BIND param[3]; // Used both for input and output
        int role = 0;
        if(!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn)) {
                print_stmt_error(login_procedure, "Unable to initialize login statement\n");
                goto err2;
        }

        // Prepare parameters
        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
        param[0].buffer = username;
        param[0].buffer_length = strlen(username);
        param[1].buffer_type = MYSQL_TYPE_VAR_STRING; // IN
        param[1].buffer = password;
        param[1].buffer_length = strlen(password);
        param[2].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[2].buffer = &role;
```

```
param[2].buffer_length = sizeof(role);

if (mysql_stmt_bind_param(login_procedure, param) != 0) { // Note _param
        print_stmt_error(login_procedure, "Could not bind parameters for login");
        goto err;
}

// Run procedure
if (mysql_stmt_execute(login_procedure) != 0) {
        print_stmt_error(login_procedure, "Could not execute login procedure");
        goto err;
}

// Prepare output parameters
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
param[0].buffer = &role;
param[0].buffer_length = sizeof(role);

if(mysql_stmt_bind_result(login_procedure, param)) {
        print_stmt_error(login_procedure, "Could not retrieve output parameter");
        goto err;
}
```

```
        // Retrieve output parameter
        if(mysql_stmt_fetch(login_procedure)) {
                print_stmt_error(login_procedure, "Could not buffer results");
                goto err;
        }
        mysql_stmt_close(login_procedure);
        return role;
    err:
        mysql_stmt_close(login_procedure);
    err2:
        return FAILED_LOGIN;
}

int main(void) {
        role_t role;
        if(!parse_config("users/login.json", &conf)) {
                fprintf(stderr, "Unable to load login configuration\n");
                exit(EXIT_FAILURE);
        }
        conn = mysql_init (NULL);
        if (conn == NULL) {
                fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
                exit(EXIT_FAILURE);
        }
```

```
if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
        fprintf (stderr, "mysql_real_connect() failed\n");
        mysql_close (conn);
        exit(EXIT_FAILURE);
    }
    printf("Insert username: ");
    getInput(128, conf.username, false);
    printf("Insert password: ");
    getInput(128, conf.password, true);
    role = attempt_login(conn, conf.username, conf.password);
    switch(role) {
        case AMMINISTRATORE:
            run_as_amministratore(conn);
            break;

        case SEGRETERIA:
            run_as_segreteria(conn);
            break;

        case INSEGNANTE:
            run_as_insegnante(conn);
            break;

         case ALLIEVO:
```

```
                run_as_allievo(conn);
                break;

            case FAILED_LOGIN:
                fprintf(stderr, "Invalid credentials\n");
                exit(EXIT_FAILURE);
                break;

