



Basi di Dati

Progetto A.A. 2021/2022

# My English School

0266910

Simone Nicosanti

## Indice

# 1. Descrizione del Minimondo

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 |
|---|---|---|---|
| 27 | Lezione individuale | Lezione Privata | Risoluzione della sinomia |
| 22 | Recapito | Numero di telefono | Specificare la natura del recapito |
| 16 | Iscritto | Allievo | Risoluzione della sinomia |
| 34 | Iscriversi | Prenotarsi | Disambiguazione rispetto all'iscrizione ad un certo corso |
| 20 | corso | Lezione di Corso | Disambiguazione tra il corso e le sue lezioni che l'insegnant può fare |
| 24 | Giorni specifici | Date specifiche | Disambiguazione tra giorno della settimana in cui si tiene un corso e data data specifica in cui un allievo può essere assente |
| | | | |

**Specifica disambiguata**

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. Per il livello rappresentiamo il nome del livello stesso (ad esempio Elementary, Intermediate, First Certificate, Advanced, Proficiency), che lo identifica, 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 un corso rappresentiamo la data di attivazione, il numero degli allievi, le informazioni anagrafiche degli allievi, l'elenco dei giorni ed orari in cui è tenuto.

Per gli insegnanti rappresentiamo il nome, l'indirizzo, la nazione di provenienza, ed i corsi a cui sono stati assegnati. Un insegnante può essere assegnato a più di un corso, assicurandosi che in una

determinata fascia oraria un insegnante sia assegnato ad una sola lezione di corso.

Per gli allievi rappresentiamo il nome, un numero di telefono, il corso a cui sono iscritti, la data di iscrizione al corso e il numero di assenze fatte finora (è di interesse tenere traccia delle date specifiche 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 privata 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 rappresentiamo la data e l'ora in cui verrà tenuta. Nel caso di proiezioni in lingua originale, rappresentiamo il nome del film ed il nome del regista. Nel caso di conferenze, rappresentiamo 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.

## Glossario dei Termini

| Termine | Descrizione | Sinonimi | Collegamenti |
|---|---|---|---|
| Livello | Livello di un certo corso; può prevedere un esame | | Corso |
| Corso | Corso impartito all'interno della scuola | | Livello, Allievo, Insegnante |
| Insegnante | Individuo assegnato alla docenza di uno o più corsi all'interno della scauola | | Corso, Lezione Privata |
| Lezione di Corso | Lezione di un Corso | | Corso |

| | | | |
|---|---|---|---|
| | tenuto nella scuola | | |
| Allievo | Individuo iscritto ad un corso all'interno della scuola | | Corso, Lezione Privata, Assenza, Attività Culturale |
| Assenza | Assenza commessa da un certo allievo | | Allievo |
| Lezione Privata | Lezione prenotata da un allievo | | Allievo, Insegnante |
| Attività Culturale | Attività extra organizzata dalla scuola; possono essere proiezioni di film oppure conferenze | | Allievo |

## Raggruppamento dei requisiti in insiemi omogenei

| **Frasi relative a Livello** |
|---|
| Per il livello rappresentiamo il nome del livello stesso (ad esempio Elementary, Intermediate, First Certificate, Advanced, Proficiency), che lo identifica, il nome del libro di testo e se viene richiesto di sostenere un esame finale. |

| **Frasi relative a Corso** |
|---|
| I corsi sono organizzati per livelli.<br>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 un corso rappresentiamo la data di attivazione, il numero degli allievi, le informazioni anagrafiche degli allievi, l'elenco dei giorni ed orari in cui è tenuto. |

| **Frasi relative a Allievo** |
|---|
| Per gli allievi rappresentiamo il nome, un recapito, il corso a cui sono iscritti, la data di iscrizione al corso e il numero di assenze fatte finora. Gli allievi possono anche prenotare lezioni private... |

| **Frasi relative a Insegnante** |
|---|
| Per gli insegnanti rappresentiamo il nome, l'indirizzo, la nazione di provenienza, ed i corsi a cui sono stati assegnati. Un insegnante può essere assegnato più di un corso, assicurandosi che in una determinata fascia oraria un insegnante sia assegnato ad un solo corso. |

| **Frasi relative a Assenza** |
|---|
| è di interesse tenere traccia delle date specifiche in cui un allievo è stato assente |

| **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 privata non può avvenire in concomitanza di un altro impegno di un insegnante. |

| **Frasi relative a Attività Culturale** |
| --- |
| Ciascuna attività è identificata da un codice progressivo, e rappresentiamo la data e l'ora in cui verrà tenuta. Nel caso di proiezioni in lingua originale, rappresentiamo il nome del film ed il nome del regista. Nel caso di conferenze, rappresentiamo l'argomento che verrà trattato ed il nome del conferenziere. Per poter partecipare alle attività gli allievi devono iscriversi. |

# 3. Progettazione concettuale

## Costruzione dello schema E-R

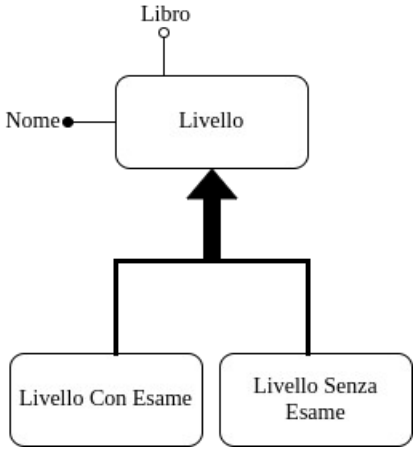



*Figura 1: Livello*

In fig.1 si mostra la costruzione dell'entità Livello che rappresenta il livello a cui un corso afferisce; come indicato da specifica, si rappresentano il nome del livello, identificante per il livello stesso e si rappresenta il nome del libro di testo.
Per rappresentare la differenza tra un livello che prevede un esame e livello che non lo prevede, si è costruita una generalizzazione con le entità "Livello Con Esame" e "Livello Senza Esame": per queste specializzazioni non vi sono proprietà significative da rappresentare.

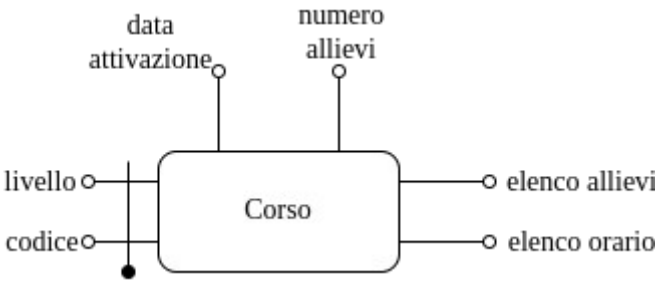



*Figura 2: Corso*

In fig.2 si mostra la costruzione dell'entità Corso che rappresenta un corso all'interno della scuola. Il corso è identificato dal nome del livello a cui afferisce e da un codice progressivo; per esso vengono

poi rappresentate la data in cui il corso è attivato, il numero di allievi, l'elenco degli allievi e l'elenco degli orari in cui le lezioni del corso sono svolte.

Partendo da questo si può notare che:

- Il Livello è un concetto significativo all'interno del Minimondo che è già rappresentato da un'entità (vedere fig.1): per cui si può rendere Corso un' Entità debole identificata esternamente dal Livello;
- Dovendo per gli Allievi rappresentare varie informazioni di anagrafica, possiamo reificare l'elenco e costruire una relazione tra Allievo e Corso;
- Possiamo considerare un Corso come costituito dalle lezioni del suo orario e quindi possiamo applicare il Pattern di Composizione.