            default:
                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                abort();
        }
        printf("Bye!\n");
        mysql_close (conn);
        return 0;
}
```

Amministratore.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "defines.h"

struct insegn{
   int codice;
```

```
    char username[46];
    char nome[46];
    char indirizzo[46];
    char nazionalita[46];
};



static void reset_anno(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
    char options[2] = {'y','n'};
        char op;
    //Confirmation message
    printf("*** Are you sure?(y/n): ***\n");
    op = multiChoice("Select an option: ", options, 2);

    switch(op) {
       case 'y':
          break;
       case 'n':
          printf("Operation stopped\n");
          return;
       default:
```

```
        fprintf(stderr, "Could not parse the type string\n");
        abort();
    }

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call reset_anno", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize reset year statement\n", false);
        }

        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while the reset year.");
                goto out;
        }

        printf("Operation correctly executed!\n");
    //close statement
    out:
        mysql_stmt_close(prepared_stmt);
}
```

```
static size_t func_ins(MYSQL *conn, MYSQL_STMT *stmt, struct insegn ** ins)
```

```
{
        int status;
        size_t row = 0;
        MYSQL_BIND param[5];

   //Output from stored procedures about teachers
   int codice;
   char username[46];
   char nome[46];
   char indirizzo[46];
   char nazionalita[46];
   //for null results
   my_bool is_null1, is_null2;
        if (mysql_stmt_store_result(stmt)) {
                fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
                fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
                exit(0);
        }
        *ins = malloc(mysql_stmt_num_rows(stmt) * sizeof(struct insegn));

        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_LONG;
        param[0].buffer = &codice;
        param[0].buffer_length = sizeof(codice);
```

```
  param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
   param[1].buffer = username;
   param[1].buffer_length = 46;
   param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
   param[2].buffer = nome;
   param[2].buffer_length = 46;
   param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
   param[3].buffer = indirizzo;
   param[3].buffer_length = 46;
   param[3].is_null = &is_null1;
   param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
   param[4].buffer = nazionalita;
   param[4].buffer_length = 46;
   param[4].is_null = &is_null2;
   if(mysql_stmt_bind_result(stmt, param)) {
            finish_with_stmt_error(conn, stmt, "Unable to bind column parameters\n", true);
   }
   /* assemble Teacher general information */
   while (true) {
            status = mysql_stmt_fetch(stmt);
            if (status == 1 || status == MYSQL_NO_DATA){
                     break;
}
(*ins)[row].codice = codice;
```

```
        strcpy((*ins)[row].username, username);
        strcpy((*ins)[row].nome, nome);
        strcpy((*ins)[row].indirizzo, indirizzo);
        strcpy((*ins)[row].nazionalita, nazionalita);
                row++;
        }
        return row;
}

static void report_amministratore(MYSQL *conn) {
        MYSQL_STMT *prepared_stmt;
        int status;
        struct insegn* ins;
        bool first = true;
        int index = 0;
        char header[512];

        // Print generic information useful for the function
        if(!setup_prepared_stmt(&prepared_stmt, "call report_amministratore", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize report statement\n",
false);
        }

                // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
```

```
            print_stmt_error(prepared_stmt, "An error occurred while retrieving the  report.");
            goto out;
        }

        printf("Administration Report\n");
        do {
            // Skip OUT variables (although they are not present in the procedure...)
            if(conn->server_status & SERVER_PS_OUT_PARAMS) {
                goto next;
            }
            //first select about teacher's information
            if(first) {
                func_ins(conn, prepared_stmt, &ins);
                first = false;
            } else {
                sprintf(header, "\nTeacher code: %d\nUsername: %s\nName: %s\nAddress: %s\nNazionality: %s\n", ins[index].codice, ins[index].username, ins[index].nome, ins[index].indirizzo, ins[index].nazionalita);
        //print results
                dump_result_set(conn, prepared_stmt, header);
                index++;
            }

            // more results? -1 = no, >0 = error, 0 = yes (keep looking)
          next:
```

```
                status = mysql_stmt_next_result(prepared_stmt);
                if (status > 0)
                        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        } while (status == 0);

   //close statement
   out:
        mysql_stmt_close(prepared_stmt);
}

static void registra_allievo(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[8];
        // Input for the registration routine
        char username[46];
        char pass[46];
        char nome[46];
        char cognome[46];
        char telefono[46];
        long telefono_num;
        MYSQL_TIME date_time = {};
         char  date[21];
        char indirizzo[46];
```

```
 //Input
 int matricola;


// Get the required information
printf("\nUsername: ");
  getInput(46, username, false);
 printf("Password: ");
 getInput(46, pass, false);
printf("Nome: ");
getInput(46, nome, false);
printf("Cognome: ");
getInput(46, cognome, false);
  printf("Telefono: ");
  getInput(46, telefono, false);
 printf("Date (YYYY/MM/DD): ");
 getInput(46, date, false);
 printf("Indirizzo: ");
 getInput(46, indirizzo, false);


 telefono_num = strtol(telefono, NULL, 10);


if(convert_date(date, &date_time) == -1){
    fprintf(stderr, "Could not parse date string\n");
```

```
        abort();
    }

        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call iscrivi_allievo(?, ?, ?, ?, ?, ?, ?, ?)", conn)) {
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student's registration statement\n", false);
        }
        // Prepare parameters
        memset(param, 0, sizeof(param));
        param[0].buffer_type =  MYSQL_TYPE_VAR_STRING;
        param[0].buffer = username;
        param[0].buffer_length = strlen(username);