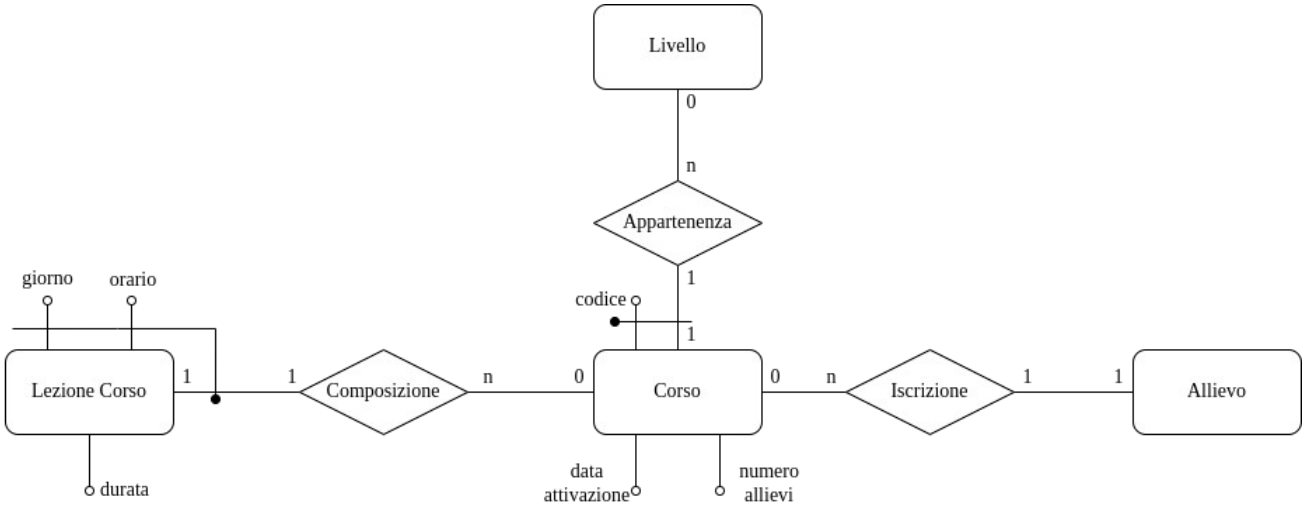



*Figura 3: Corso Modificato*

Dalle considerazioni di cui sopra otteniamo quindi le entità e relazioni in fig.3.

- Una "Lezione Corso" è identificata dal corso di cui è lezione (come indicato dal Pattern di Composizione), dal giorno e dall'orario in cui si tiene (questo permette che vi possano essere più lezioni dello stesso corso nello stesso giorno) ;
- Un Corso può avere un unico Livello (d'altronde il Livello è identificante), mentre un livello può avere 0 (qualora in un certo anno scolastico non siano stati avviati corsi per un certo livello) o molti corsi associati.
- Un Corso può avere da 0 (qualora ad un corso non si siano ancora mai iscritti degli allievi) o molti Allievi; da specifica, un Allievo è iscritto ad un unico Corso. E da qui la cardinalità (1,1)

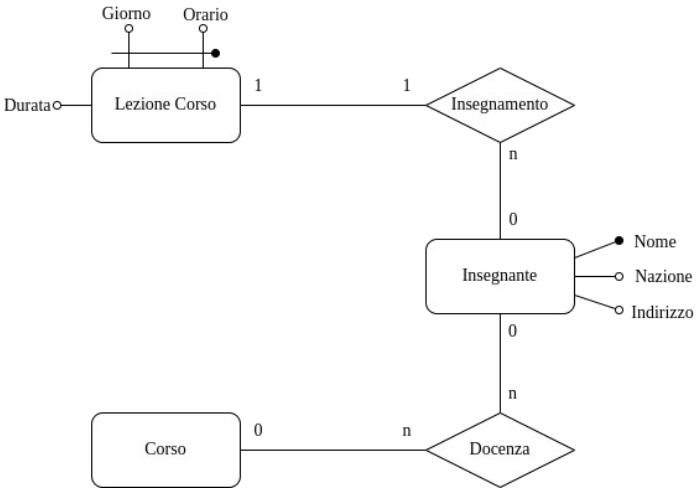



*Figura 4: Insegnante*

In fig.4 mostriamo la costruzione dell'entità Insegnante. Per l'insegnante rappresentiamo:

- Nome. Supponiamo che, data la dimensione del Minimondo di riferimento, la probabilità che vi siano due insegnanti con lo stesso nome sia bassa e che quindi si possa assumere il nome dell'insegnante come identificante per l'entità
- Nazione di provenienza
- Indirizzo

Per le associazioni abbiamo che:

- Un Insegnante può avere la Docenza di 0 (nel momento in cui non è stato ancora assegnato a nessun corso) o n Corsi ; viceversa un Corso può avere molteplici insegnanti
- Un Insegnante può essere assegnato a 0 o n lezioni di un Corso; una lezione è insegnata da un unico Insegnante. L'Insegnante deve essere in relazione con Lezione Corso perché da specifica "...in una determinata fascia oraria un insegnante sia assegnato ad un solo corso.", quindi è necessario tenere traccia di quale Insegnante è assegnato ad una certa lezione; d'altronde, per poter individuare delle sovrapposizioni negli impegni dell'insegnate, è necessario tenere in considerazione la fascia oraria e quindi si introduce un attributo durata alla lezione del corso, che permette proprio di tracciare la fascia oraria dell'impegno.

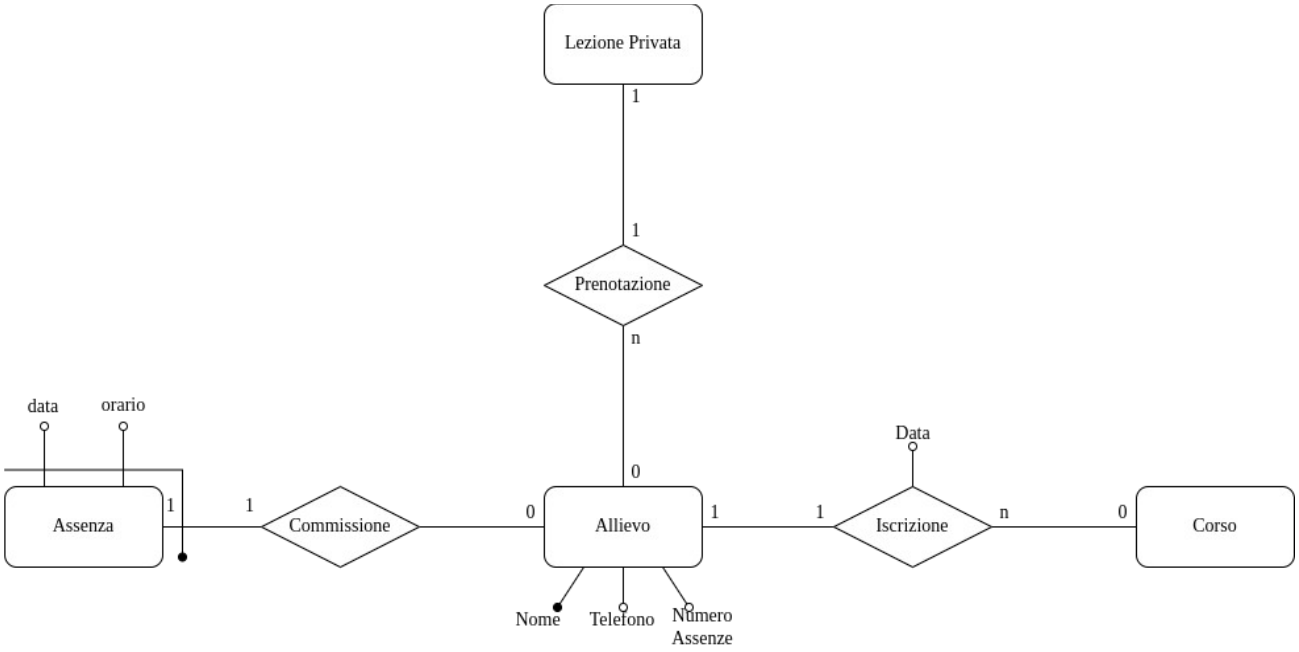



*Figura 5: Allievo*

In fig.5 rappresentiamo l'entità Allievo consideriamo:

- Nome. Come nel caso dell'Insegnante, supponiamo che, per la dimensione del minimondo di riferimento, il nome dell'Allievo sia sufficiente ad identificarlo
- Telefono e Numero di assenze

Rappresentiamo poi:

- Corso a cui l'allievo è iscritto tramite l'associazione Iscrizione. Viene richiesto inoltre anche di tenere traccia della data in cui l'allievo si è iscritto al corso: data la semantica di questa informazione, si nota che essa non afferisce in modo specifico all'allievo, bensì al modo con cui l'allievo si iscrive al corso e quindi è stato introdotto un attributo "data" alla relazione iscrizione;
- Lezioni private prenotate. L'Allievo ha la possibilità di prenotare 0 o più lezioni private e questa possibilità è esplicitata nella relazione prenotazione tra le due entità
- Assenze. Da specifica è di interesse tenere traccia delle assenze fatte dall'allievo e in particolare della loro data e si introduce quindi l'entità Assenza. Tuttavia, avendo considerato la possibilità che in uno stesso giorno possano esserci più lezioni dello stesso corso a cui l'Allievo è iscritto, consideriamo questo scenario: l'Allievo Simone fa assenza ad una certa lezione il 22-03-2022 e il suo corso, per quel giorno, prevede due lezioni; se Simone facesse assenza anche alla seconda lezione, questa informazione non potrebbe essere catturata perché sarebbe già presente nell'insime Assenza un'occorrenza per quel giorno. Tenuto conto di ciò,

aggiungiamo l'attributo orario all'Assenza, per distinguere le assenze che lo stesso Allievo può fare nello stesso giorno.