        param[1].buffer_type =  MYSQL_TYPE_VAR_STRING;
        param[1].buffer = pass;
        param[1].buffer_length = strlen(pass);
        param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[2].buffer = nome;
        param[2].buffer_length = strlen(nome);
        param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
        param[3].buffer = cognome;
        param[3].buffer_length = strlen(cognome);
        param[4].buffer_type =  MYSQL_TYPE_LONGLONG;
        param[4].buffer = &telefono_num;
        param[4].buffer_length = sizeof(telefono_num);
```

```
param[5].buffer_type =  MYSQL_TYPE_DATE;
param[5].buffer = &date_time;
param[5].buffer_length = sizeof(date_time);
param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
param[6].buffer = indirizzo;
param[6].buffer_length = strlen(indirizzo);
param[7].buffer_type = MYSQL_TYPE_LONG; // OUT
param[7].buffer = &matricola;
param[7].buffer_length = sizeof(matricola);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student's registration\n", true);
}
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while student's registration.");
        goto out;
}
// Get back the ID of the newly-added student
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
param[0].buffer = &matricola;
param[0].buffer_length = sizeof(matricola);

if(mysql_stmt_bind_result(prepared_stmt, param)) {
```

```
		finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
	}
	// Retrieve output parameter
	if(mysql_stmt_fetch(prepared_stmt)) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
	}
	printf("Student's registration correctly added with ID %d...\n", matricola);
  out:
	mysql_stmt_close(prepared_stmt);
}
static void registra_insegnante(MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[6];

	// Input
	char username[46];
	char pass[46];
	char nome[46];
	char indirizzo[46];
	char nazionalita[46];

  //output
	int codice;
```

```
// Get the required information
printf("\nUsername: ");
getInput(46, username, false);
printf("Password: ");
getInput(46, pass, false);
printf("Nome: ");
getInput(46, nome, false);
printf("Indirizzo: ");
getInput(46, indirizzo, false);
printf("Nazionalita: ");
getInput(46, nazionalita, false);
// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call iscrivi_insegnante(?, ?, ?, ?, ?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize teacher's registration statement\n", false);
}

// Prepare parameters
memset(param, 0, sizeof(param));
param[0].buffer_type =  MYSQL_TYPE_VAR_STRING;
param[0].buffer = username;
param[0].buffer_length = strlen(username);

param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[1].buffer = pass;
param[1].buffer_length = strlen(pass);
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = nome;
param[2].buffer_length = strlen(nome);
param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
param[3].buffer = indirizzo;
param[3].buffer_length = strlen(indirizzo);
param[4].buffer_type =  MYSQL_TYPE_VAR_STRING;
param[4].buffer = nazionalita;
param[4].buffer_length = strlen(nazionalita);
param[5].buffer_type = MYSQL_TYPE_LONG; // OUT
param[5].buffer = &codice;
param[5].buffer_length = sizeof(codice);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for teacher's registration\n", true);
}
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while teacher's registration.");
        goto out;
}
// Get back the ID of the newly-added student
memset(param, 0, sizeof(param));
```

```
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[0].buffer = &codice;
        param[0].buffer_length = sizeof(codice);
        if(mysql_stmt_bind_result(prepared_stmt, param)) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
        }
        // Retrieve output parameter
        if(mysql_stmt_fetch(prepared_stmt)) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
        }
        printf("Teacher's registration correctly added with ID %d...\n", codice);
   out:
        mysql_stmt_close(prepared_stmt);
}
static void aggiungi_lezioneprivata(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[5];

        // Input
   MYSQL_TIME inizio_time = {};
   char  inizio[21];
   MYSQL_TIME fine_time = {};
   char  fine[21];
```

```
    int insegnante_int;
    char insegnante[46];
    int allievo_int;
    char allievo[46];

    //Output of the stored procedure
    int codice;

    // Print generic information useful for the function
      if(!setup_prepared_stmt(&prepared_stmt, "Select codice, username, nome from insegnante", conn)) {
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of teachers statement\n", false);
        }
// Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error(prepared_stmt, "An error occurred while printing teachers.");
            goto out;
        }
    dump_result_set(conn, prepared_stmt, "\nTeachers: ");
    mysql_stmt_next_result(prepared_stmt);
    mysql_stmt_close(prepared_stmt);

      if(!setup_prepared_stmt(&prepared_stmt, "Select matricola, username, nome, cognome from allievo", conn)) {
```

```
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of students statement\n", false);
        }
// Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error(prepared_stmt, "An error occurred while printing students.");
            goto out;
        }
        dump_result_set(conn, prepared_stmt, "\nStudents: ");
        mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);

        // Get the required information
        printf("\nBegin (YYYY/MM/DD hh:mm): ");
        getInput(46, inizio, false);
        printf("End (YYYY/MM/DD hh:mm): ");
        getInput(46, fine, false);
        printf("Teacher: ");
        getInput(46, insegnante, false);
        printf("Student: ");
        getInput(46, allievo, false);
        //cast char to int AND convert char to MYSQL_TIME
        insegnante_int = atoi(insegnante);
   allievo_int = atoi(allievo);
   if(convert_timestamp(inizio, &inizio_time) == -1){
```

```
        fprintf(stderr, "Could not parse the begin private lesson string\n");
        abort();
    }
    if(convert_timestamp(fine, &fine_time) == -1){
        fprintf(stderr, "Could not parse the end private lesson string\n");
        abort();
    }
            // Prepare stored procedure call
            if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_lezioneprivata(?, ?, ?, ?, ?)", conn))
{
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize private lesson insertion
statement\n", false);
            }
            // Prepare parameters
            memset(param, 0, sizeof(param));
            param[0].buffer_type =  MYSQL_TYPE_TIMESTAMP;
            param[0].buffer = &inizio_time;