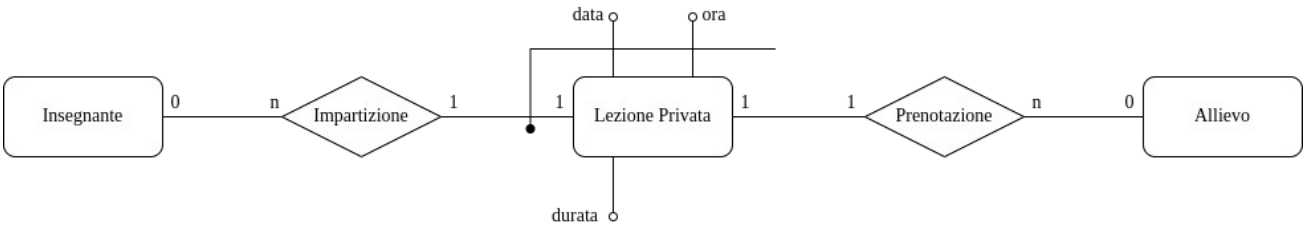



*Figura 6: Lezione Privata*

In fig.7 viene mostrata la costruzione dell'entità Lezione Privata. Per essa si mantengono:

- Data
- Ora e durata. L'ora e la durata devono essere espresse per tenere in considerazione le fasce orarie di svolgimento della lezione. Le considerazioni da questo punto di vista sono analoghe a quelle fatte per la Lezione Corso.
- Allievo che prenota la lezione tramite l'associazione prenotazione:
- Insegnante che impartisce la lezione tramite la relazione impartizione

Come identificatore scegliamo:

- data
- ora
- insegnante. Data e Ora non sonosufficienti ad identificare in modo univoco la lezione privata visto che ci possono essere più lezioni prenotate per la stessa data alla stessa ora. D'altrocanto però, da specifica, un'insegnante può avere un'unica lezione privata in una certa fascia oraria e quindi possiamo far identificare esternamente la lezione dall'insegnante che la impartisce.

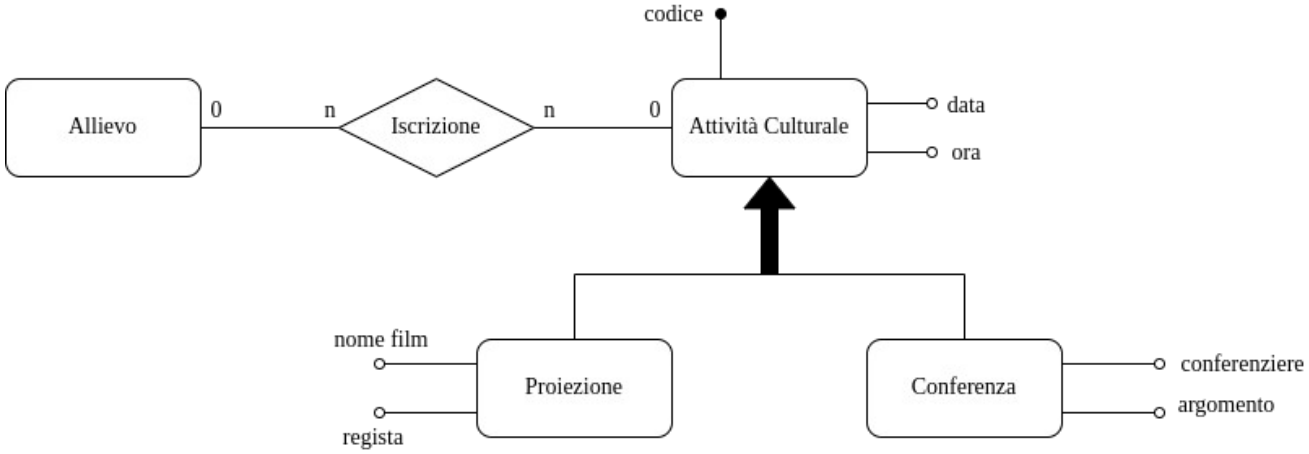



*Figura 7: Attività Culturale*

In fig.8 si mostra la costruzione dell'entità Attività Culturale. Le attività culturali si dividono in due tipologie:

- Proiezioni
- Conferenze

Questa suddivisione viene rappresentata attraverso una generalizzazione totale dell'entità.

La possibilità che gli allievi hanno di iscriversi ad un'Attività Culturale è rappresentata dalla relazione iscrizione tra le due entità:

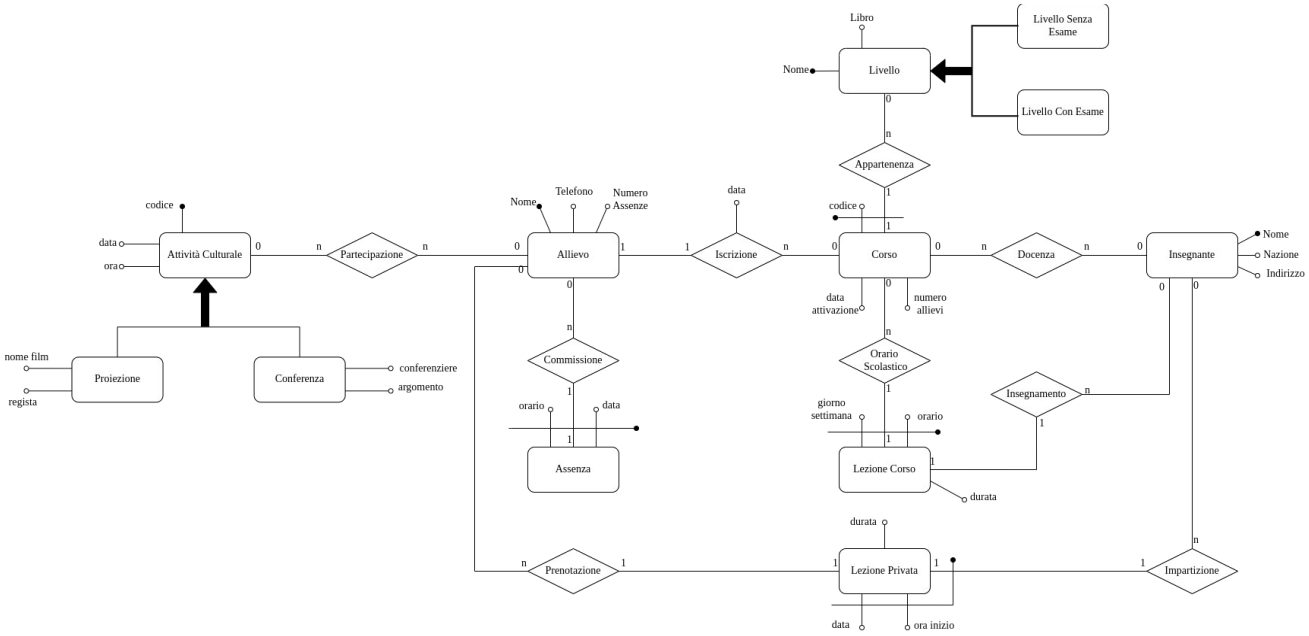
- un allievo può iscriversi a 0 o n attività culturali ;
- un'attività culturale può avere 0 o n allievi iscritti.

**Integrazione finale**

Si risolve il nome dell'associazione Iscrizione tra Allievo e Attività Culturale in Partecipazione, per evitare conflitto con l'Associazione Iscrizione tra Allievo e Corso.

Per usare un nome più significativo, si sostituisce l'associazione Composizione tra Corso e Lezione Corso con Orario Scolastico.

Per usare un nome più significativo si sostituisce l'attributo "giorno" di Lezione Corso con "giorno settimana".




## Regole aziendali

- Insegnante
  1. Un insegnante NON DEVE avere più di una Lezione (di Corso o Privata) assegnata nella stessa fascia oraria
  2. Un Insegnante DEVE insegnare Lezioni Corso solo di un Corso di cui è docente
- Lezione Privata
  1. Una Lezione Privata NON DEVE collidere con un altro impegno (di un corso o Privata) dell'Insegnante che la impartisce
- Corso
  1. Un Corso NON DEVE avere Lezioni Corso nello stesso giorno e in orari che collidono
- Assenza
  1. Un'Assenza DEVE avere una data il cui giorno della settimana coincide con quello di una Lezione Corso del Corso a cui l'Allievo è iscritto.
  2. Un'Assenza DEVE avere un orario che coincide con quello di una Lezione Corso a cui l'Allievo è Iscritto.
- Allievo
  1. Un Allievo NON DEVE prenotare Lezioni Private diverse in fasce orarie che collidono

## Dizionario dei dati

| Entità | Descrizione | Attributi | Identificatori |
|---|---|---|---|
| Livello | Livello dei corsi della scuola | Nome livello,<br>Libro | Nome livello |
| Livello con Esame | Livello che prevede un esame finale | (ereditati da Livello) | Nome Livello |
| Livello senza Esame | Livello che non prevede un esame finale | (ereditati da Livello) | Nome Livello |
| Corso | Corso attivato all'interno della scuola | Codice,<br>data attivazione,<br>numero allievi, | Codice,<br>Livello |
| Allievo | Allievo che è iscritto ad un corso all'interno della scuola | Nome,<br>telefono,<br>numero assenze | nome |
| Insegnante | Insegnante che ha la docenza di uno o più corsi all'interno della scuola | Nome,<br>nazione,<br>indirizzo | nome |
| Lezione Corso | Lezione del corso che si tiene in un certo giorno della settimana e che ha un certo orario di inizio e una certa durata(che ne definiscono la fascia oraria) | Giorno settimana,<br>ora di inizio,<br>durata | Giorno settimana,<br>ora di inizio,<br>*Corso* |
| Assenza | Assenza commessa da un allievo | Data, orario | Data,<br>Orario,<br>*Allievo* |
| Lezione Privata | Lezione Privata prenotata da un allievo | Data,<br>ora inizio,<br>durata | Data,<br>ora inizio,<br>durata,<br>*Insegnante* |
| Attività Culturale | Attività culturale organizzata dalla scuola | Codice,<br>data,<br>ora | codice |
| Proiezione | Proiezione organizzata dalla scuola | (ereditati da Attività Culurale),<br>film,<br>regista | codice |
| Conferenza | Conferenza organizzata dalla scuola | (ereditati da Attività Culurale),<br>argomento,<br>conferenziere | codice |
| | | | |

| Relazione | Descrizione | Entità Coinvolte | Attributi |
|---|---|---|---|
| Appartenenza | Associa un Corso al suo Livello | Corso (1,1)<br>Livello (0,n) | |
| Docenza | Associa un Insegnante ai corsi a cui è assegnato | Corso (0,n),<br>Insegnate (0,n) | |
| Iscrizione | Associa un Allievo al Corso a cui è iscritto | Corso (0,n),<br>Insegnate (1,1) | data |
| Commissione | Associa un Allievo alle Assenze che commette | Allievo (0,n)<br>Assenza (1,1) | |
| Orario Scolastico | Associa un Corso e le Lezioni di Corso che compongono il suo orario | Corso (0,n),<br>Lezione Corso (1,1) | |
| Insegnamento | Associa una Lezione Corso e l'Insegnante assegnato a quella lezione | Lezione Corso (1,1),<br>Insegnante(0,n) | |
| Prenotazione | Associa una Lezione Privata e l'Allievo che la prenota | Allievo (0,n),<br>Lezione Privata (1,1) | |
| Impartizione | Associa una Lezione Privata e l'insegnate che la impartisce | Insegnante (0,n),<br>Lezione Privata (1,1) | |
| Partecipazione | Associa un Allievo con le Attività Culturali a cui sceglie di partecipare | Allievo (0,n),<br>Attività Culturale (0,n) | |
| | | | |

# 4. Progettazione logica

## Volume dei dati

Come dice la specifica, tutte le informazioni mantenute nella base di dati sono relative ad un unico anno: possiamo quindi dire che il volume dei dati, le frequenze e I costi, sono tutti quanti a regime. Infatti le informazioni all'interno della base di dati non cresceranno al trascorrere del tempo, (come potrebbe accadere se mantenessi anche infromazioni relative ad anni scolastici precedenti), quindi I carichi che la base supporta saranno più o meno stabili.

- Supponiamo che 2/3 dei livelli prevedano un esame e 1/3 dei livelli non lo prevedano.
- Per le Attività Culturali consideriamo le 52 settimane dell'anno; supponiamo di escludere un mese per l'estate e uno per le vacanze di Natale e abbiamo 44 settimane; supponiamo che in queste settimane vengano organizzati tre proiezioni nei giorni feriali (ad esempio la sera dopo le lezioni di lunedì, mercoledì e venerdì) e una conferenza il sabato pomeriggio
  - Delle attività quindi ¾ sono proiezioni e ¼ sono conferenze
- Supponiamo che per ogni livello ci siano all'incirca quattro corsi
- Supponiamo che vi siano in media 25 allievi a corso (dimensione media di una classe italiana)
- Supponiamo che ci siano in media un terzo dei corsi di insegnanti
- Supponiamo che un allievo faccia in media 25 assenze
- Supponiamo che ogni corso abbia in media quattro lezioni a settimana (due per speaking e listening, una per writing e una per literature)
- Supponiamo che ogni allievo richieda in media 40 lezioni private
- Supponiamo che ad ogni attività partecipino i ¾ degli allievi
- Supponiamo che in media un insegnante abbia quattro corsi assegnati

| Concetto nello schema | Tipo[1] | Volume atteso |
|---|---|---|
| Livello | E | 15 |
| Livello con Esame | E | 10 |
| Livello senza Esame | E | 5 |
| Corso | E | 60 |
| Allievo | E | 1500 |
| Insegnante | E | 20 |
| Assenza | E | 37500 |
| Lezione Corso | E | 240 |
| Lezione Privata | E | 60000 |

1 Indicare con E le entità, con R le relazioni

| | | |
|---|---|---|
| Attività Culturale | E | 176 |
| Proiezione | E | 132 |
| Conferenza | E | 44 |
| | | |
| Appartenenza | R | 60 |
| Iscrizione | R | 1500 |
| Docenza | R | 80 |
| Orario Scolastico | R | 240 |
| Insegnamento | R | 240 |
| Impartizione | R | 60000 |
| Commissione | R | 37500 |
| Prenotazione | R | 60000 |
| Partecipazione | R | 198000 |
| | | |
| | | |