            param[0].buffer_length = sizeof(inizio_time);
            param[1].buffer_type = MYSQL_TYPE_TIMESTAMP;
            param[1].buffer = &fine_time;
            param[1].buffer_length = sizeof(fine_time);
            param[2].buffer_type = MYSQL_TYPE_LONG;
            param[2].buffer = &insegnante_int;
            param[2].buffer_length = sizeof(insegnante_int);
            param[3].buffer_type =  MYSQL_TYPE_LONG;
```

```
param[3].buffer = &allievo_int;
param[3].buffer_length = sizeof(allievo_int);
param[4].buffer_type = MYSQL_TYPE_LONG; // OUT
param[4].buffer = &codice;
param[4].buffer_length = sizeof(codice);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for private lesson insertion\n", true);
}
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
    print_stmt_error(prepared_stmt, "An error occurred while adding the private lesson.");
    goto out;
}
// Get back the ID of the newly-added student
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
param[0].buffer = &codice;
param[0].buffer_length = sizeof(codice);
if(mysql_stmt_bind_result(prepared_stmt, param)) {
    finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
}
// Retrieve output parameter
```

```
        if(mysql_stmt_fetch(prepared_stmt)) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
        }
        printf("Private lesson correctly added with ID %d...\n", codice);
   out:
        mysql_stmt_close(prepared_stmt);
}

static void crea_attivita(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[8];
        //print corsi, insegnqanti, allievi; IMPORTANTE
        // Input
        MYSQL_TIME inizio_time = {};
        char  inizio[21];
        MYSQL_TIME fine_time = {};
        char  fine[21];
        int tipo_int = 0;
        char regista[46];
        char film[46];
        char argomento[46];
        char conferenziere[46];
        char options[5] = {'1','2'};
```

```
char op;
//Output of the stored procedure
int codice;

// Get activity type
printf("\n*** What type of activity? ***\n\n");
printf("1) Proiezione\n");
printf("2) Conferenza\n");
op = multiChoice("Select an option", options, 2);
switch(op) {
case '1':
        tipo_int = 1;
        printf("Director: ");
        getInput(46, regista, false);
        printf("Film: ");
        getInput(46, film, false);
        //set conference parameters null
        strcpy(argomento,"/");
        strcpy(conferenziere, "/");
        break;
case '2':
        tipo_int = 2;
printf("Sudject: ");
getInput(46, argomento, false);
```

```
        printf("Speaker: ");
        getInput(46, conferenziere, false);

        //set Proiezione parameters null
        strcpy(regista, "/");
        strcpy(film, "/");
        break;
    default:
        fprintf(stderr, "Could not parse the type string\n");
        abort();
        }
        printf("Begin (YYYY/MM/DD hh:mm): ");
        getInput(46, inizio, false);
        printf("End (YYYY/MM/DD hh:mm): ");
        getInput(46, fine, false);
        //convert char to MYSQL_TIME
        if(convert_timestamp(inizio, &inizio_time) == -1){
        fprintf(stderr, "Could not parse the begin activity string\n");
        abort();
}
if(convert_timestamp(fine, &fine_time) == -1){
    fprintf(stderr, "Could not parse the end activity string\n");
    abort();
}
```

```
// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call crea_attivita(?, ?, ?, ?, ?, ?, ?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize activity insertion statement\n", false);
}
// Prepare parameters
memset(param, 0, sizeof(param));
param[0].buffer_type =  MYSQL_TYPE_TIMESTAMP;
param[0].buffer = &inizio_time;
param[0].buffer_length = sizeof(inizio_time);
param[1].buffer_type = MYSQL_TYPE_TIMESTAMP;
param[1].buffer = &fine_time;
param[1].buffer_length = sizeof(fine_time);
param[2].buffer_type = MYSQL_TYPE_LONG;
param[2].buffer = &tipo_int;
param[2].buffer_length = sizeof(tipo_int);
param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
param[3].buffer = regista;
param[3].buffer_length = strlen(regista);
param[4].buffer_type =  MYSQL_TYPE_VAR_STRING;
param[4].buffer = film;
param[4].buffer_length = strlen(film);
param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
param[5].buffer = argomento;
param[5].buffer_length = strlen(argomento);
```

```
param[6].buffer_type =  MYSQL_TYPE_VAR_STRING;
param[6].buffer = conferenziere;
param[6].buffer_length = strlen(conferenziere);
param[7].buffer_type = MYSQL_TYPE_LONG; // OUT
param[7].buffer = &codice;
param[7].buffer_length = sizeof(codice);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for activity insertion\n", true);
}
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while adding the activity.");
        goto out;
}
// Get back the ID of the newly-added student
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
param[0].buffer = &codice;
param[0].buffer_length = sizeof(codice);
if(mysql_stmt_bind_result(prepared_stmt, param)) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
}
// Retrieve output parameter
```

```
        if(mysql_stmt_fetch(prepared_stmt)) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
        }
        printf("Activity correctly added with ID %d...\n", codice);
   out:
        mysql_stmt_close(prepared_stmt);
}

static void crea_corso(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[2];
        // Input
        char livello[46];
        //Output of the stored procedure
        int codice;
        // Print generic information useful for the function
        if(!setup_prepared_stmt(&prepared_stmt, "Select nome from livello", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of levels statement\n", false);
        }


        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
```

```
            print_stmt_error(prepared_stmt, "An error occurred while printing levels.");
            goto out;
        }
        dump_result_set(conn, prepared_stmt, "\nLEVELS: ");
        mysql_stmt_next_result(prepared_stmt);
        mysql_stmt_close(prepared_stmt);
        // Get the required information
        printf("\nLevel: ");
        getInput(46, livello, false);
        // Prepare stored procedure call
        if(!setup_prepared_stmt(&prepared_stmt, "call crea_corso(?, ?)", conn)) {
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize course insertion statement\n", false);
        }
        // Prepare parameters
        memset(param, 0, sizeof(param));
        param[0].buffer_type =  MYSQL_TYPE_VAR_STRING;
        param[0].buffer = livello;
        param[0].buffer_length = strlen(livello);