## Tavola delle operazioni

- Supponiamo che ogni insegnante generi la sua agenda circa due volta al giorno, per vedere gli impegni della mattina e del pomeriggio di quella giornata
- Supponiamo che il report sulle attività sia generato una volta ogni due settimane per ogni insegnante
- Supponiamo che un Allievo partecipi in media a ¾ delle Attività Culturali previste nella settimana
- Supponiamo che ogni Allievo abbia tre giorni preferiti per le lezioni private e che mediamente nel primo non ci siano Insegnanti per fare lezione (generazione del report per due volte per ogni lezione privata che viene prenotata)
- Supposto che il report delle assenze sia importante per permettere ad un Allievo di fare l'esame che eventualmente il livello prevede e che sia monitorato una volta ogni due settimane

| Cod. | Descrizione | Frequenza attesa | Unità Tempo |
|---|---|---|---|
| Op.1 | Riavvio Anno Scolastico | 1 | anno |
| | | | |
| Op.2 | Inserimento del Livello e delle sue informazioni | 15 | anno |
| Op.3 | Inserimento del Corso e delle sue informazioni | 60 | anno |
| Op.4 | Inserimento di un Insegnante e delle sue informazioni | 20 | anno |
| Op.5 | Iscrizione di un Allievo con le relative informazioni | 34,09 | settimana |
| Op.6 | Inserimento di una Lezione Corso indicando l'Insegnante che la Insegna | 5,45 | settimana |
| Op.7 | Inserimento di un'Attività Culturale organizzata dalla Scuola | 4 | settimana |
| Op.8 | Assegnazione di un Insegnante ad un Corso | 1,82 | settimana |

| | | | |
|---|---|---|---|
| Op.9 | Registrazione Assenza di un Allievo | 121,75 | giorno |
| Op.10 | Generazione Agenda di Insegnante | 40 | giorno |
| Op.11 | Generazione Report riguardo Insegnante | 40 | settimana |
| Op.12 | Registrazione Partecipazione di Allievo ad Attività Culturale | 642,86 | giorno |
| Op.13 | Prenotazione di un Allievo per una Lezione Privata con un Insegnante | 194,81 | giorno |
| | | | |
| Op.14 | Stampa tutti gli Insegnanti senza impegni in una certa data e in certa fascia oraria | 389,61 | giorno |
| Op.15 | Stampa report assenze di un corso indicando il numero di assenze per ogni allievo | 120 | settimana |
| Op. 16 | Stampa tutte le attività culturali dopo data attuale | 642,86 | settimana |
| | | | |
| Op. 17 | Stampa tutti I livelli | 60 | anno |
| Op. 18 | Stampa tutti i corsi | 35,91 | settimana |
| Op. 19 | Stampa tutti gli insegnanti | 41,82 | settimana |
| Op. 20 | Stampa tutte le docenze degli insegnanti | 5,45 | settimana |
| | | | |
| | | | |
| | | | |

## Costo delle operazioni

*Tabella 1: Operazione 1*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Tutti | E-R | 457322 | S |

*Tabella 2: Operazione 2*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Livello | E | 1 | S |

*Tabella 3: Operazione 3*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Corso | E | 1 | S |
| Livello | E | 1 | L |
| Appartenenza | R | 1 | S |

*Tabella 4: Operazione 4*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Insegnante | E | 1 | S |

*Tabella 5: Operazione 5*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Allievo | E | 1 | S |
| Corso | E | 1 | L |
| Iscrizione | R | 1 | S |
| Corso | E | 1 | S |

Inserire un Allievo per un corso necessita di aggiornare il numero di allievi di quel corso

*Tabella 6: Operazione 6*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Corso | E | 1 | L |
| Docenza | R | 4 | L |
| Insegnante | E | 1 | L |
| Insegnamento | R | 12 | L |
| Lezione Corso | E | 16 | L |
| Impartizione | R | 10 | L |
| Lezione Privata | E | 10 | L |
| Lezione Corso | E | 1 | S |

Per eseguire questa operazione, contestualmente all'inserimento devono essere effettuati anche controlli sugli impegni dell'insegnante e quindi si devono leggere le Lezioni Private e le Lezioni Corso associate allo stesso. Tuttavia possiamo supporre che le Lezioni Corso siano aggiunte per lo più all'inizio dell'anno scolastico, prima che sia prenotata qualunque Lezione Privata o comunque quando esse sono ancora poche: possiamo quindi considerare basso il numero di Lezioni Private ed impartizioni che leggiamo per il controllo sul conflitto degli impegni; se non si vuole stare a questa limitazione si dovrebbe controllare la conflittualità della nuova Lezione Corso con le Lezioni Private successive al momento dell'inserimento. Si deve inoltre controllare anche che l'Insegnante sia

Docente del Corso di cui si sta inserendo la Lezione.
Si deve inoltre controllare che un corso non abbia più lezioni conflittuali nello stesso giorno.

*Tabella 7: Operazione 7*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Attività Culturale | E | 1 | S |

*Tabella 8: Operazione 8*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Insegnante | E | 1 | L |
| Corso | E | 1 | L |
| Docenza | R | 1 | S |

*Tabella 9: Operazione 9*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Allievo | E | 1 | L |
| Corso | E | 1 | L |
| Lezione Corso | E | 1 | L |
| Assenza | E | 1 | S |
| Commissione | R | 1 | S |
| Allievo | E | 1 | S |

Quando si inserisce un'Assenza per un Allievo si deve aggiornare il numero delle assenze dell'allievo.

Si deve inoltre controllare che esista una LezioneCorso nel giorno della settimana della data e nell'orario dell'assenza

*Tabella 10: Operazione 10*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Insegnante | E | 1 | L |

| | | | |
|---|---|---|---|
| Impartizione | R | 68,18 | L |
| Insegnamento | R | 12 | L |
| Lezione Corso | E | 12 | L |
| Lezione Privata | E | 68,18 | L |

Visto che la generazione dell'agenda è settimanale, leggeremo soltanto le Lezioni Private dell'insegnante di quella specifica settimana, mentre leggeremo tutte le Lezioni Corso visto che le lezioni sono state supposte su base settimanale.

*Tabella 11: Operazione 11*

| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
|---|---|---|---|
| Insegnante | E | 1 | L |
| Impartizione | R | 300 | L |
| Insegnamento | R | 12 | L |
| Lezione Privata | E | 300 | L |
| Lezione Corso | E | 12 | L |

Il report è su base mensile, quindi recuperiamo soltanto le Lezioni Private che sono state svolte da quell'insegnante in quello specifico mese

*Tabella 12: Operazione 12*

| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
|---|---|---|---|
| Allievo | E | 1 | L |
| Attività Culturale | E | 1 | L |
| Partecipazione | R | 1 | S |

*Tabella 13: Operazione 13*

| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
|---|---|---|---|
| Allievo | E | 1 | L |
| Insegnante | E | 1 | L |
| Insegnamento | R | 2 | L |

| | | | |
|---|---|---|---|
| Impartizione | R | 9,74 | L |
| Lezione Corso | E | 2 | L |
| Lezione Privata | E | 9,74 | L |
| Prenotazione | R | 0,13 | L |
| Lezione Privata | E | 0,13 | L |

Per prenotare una Lezione Privata, bisogna verificare che:

- Non ci siano Lezioni Corso dell'Insegnante in quel giorno. Prendiamo quindi il numero medio di Lezioni Corso di un Insegnante nel giorno della settimana che coincide con la data della lezione
- Non ci siano Lezioni Private dell'Insegnante in orari che collidono. Prendiamo quindi il numero medio di Lezioni Privte di un Insegnante in un giorno (considero l'anno di 44 settimane)
- Non ci siano altre Lezioni Private dell'Allievo in un orario che collide. Prendiamo quindi il numero medio di Lezioni Private prenotate da un Allievo in un certo giorno

*Tabella 14: Operazione 14*

| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
|---|---|---|---|
| Insegnante | E | 20 | L |
| Insegnamento | R | 4,44 | L |
| Lezione Corso | E | 4,44 | L |
| Impartizione | R | 21,65 | L |
| Lezione Privata | E | 21,65 | L |

Le considerazioni da fare sono simili a quelle fatte per Op.13. Consideriamo la giornata lavorativa della scuola dalle 9:00 alle 18:00 e la durata media di una lezione di un'ora.
Prendo quindi Il numero medio di lezioni corso e lezioni private in una fascia oraria.

*Tabella 15: Operazione 15*

| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
|---|---|---|---|
| Corso | E | 1 | L |
| Iscrizione | R | 25 | L |
| Allievo | E | 25 | L |

*Tabella 16: Operazione 16*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Attività Culturali | E | 16 | L |

Si è supposto che il numero medio di attività culturali a settimana fosse 4. Supponiamo che vengano organizate a cadenza mensile e che quindi ad un certo istante si conoscano le attività che ci saranno da lì a 30 giorni

*Tabella 17: Operazione 17*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Livello | E | 15 | L |

*Tabella 18: Operazione 18*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Corso | E | 60 | L |

*Tabella 19: Operazione 19*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Insegnante | E | 20 | L |

*Tabella 20: Operazione 20*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Docenza | E | 80 | L |

*Tabella 21: Tabella Costi Complessivi*

| Operazione | Letture | Scritture | Costo Totale |
|---|---|---|---|
| Op.1 | 0 | 457322 | 914644 |
| Op.2 | 0 | 1 | 2 |
| Op.3 | 1 | 2 | 5 |
| Op.4 | 0 | 1 | 2 |

| | | | |
|---|---|---|---|
| Op.5 | 1 | 3 | 7 |
| Op.6 | 54 | 1 | 56 |
| Op.7 | 0 | 1 | 2 |
| Op.8 | 2 | 1 | 4 |
| Op.9 | 3 | 3 | 9 |
| Op.10 | 161,36 | 0 | 161,36 |
| Op.11 | 625 | 0 | 625 |
| Op.12 | 2 | 1 | 4 |
| Op.13 | 25,48 | 0 | 25,48 |
| Op.14 | 72,18 | 0 | 72,18 |
| Op.15 | 51 | 0 | 51 |
| Op.16 | 16 | 0 | 16 |
| Op.17 | 15 | 0 | 15 |
| Op.18 | 60 | 0 | 60 |
| Op.19 | 20 | 0 | 20 |
| Op.20 | 80 | 0 | 80 |
| | | | |
| | | | |

## Ristrutturazione dello schema E-R

### Analisi delle Ridondanze

*Numero di Assenze*

Il numero di assenze dell'entità Allievo rappresenta ridondanza in quanto esso è ricavabile per conteggio dalle occorrenze dell'entità assenza.

Mantenimento della Ridondanza

Le operazioni che operano in qualche modo sul numero di assenze di un allievo sono:

- Operazione 9
- Operazione 15

Per quanto concerne I costi di queste operazioni in presenza di ridondanza rimandiamo a quanto discusso sopra nelle tabelle.

Per l'eventuale utilizzo di memoria aggiuntiva derivante dal mantenimento della ridondanza,

possiamo supporre che il numero di assenze di un allievo sia mantenuto in un tipo di dato intero e quindi occupante 4 B. Considerando la tavola dei volumi di cui sopra abbiamo un totale di

$$4\,B * 1500\,Allievi = 6000\,B = 6\,kB$$

di memoria utilizzata.

Rimozione della Ridondanza

Rimuovendo la ridondanza dobbiamo rivalutare I costi delle operazioni che operano sulle assenze.

*Tabella 22: Operazione 9 - Senza Ridondanza "Numero Assenze"*

| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
|---|---|---|---|
| Allievo | E | 1 | L |
| Corso | E | 1 | L |
| Lezione Corso | E | 1 | L |
| Assenza | E | 1 | S |
| Commissione | R | 1 | S |

*Tabella 23: Operazione 15 - Senza Ridondanza "Numero Assenze"*

| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
|---|---|---|---|
| Corso | E | 1 | L |
| Iscrizione | R | 25 | L |
| Allievo | E | 25 | L |
| Commissione | R | 625 | L |
| Assenza | E | 625 | L |

*Tabella 24: Costi Complessivi Senza Ridondanza "Numero Assenze"*

| **Operazione** | **Letture** | **Scritture** | **Costo Totale** |
|---|---|---|---|
| Op.9 | 3 | 2 | 7 |
| Op.15 | 1301 | 0 | 1301 |

Valutazione Complessiva

*Tabella 25: Valutazione Complessiva – Ridondanza "Numero Assenze"*

| | Con Ridondanza | Senza Ridondanza | Frequenza |
|---|---|---|---|
| Op.9 | 9 | 7 | 121,75 / giorno |
| Op.15 | 51 | 1301 | 120 / giorno |
| Memoria | 6 kB | 0 | |

Notiamo quindi che, mentre il costo dell'operazione 9 non cambia significamente alla rimozione della ridondanza, il costo dell'operazione 15 cambia molto. Alla luce di ciò, tenuto in considerazione il basso spreco di memoria, possiamo MANTENERE la ridondanza.

*Numero di Allievi*

Il numero di allievi di un corso rappresenta ridondanza in quanto ricavabile per conteggio di occorrenze dell'entità Allievo navigata tramite l'associazione Iscrizione.

Mantenimento della Ridondanza

Le operazioni che operano sul numero di iscritti al corso sono quelle che operano sulle iscrizioni degli allievi; in particolare abbiamo:.

- Operazione 5
- Operazione 18

I cui costi sono quelli delle tabelle di cui sopra.

Per quanto riguarda l'utilizzo di memoria, supponiamo che il numero di allievi di un corso sia mantenuto in un intero da 4 B. Questo comporta, dato il numero medio di corsi:

$$4\,B * 60\,Corsi = 240\,B = 0.24\,kB$$

Rimozione della Ridondanza

Valutiamo I costi delle operazioni in assenza di ridondanza.

*Tabella 26: Operazione 5 - Senza Ridondanza "Numero Allievi"*

| Concetto | Costrutto | Accessi | Tipo |
|---|---|---|---|
| Allievo | E | 1 | S |
| Corso | E | 1 | L |

| | | | |
|---|---|---|---|
| Iscrizione | R | 1 | S |

*Tabella 27: Operazione 18 - Senza Ridondanza "Numero Allievi"*

| **Concetto** | **Costrutto** | **Accessi** | **Tipo** |
|---|---|---|---|
| Corso | E | 60 | L |
| Iscrizione | R | 1500 | L |
| Allievo | E | 1500 | L |

*Tabella 28: Costi Complessivi Senza Ridondanza "Numero Allievi"*

| **Operazione** | **Letture** | **Scritture** | **Costo Totale** |
|---|---|---|---|
| Op.5 | 1 | 2 | 5 |
| Op.18 | 3060 | 0 | 3060 |

Valutazione Complessiva

*Tabella 29: Valutazione Complessiva - Ridondanza "Numero Allievi"*

| | **Con Ridondanza** | **Senza Ridondanza** | **Frequenza** |
|---|---|---|---|
| Op.5 | 7 | 5 | 34,09 / settimana |
| Op.18 | 60 | 3060 | 35,91 / settimana |
| Memoria | 0.24 kB | 0 | |

Considerando quindi la differenza tra le due situazioni, scegliamo di MANTENERE la ridondanza.

*Relazione Docenza*

La relazione docenza può essere considerata una ridondanza in quanto la docenza di un Insegnante verso un Corso può essere ricavata per navigazione di Insegnamento, Lezione Corso e Orario Scolastico. C’è da dire però che I concetto rappresentato dalla relazione Docenza non è il medesimo rappresentato dalla navigazione. La relazione docenza permette infatti di inserire la Docenza di un Insegnante di un corso anche nel momento in cui il corso non ha ancora una lezione assegnata per quel corso. Questa cosa può essere utile in un contesto scolastico in cui prima si fanno le assegnazioni delle cattedre e poi si stabilisce l’orario scolastico. Ricavare la Docenza tramite la

navigazione invece implichrebbe che il docente deve avere necessariamente una lezione di un corso assegnata per poter essere ufficialmente assegnato a quel corso.
Data questa differenza, scegliamo di MANTENERE la relazione Docenza proprio perché a livello concettuale non è ricavabile altrove.

**Eliminazione delle Generalizzazioni**

*Livello e Figlie*

Per quanto riguarda il Livello con le sue generalizzazioni Livello Con Esame e Livello Senza Esame abbiamo che le operazioni indicate non prevedono distinzioni particolari tra figlie e genitore. Possiamo quindi ipotizzare di accorpare le figlie nel genitore: questo porta all'aggiunta di un attributo "esame" nell'entità Livello che permetta di discriminare un livello di un tipo da un livello di un altro. Essendo la distinzione binaria, possiamo supporre che sia sufficiente un bit per indicare questa distinzione ed essendo i livelli 15 in totale abbiamo un consumo di memoria di 15 bit, che possiamo permetterci viste le dimensioni complessive della base di dati

*Attività Culturale e Figlie*

Per quanto riguarda l'attività culturale, valgono le stesse considerazioni del livello: anche qui non abbiamo operazioni che si riferiscono ad una figlia o all'altra in modo particolare. Supponendo di accorpare le entità figlie nell'entità genitore, questo comporta:

- sia necessario un attributo "tipo" sull'Attività culturale che ci permetta di distinguere un'istanza di figlia da un'istanza di un'altra figlia; essendo la generalizzazione totale l'attributo tipo è obbligatorio per tutte le occorrenze.
  - Come costo possiamo considerare anche qui un bit per ogni occorrenza e quindi in totale 176 bit = 22 B = 0,022 kB
- sia necessario portare gli attributi delle entità figlie nell'entità genitore imponendo come vincolo di integrità che
  - film e regista siano nulli per occorrenze di tipo "Conferenza"
    - supponendo di usare una stringa di 100 char per il titolo e di 100 char per il regista, abbiamo 200 B di null per ogni conferenza, per un totale di 200 B * 44 Conferenza = 8800 B = 8.8 kB
  - argomento e conferenziere siano nulli per occorrenze di tipo "Proiezione"
    - Supponendo di usare una stringa di 100 char per l'argomento e di 100 char per il nome del conferenziere abbiamo 200 B di null per ogni Proiezione, per un totale di 200 B * 132 Proiezioni = 26400 B = 26,4 kB

Accorpando quindi le figlie nella genitore abbiamo uno spreco di memoria totale di 35,22 kB.