        param[1].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[1].buffer = &codice;
        param[1].buffer_length = sizeof(codice);

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
```

```
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for course insertion\n", true);
        }
        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error(prepared_stmt, "An error occurred while adding the course.");
            goto out;
        }
        // Get back the ID of the newly-added student
        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[0].buffer = &codice;
        param[0].buffer_length = sizeof(codice);
        if(mysql_stmt_bind_result(prepared_stmt, param)) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
        }
        // Retrieve output parameter
        if(mysql_stmt_fetch(prepared_stmt)) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
        }
        printf("Course correctly added with ID %d...\n", codice);
    out:
        mysql_stmt_close(prepared_stmt);
}
```

```
static void aggiungi_lezione(MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[6];
	// Input
	int corso_int;
	char corso[46];
	char livello[46];
	MYSQL_TIME inizio_time= {};
	char  inizio[21];
	MYSQL_TIME fine_time= {};
	char  fine[21];
	int insegnante_int;
	char insegnante[46];
	//Output of the stored procedure
	int numero;

	// Print generic information useful for the function
	if(!setup_prepared_stmt(&prepared_stmt, "Select * from corso", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of courses statement\n", false);
	}

	// Run procedure
```

```
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while printing courses.");
        goto out;
}
dump_result_set(conn, prepared_stmt, "\nCourses: ");
mysql_stmt_next_result(prepared_stmt);
mysql_stmt_close(prepared_stmt);

 if(!setup_prepared_stmt(&prepared_stmt, "Select codice, username, nome from insegnante", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of teachers statement\n", false);
}

// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while printing teachers.");
        goto out;
}
dump_result_set(conn, prepared_stmt, "\nTeachers: ");
mysql_stmt_next_result(prepared_stmt);
mysql_stmt_close(prepared_stmt);



// Get the required information
```

```
printf("\nCourse: ");
getInput(46, corso, false);
printf("Level: ");
getInput(46, livello, false);
printf("Teacher: ");
getInput(46, insegnante, false);
printf("Begin (YYYY/MM/DD hh:mm): ");
getInput(46, inizio, false);
printf("End (YYYY/MM/DD hh:mm): ");
getInput(46, fine, false);

//cast char to int AND cast char to MYSQL_TIME
corso_int = atoi(corso);
insegnante_int = atoi(insegnante);
if(convert_timestamp(inizio, &inizio_time) == -1){
        fprintf(stderr, "Could not parse the begin lesson string\n");
        abort();
}
if(convert_timestamp(fine, &fine_time) == -1){
        fprintf(stderr, "Could not parse the end lesson string\n");
        abort();
}

// Prepare stored procedure call
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_lezione(?, ?, ?, ?, ?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize lesson insertion statement\n", false);
}
// Prepare parameters
memset(param, 0, sizeof(param));
param[0].buffer_type =  MYSQL_TYPE_LONG;
param[0].buffer = &corso_int;
param[0].buffer_length = sizeof(corso_int);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = livello;
param[1].buffer_length = strlen(livello);
param[2].buffer_type =  MYSQL_TYPE_TIMESTAMP;
param[2].buffer = &inizio_time;
param[2].buffer_length = sizeof(inizio_time);
param[3].buffer_type = MYSQL_TYPE_TIMESTAMP;
param[3].buffer = &fine_time;
param[3].buffer_length = sizeof(fine_time);
param[4].buffer_type =  MYSQL_TYPE_LONG;
param[4].buffer = &insegnante_int;
param[4].buffer_length = sizeof(insegnante_int);
param[5].buffer_type = MYSQL_TYPE_LONG; // OUT
param[5].buffer = &numero;
param[5].buffer_length = sizeof(numero);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
```

```
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for lesson insertion\n", true);
        }
        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error(prepared_stmt, "An error occurred while adding the lesson.");
            goto out;
        }
        // Get back the ID of the newly-added student
        memset(param, 0, sizeof(param));
        param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
        param[0].buffer = &numero;
        param[0].buffer_length = sizeof(numero);
        if(mysql_stmt_bind_result(prepared_stmt, param)) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
        }
        // Retrieve output parameter
        if(mysql_stmt_fetch(prepared_stmt)) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
        }
        printf("Lesson correctly added with ID %d...\n", numero);
    out:
        mysql_stmt_close(prepared_stmt);
}
```

```
void run_as_amministratore(MYSQL *conn)
{
	char options[9] = {'1','2', '3', '4', '5', '6', '7', '8', '9'};
	char op;
	printf("Switching to administrative role...\n");
	if(!parse_config("users/amministratore.json", &conf)) {
		fprintf(stderr, "Unable to load administrator configuration\n");
		exit(EXIT_FAILURE);
	}
	if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
		fprintf(stderr, "mysql_change_user() failed\n");
		exit(EXIT_FAILURE);
	}
	while(true) {
		printf("\033[2J\033[H");
		printf("*** What should I do for you? ***\n\n");
		printf("1) Add lesson\n");
		printf("2) Add course\n");
		printf("3) Create activity\n");
		printf("4) Add private lesson\n");
		printf("5) Register new teacher\n");
		printf("6) Register new student\n");
		printf("7) Teacher's report\n");
```

```
printf("8) Start new scholastic year\n");
printf("9) Exit\n");
op = multiChoice("Select an option: ", options, 9);
switch(op) {
        case '1':
                aggiungi_lezione(conn);
                break;
        case '2':
                crea_corso(conn);
                break;
        case '3':
                crea_attivita(conn);
                break;
        case '4':
                aggiungi_lezioneprivata(conn);
                break;
        case '5':
                registra_insegnante(conn);
                break;
        case '6':
                registra_allievo(conn);
                break;
        case '7':
                report_amministratore(conn);
```

```
                        break;
                case '8':
                        reset_anno(conn);
                        break;
                case '9':
                        return;
                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
        }
        getchar();
    }
}
```