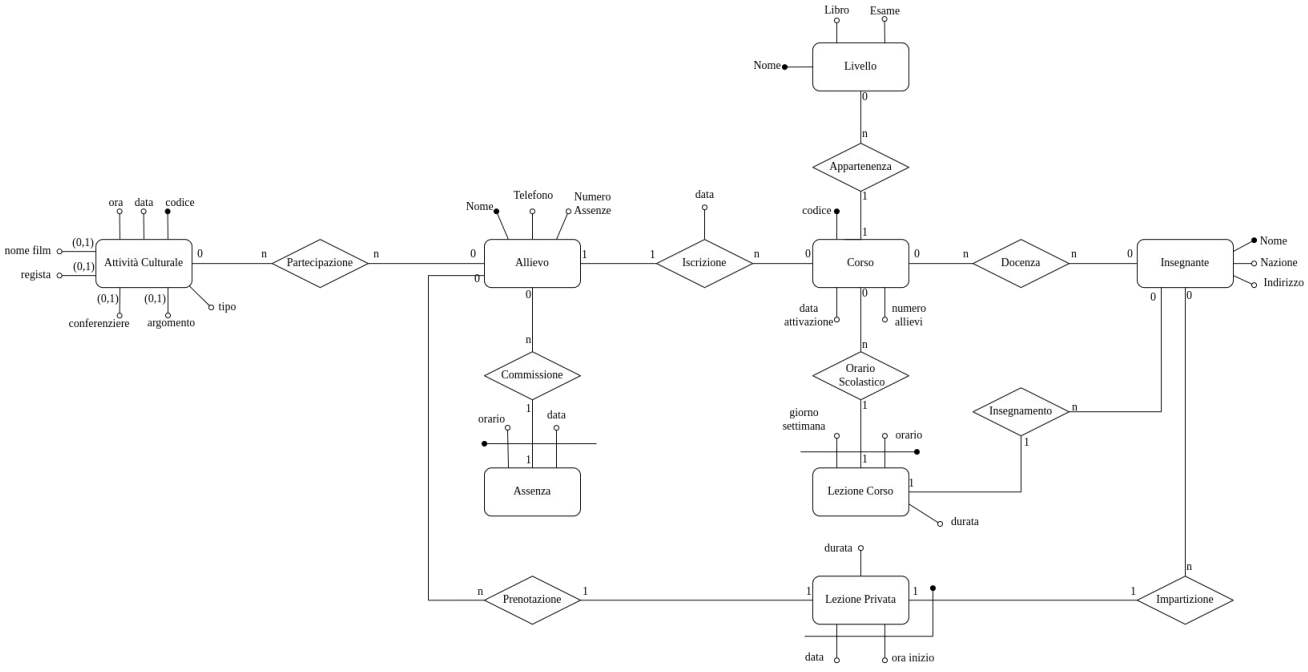
In alternativa all'accorpamento delle figlie nel genitre possiamo valutare di creare delle entità distinte relazionate al genitore e identificate esternamente tramite esso; questa scelta permetterebbe di avere delle relazioni più piccole e quindi più facili da maneggiare e ci permetterebbe di risparmiare i 35 kB di memoria null di cui sopra. Tuttavia, non essendoci necessità nelle operazioni di accedere all'una o all'altra entità in modo distinto, questa scelta non sembra essere la migliore da percorrere: infatti le operazioni relative alle Attività Culturali sono:

- Inserimento dell'Attività Culturale. Per questa operazione sicuramente l'accorpamento delle entità nelle figlie è la più conveniente perché permette di inserire solo un'occorrenza dell'unica entità accorpata: la seconda opzione infatti necessiterebbe dell'aggiunta di 3 occorrenze, una per entità padre, una per entità figlia e una per l'associazione che le collega

## Trasformazione di attributi e identificatori

Notiamo che da dominio, il Corso è identificato da un codice numerico e dal nome del Livello a cui afferisce: scegliamo di rimuovere l'identificazione esterna del Livello e di lasciare soltanto un codice numerico univoco che identifichi il Corso in toto, senza necessità del nome del Livello.

## Traduzione di entità e associazioni



Partendo dal diagramma ER ristrutturato di sopra, costruiamo le seguenti relazioni.

*Livello(nomeLivello, libro, esame)*

*Corso(codice, nomeLivello, dataAttivazione, numeroAllievi)*

- Corso(nomeLivello) ⊆ Livello(nomeLivello)

*Allievo(nomeAllievo, telefonoAllievo, numeroAssenze, dataIscrizione, codiceCorso)*

- Allievo(codiceCorso) ⊆ Corso(codice)

*Insegnante(nomeInsegnante, nazione, indirizzo)*

*Docenza(codiceCorso, nomeInsegnante)*

- Docenza(codiceCorso) ⊆ Corso(codice)
- Docenza(nomeInsegnante) ⊆ Insegnante(nomeInsegnante)

*LezioneCorso(giornoSettimana, orarioInizio, codiceCorso, durata, insegnante)*

- LezioneCorso(codiceCorso) ⊆ Corso(codice)
- LezioneCorso(insegnante) ⊆ Insegnante(nomeInsegnante)

*Assenza(nomeAllievo, data, orario)*

- Assenza(nomeAllievo) ⊆ Allievo(nomeAllievo)

*LezionePrivata(dataLezione, orarioLezione, nomeInsegnante, durataLezione, nomeAllievo)*

- LezionePrivata(nomeInsegnante) ⊆ Insegnante(nomeInsegnante)
- LezionePrivata(nomeAllievo) ⊆ Allievo(nomeAllievo)

*Partecipazione(codiceAttività, nomeAllievo)*

- Partecipazione(codiceAttività) ⊆ AttivitàCulturale(codiceAttività)
- Partecipazione(nomeAllievo) ⊆ Allievo(nomeAllievo)

*AttivitàCulturale(codiceAttività, dataAttività, oraAttività, tipoAttività, film*, registaFilm*, conferenziere*, argomentoConferenza*)*

Nota. Si indicano con * I possibili valori null

Per AttivitaCulturale deve valere che:

- tipoAttività = 0 OPPURE tipoAttivita = 1
- se tipoAttività = 0
  - film != NULL e registaFilm != NULL
  - conferenziere = NULL e argomentoConferenza = NULL
- se tipoAttività = 1
  - film = NULL e registaFilm = NULL
  - conferenziere != NULL e argomentoConferenza != NULL

## Normalizzazione del modello relazionale

1NF

Ricordando che una relazione soddisfa la 1NF se

- è presente chiave primaria
- non ci sono attributi ripetuti
- gli attributi non sono strutture complesse

Abbiamo che tutte le relazioni soddisfano la 1NF

2NF

Ricordando che una relazione è in 2NF se la chiave scelta non contiene chivi più piccole (ovvero la chiave scelta è effettivamente una chiave e non una superchiave).

In tutte le relazioni le chiavi non contengono chiavi più piccole e quindi sono minimali.

3NF

Una relazione è in terza forma normale se per ogni dipendenza funzionale $X \rightarrow A$ vale una delle seguenti:

- X è superchiave della relazione
- A appartiene ad almeno una chiave

Nel nostro caso questo vale per tutte le relazioni.