Segreteria.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "defines.h"

static void partecipa_attivita(MYSQL *conn)
{
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];
```

```
// Input
int allivo_int;
char allievo[46];
int attivita_int;
char attivita[46];
// Print generic information useful for the function
if(!setup_prepared_stmt(&prepared_stmt, "Select codice, tipo, inizio, fine from attivita", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize the printing of activities statement\n", false);
}
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while printing activities.");
        goto out;
}

dump_result_set(conn, prepared_stmt, "\nActivities: ");
mysql_stmt_next_result(prepared_stmt);
mysql_stmt_close(prepared_stmt);

if(!setup_prepared_stmt(&prepared_stmt, "select matricola, username, nome, cognome from allievo", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of students statement\n", false);
}
```

```
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while printing students.");
        goto out;
}

dump_result_set(conn, prepared_stmt, "\nStudents: ");
mysql_stmt_next_result(prepared_stmt);
mysql_stmt_close(prepared_stmt);
// Get the required information
printf("\nStudent: ");
getInput(46, allievo, false);
printf("Activity: ");
getInput(46, attivita, false);
allievo_int = atoi(allievo);
attivita_int = atoi(attivita);
// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call partecipa_attivita(?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize activity partecipation statement\n", false);
}
// Prepare parameters
memset(param, 0, sizeof(param));
param[0].buffer_type =  MYSQL_TYPE_LONG;
param[0].buffer = &allievo_int;
```

```
        param[0].buffer_length = sizeof(allievo_int);
        param[1].buffer_type =  MYSQL_TYPE_LONG;
        param[1].buffer = &attivita_int;
        param[1].buffer_length = sizeof(attivita_int);
        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
                finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for partecipation insertion\n", true);
        }
        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while adding the partecipation.");
                goto out;
        }
        printf("Partecipation correctly added...\n");
   out:
        mysql_stmt_close(prepared_stmt);
}

static void aggiungi_assenza(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[4];
        // Input for the registration routine
        int corso_int;
   char corso[46];
```

```
char livello[46];
    int numero_int;
    char numero[46];
    int allievo_int;
    char allievo[46];

     //View courses
    if(!setup_prepared_stmt(&prepared_stmt, "Select * from lezione", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of lessons statement\n", false);
            }
            // Run procedure
            if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while printing the lessons.");
                goto out;
            }

    dump_result_set(conn, prepared_stmt, "\nLessons: ");
    mysql_stmt_next_result(prepared_stmt);
    mysql_stmt_close(prepared_stmt);
    //View courses
       if(!setup_prepared_stmt(&prepared_stmt, "Select matricola, username, nome, cognome from allievo", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of students statement\n", false);
```

```
}
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while printing the students.");
        goto out;
}
dump_result_set(conn, prepared_stmt, "\nStudents: ");
mysql_stmt_next_result(prepared_stmt);
mysql_stmt_close(prepared_stmt);

// Get the required information
printf("\nCourse: ");
getInput(46, corso, false);
printf("Level: ");
getInput(46, livello, false);
printf("Number of lesson: ");
getInput(46, numero, false);
printf("Student: ");
getInput(46, allievo, false);

corso_int = atoi(corso);
numero_int = atoi(numero);
allievo_int = atoi(allievo);
// Prepare stored procedure call
```

```
    if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_assenza(?, ?, ?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize absence insertion statement\n", false);
    }
    // Prepare parameters
    memset(param, 0, sizeof(param));
    param[0].buffer_type =  MYSQL_TYPE_LONG;
    param[0].buffer = &corso_int;
    param[0].buffer_length = sizeof(corso_int);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = livello;
    param[1].buffer_length = strlen(livello);
    param[2].buffer_type =  MYSQL_TYPE_LONG;
    param[2].buffer = &numero_int;
    param[2].buffer_length = sizeof(numero_int);
    param[3].buffer_type = MYSQL_TYPE_LONG;
    param[3].buffer = &allievo_int;
    param[3].buffer_length = sizeof(allievo_int);
    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for absence insertion\n", true);
    }



    // Run procedure
```

```
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while adding the absence.");
                goto out;
        }
        printf("Absence correctly added...\n");
   out:
        mysql_stmt_close(prepared_stmt);
}

static void registra_studente_corso(MYSQL *conn)
{
        MYSQL_STMT *prepared_stmt;
        MYSQL_BIND param[3];
        // Input for the registration routine
        int corso_int;
        char corso[46];
        char livello[46];
        int allievo_int;
        char allievo[46];
        //View courses
        if(!setup_prepared_stmt(&prepared_stmt, "Select * from corso", conn)) {
                finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of courses statement\n", false);
        }
        // Run procedure
```

```
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while printing of courses.");
        goto out;
}

dump_result_set(conn, prepared_stmt, "\nCourses: ");
mysql_stmt_next_result(prepared_stmt);
mysql_stmt_close(prepared_stmt);
//View courses
if(!setup_prepared_stmt(&prepared_stmt, "select matricola, username, nome, cognome from allievo", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of students statement\n", false);
}
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error(prepared_stmt, "An error occurred while printing of students.");
        goto out;
}
dump_result_set(conn, prepared_stmt, "\nStudents: ");
mysql_stmt_next_result(prepared_stmt);
mysql_stmt_close(prepared_stmt);



// Get the required information
```

```
printf("\nCourse: ");
getInput(46, corso, false);
printf("Level: ");
getInput(46, livello, false);
printf("Student: ");
getInput(46, allievo, false);
corso_int = atoi(corso);
allievo_int = atoi(allievo);
// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call iscrivi_allievo_corso(?, ?, ?)", conn)) {
    finish_with_stmt_error(conn, prepared_stmt, "Unable to register student course statement\n", false);
}
// Prepare parameters
memset(param, 0, sizeof(param));
param[0].buffer_type =  MYSQL_TYPE_LONG;
param[0].buffer = &corso_int;
param[0].buffer_length = sizeof(corso_int);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = livello;
param[1].buffer_length = strlen(livello);
param[2].buffer_type =  MYSQL_TYPE_LONG;
param[2].buffer = &allievo_int;
param[2].buffer_length = sizeof(allievo_int);
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
```

```
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for register student course\n", true);
        }
        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
            print_stmt_error(prepared_stmt, "An error occurred while register student course.");
            goto out;
        }
        printf("Registration student course correctly executed\n");
    out:
        mysql_stmt_close(prepared_stmt);
}

void run_as_segreteria(MYSQL *conn)
{
        char options[4] = {'1','2', '3', '4'};
        char op;
        printf("Switching to Secretariat role...\n");
        if(!parse_config("users/segreteria.json", &conf)) {
            fprintf(stderr, "Unable to load segreteria configuration\n");
            exit(EXIT_FAILURE);
        }
        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
            fprintf(stderr, "mysql_change_user() failed\n");
            exit(EXIT_FAILURE);
```

```
}
while(true) {
        printf("\033[2J\033[H");
        printf("*** What should I do for you? ***\n\n");
        printf("1) Register student course\n");
        printf("2) Add student's absence\n");
        printf("3) Partecipate student to activity\n");
        printf("4) Exit\n");
        op = multiChoice("Select an option: ", options, 4);
        switch(op) {
                case '1':
                        registra_studente_corso(conn);
                        break;
                case '2':
                        aggiungi_assenza(conn);
                        break;
                case '3':
                        partecipa_attivita(conn);
                        break;
                case '4':
                        return;
                default:
                        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                        abort();
```

```
                }
                getchar();
        }
}
```

Insegnante.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "defines.h"
struct insegn{
   int codice;
   char username[46];
   char nome[46];
   char indirizzo[46];
   char nazionalita[46];
};

static size_t func_ins(MYSQL *conn, MYSQL_STMT *stmt, struct insegn ** ins)
{
        int status;
        size_t row = 0;
        MYSQL_BIND param[5];
```

```
int codice;
char username[46];
char nome[46];
char indirizzo[46];
char nazionalita[46];
my_bool is_null1, is_null2;

if (mysql_stmt_store_result(stmt)) {
        fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
        exit(0);
}
*ins = malloc(mysql_stmt_num_rows(stmt) * sizeof(struct insegn));
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &codice;
param[0].buffer_length = sizeof(codice);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = username;
param[1].buffer_length = 46;
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = nome;
param[2].buffer_length = 46;
param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
    param[3].buffer = indirizzo;
    param[3].buffer_length = 46;
    param[3].is_null = &is_null1;
    param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[4].buffer = nazionalita;
    param[4].buffer_length = 46;
    param[4].is_null = &is_null2;
    if(mysql_stmt_bind_result(stmt, param)) {
            finish_with_stmt_error(conn, stmt, "Unable to bind column parameters\n", true);
    }
    /* assemble course general information */
    while (true) {
            status = mysql_stmt_fetch(stmt);
            if (status == 1 || status == MYSQL_NO_DATA){
                    break;
}
(*ins)[row].codice = codice;
strcpy((*ins)[row].username, username);
strcpy((*ins)[row].nome, nome);
strcpy((*ins)[row].indirizzo, indirizzo);
strcpy((*ins)[row].nazionalita, nazionalita);
            row++;
    }
    return row;
```

```
}

static void report_insegnante(MYSQL *conn) {
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];
	int status;
	struct insegn* ins;
	bool first = true;
	int index = 0;
	char header[512];
	// Prepare stored procedure call
	if(!setup_prepared_stmt(&prepared_stmt, "call report_insegnante(?)", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize career report statement\n", false);
	}

	// Prepare parameters
	memset(param, 0, sizeof(param));
	param[0].buffer_type =  MYSQL_TYPE_VAR_STRING;
	param[0].buffer = conf.username;
	param[0].buffer_length = strlen(conf.username);
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for report\n", true);
	}
```

```
        // Run procedure
        if (mysql_stmt_execute(prepared_stmt) != 0) {
                print_stmt_error(prepared_stmt, "An error occurred while retrieving the career report.");
                goto out;
        }
        // We have multiple result sets here!
        printf("Teacher Report\n");
        do {
                // Skip OUT variables (although they are not present in the procedure...)
                if(conn->server_status & SERVER_PS_OUT_PARAMS) {
                        goto next;
                }
                if(first) {
                        func_ins(conn, prepared_stmt, &ins);
                        first = false;
                } else {
                        sprintf(header, "\nTeacher code: %d\nUsername: %s\nName: %s\nAddress: %s\nNazionality: %s\n", ins[index].codice, ins[index].username, ins[index].nome, ins[index].indirizzo, ins[index].nazionalita);
                        dump_result_set(conn, prepared_stmt, header);
                        index++;
                }

          // more results? -1 = no, >0 = error, 0 = yes (keep looking)
```

```
        next:
                status = mysql_stmt_next_result(prepared_stmt);
                if (status > 0)
                        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);

        } while (status == 0);

    out:
        mysql_stmt_close(prepared_stmt);
}

void run_as_insegnante(MYSQL *conn)
{
        char options[2] = {'1','2'};
        char op;
        printf("Switching to Teacher role...\n");
        if(!parse_config("users/insegnante.json", &conf)) {
                fprintf(stderr, "Unable to load segreteria configuration\n");
                exit(EXIT_FAILURE);
        }
        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }
```

```
        while(true) {
                printf("\033[2J\033[H");
                printf("*** What should I do for you? ***\n\n");
                printf("1) Get weekly report\n");
                printf("2) Exit\n");
                op = multiChoice("Select an option: ", options, 2);
                switch(op) {
                        case '1':
                                report_insegnante(conn);
                                break;
                        case '2':
                                return;
                        default:
                                fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
                                abort();
                }
                getchar();
        }
}
```

Allievo.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include "defines.h"
static void print_info(MYSQL *conn)
{
	MYSQL_STMT *prepared_stmt;
	MYSQL_BIND param[1];

	if(!setup_prepared_stmt(&prepared_stmt, "Select matricola, username, nome, cognome from allievo where username = ?", conn)) {
		finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize printing of student statement\n", false);
	}
	// Prepare parameters
	memset(param, 0, sizeof(param));
	param[0].buffer_type =  MYSQL_TYPE_VAR_STRING;
	param[0].buffer = conf.username;
	param[0].buffer_length = strlen(conf.username);
	if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
		finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for printing student\n", true);
	}
	// Run procedure
	if (mysql_stmt_execute(prepared_stmt) != 0) {
		print_stmt_error(prepared_stmt, "An error occurred while printing student.");
		goto out;
	}
```

```
        dump_result_set(conn, prepared_stmt, "\nStudent info: ");
        mysql_stmt_next_result(prepared_stmt);
        //close statement
   out:
      mysql_stmt_close(prepared_stmt);
}

void run_as_allievo(MYSQL *conn)
{
        //Get basical information about Student
        printf("Switching to Student role...\n");
        if(!parse_config("users/allievo.json", &conf)) {
                fprintf(stderr, "Unable to load student configuration\n");
                exit(EXIT_FAILURE);
        }
        if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
                fprintf(stderr, "mysql_change_user() failed\n");
                exit(EXIT_FAILURE);
        }
        print_info(conn);
        return;
}
```