# 5. Progettazione fisica

## Utenti e privilegi

Di seguito indichiamo le tipologie di utenti previste nella base, per ognuno l'elenco dei loro privilegi e per ognuno dei privilegi le risorse a cui questo privilegio dà accesso

- Amministrazione
  - Tipologia di utente per la connessione come utente di amministrazione
  - Il ruolo associato è 'amministrazione' con privilegi di
    - esecuzione
      - aggiungi_corso
        - Corso
      - aggiungi_insegnante
        - Insegnante
        - Utenti. Quando si inserisce un nuovo insegnante gli si devono associare delle credenziali affinché possa poi accedere al sistema
      - aggiungi_lezione
        - LezioneCorso
        - Docenza. Per il controllo dell'assegnamento dell'Insegnante al Corso di cui si sta inserendo la Lezione
        - LezionePrivata. Per il controllo sul conflitto con delle LezioniPrivate
      - aggiungi_livello.
        - Livello
      - assegna_corso
        - Docenza
      - genera_report_insegnante
        - LezioneCorso. Per leggere tutte le LezioniCorso dell'Insegnante
        - LezionePrivata. Per leggere tutte le LezioniPrivate dell'Insegnante
      - organizza_attivita_culturale
        - AttivitàCulturale
      - recupera_corsi
        - Corso
      - recupera_docenze
        - Docenza

- recupera_insegnanti
  - Insegnante
- recupera_livelli
  - Livello
- riavvia_anno.
  - Tutte Tabelle. Elimina le tuple da tutte le tabelle, tranne le due tuple utenza di amministrazione e segreteria nella tabella Utenti

- Segreteria
  - Tipologia di utente per la connessione come utente di ‘segreteria’
  - Il ruolo associato è ‘segreteria’ con privilegi di
    - esecuzione
      - aggiungi_allievo
        - Allievo
        - Corso. Aggiornamento del numero di allievi di un Corso
      - aggiungi_assenza
        - Assenza
        - Allievo
        - LezioneCorso. Per la verifica di una Lezione con giorno e orario che combaciano
      - aggiungi_partecipazione
        - Partecipazione
      - prenota_lezione_privata
        - LezionePrivata. Per inserimento e per il controllo sul conflitto di altre LezioniPrivate del medesimo allievo
        - LezioneCorso. Per controllo conflitto con LezioniCorso dello stesso insegnante
      - recupera_attivita
        - AttivitaCulturale
      - recupera_corsi
        - Corso
      - recupera_insegnanti_liberi
        - Insegnante
        - LezioneCorso. Per selezionare solo gli Insegnanti che non hanno Lezioni Corso in una certa fascia oraria

        - LezionePrivata. Per selezionare solo gli Insegnanti che non hanno Lezioni Private in una certa fascia oraria
    - report_assenze_corso
        - Allievo. Per recuperare il numero di assenze degli allievi di un certo Corso
- Insegnante
    - Tipologia di utente per la connessione come utente ‘insegnante’
    - Il ruolo associato a questo utente è ‘insegnante’ con privilegio di:
        - esecuzione
            - genera_agenda
                - LezioneCorso. Per recuperare tutte le Lezioni Corso di quell’Insegnante in certa settimana
                - LezionePrivata. Per recuperare tutte le Lezioni Private di quell’Insegnante in certa settimana
- Login
    - Tipologia di utente introdotta per stabilire la connessione con il DB
    - Il ruolo associato è ‘login’ con privilegi di
        - esecuzione
            - login.
                - Utenti. Per verificare le credenziali di colui che cerca di accedere. Questo permette di fare in modo che un utente non autenticato possa eseguire solo e unicamente il login nel sistema, mentre gli è preclusa l’esecuzione di qualsiasi altra procedura e quindi l’accesso a tutte le altre risorse del sistema

## Strutture di memorizzazione

| **Tabella Utenti** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi**[2] |
| **username** | varchar(100) | PK,NN |
| **password** | varchar(45) | NN |
| **ruolo** | ENUM('amministrazione', 'segreteria', 'insegnante') | NN |

| **Tabella Livello** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi** |

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

| | | |
|---|---|---|
| **nomeLivello** | varchar(45) | PK |
| **libro** | varchar(45) | NN |
| **esame** | smallint | NN |

| **Tabella Corso** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi** |
| **codiceCorso** | INT | PK,NN,AI |
| **nomeLivello** | varchar(45) | NN |
| **dataAttivazione** | DATE | NN |
| **numeroAllievi** | INT | NN |

| **Tabella Allievo** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi** |
| **nomeAllievo** | varchar(100) | PK,NN |
| **telefonoAllievo** | varchar(15) | NN |
| **numeroAssenze** | INT | NN |
| **dataIscrizione** | DATE | NN |
| **codiceCorso** | INT | NN |

| **Tabella Insegnante** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi** |
| **nomeInsegnante** | varchar(100) | PK,NN |
| **nazioneInsegnante** | varchar(45) | NN |
| **indirizzoInsegnante** | varchar(100) | NN |
| **username** | varchar(100) | NN, UQ |

| **Tabella Docenza** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi** |
| **nomeInsegnante** | varchar(100) | PK,NN |
| **codiceCorso** | INT | PK,NN |

| **Tabella LezioneCorso** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi** |
| **giornoSettimana** | ENUM('Lun', 'Mar', 'Mer', 'Gio', 'Ven', 'Sab', 'Dom') | PK,NN |
| **orarioInizio** | TIME | PK,NN |
| **codiceCorso** | INT | PK,NN |
| **durata** | INT | NN |
| **insegnanteLezione** | varchar(100) | NN |

| **Tabella LezionePrivata** |
|---|

| Colonna | Tipo di dato | Attributi |
|---|---|---|
| **dataLezione** | DATE | PK,NN |
| **orarioLezione** | TIME | PK,NN |
| **nomeInsegnante** | varchar(100) | PK,NN |
| **durataLezione** | INT | NN |
| **nomeAllievo** | varchar(100) | NN |

| **Tabella AttivitaCulturale** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi** |
| **codiceAttivita** | INT | PK,NN, AI |
| **dataAttivita** | DATE | NN |
| **orarioAttivita** | TIME | NN |
| **tipoAttivita** | SMALLINT | NN |
| **titoloFilm** | varchar(100) | |
| **registaFilm** | varchar(100) | |
| **conferenziere** | varchar(100) | |
| **argomentoConferenza** | varchar(100) | |

TipoAttivita : 0 = film, 1 = conferenza

| **Tabella Partecipazione** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi** |
| **codiceAttivita** | INT | PK,NN |
| **nomeAllievo** | varchar(100) | PK,NN |

| **Tabella Assenza** | | |
|---|---|---|
| **Colonna** | **Tipo di dato** | **Attributi** |
| **dataAssenza** | DATE | PK,NN |
| **nomeAllievo** | VARCHAR(100) | PK,NN |
| **orarioAssenza** | TIME | PK, NN |

## Indici

Di seguito, tutti gli indici PRIMARY sono gli indici creati per la chiave primaria della relazione e quindi per l'identificazione univoca delle tuple.

| **Tabella Allievo** | |
|---|---|
| **Indice PRIMARY** | **Tipo[3]: PR** |
| Colonna 1 | nomeAllievo |
| | |
| **Indice fk_Allievo_1_idx** | **Tipo: IDX** |
| Colonna 1 | codiceCorso |

3 IDX = index, UQ = unique, FT = full text, PR = primary.

- fk_Allievo_1_idx. Per la Foreign Key di Allievo verso il Corso a cui è iscritto

| **Tabella Assenza** | |
|---|---|
| **Indice PRIMARY** | **Tipo: PR** |
| Colonna 1 | nomeAllievo |
| Colonna 2 | orarioAssenza |
| Colonna 3 | dataAssenza |
| | |
| **Indice fk_Assenza_1_idx** | **Tipo: IDX** |
| Colonna 1 | nomeAllievo |
| | |

- fk_Assenza_1_idx. Per la Foreign Key dell'Assenza verso Allievo che la commette

| **Tabella AttivitaCulturale** | |
|---|---|
| **Indice PRIMARY** | **Tipo: PR** |
| Colonna 1 | codiceAttivita |

| **Tabella Corso** | |
|---|---|
| **Indice PRIMARY** | **Tipo: PR** |
| Colonna 1 | codiceCorso |
| | |
| **Indice fk_Corso_1_idx** | **Tipo: IDX** |
| Colonna 1 | nomeLivello |

- fk_Corso_1_idx. Per la Foreigne Key di Corso verso il Livello a cui afferisce

| **Tabella Docenza** | |
|---|---|
| **Indice PRIMARY** | **Tipo: PR** |
| Colonna 1 | codiceCorso |
| Colonna 2 | nomeInsegnante |
| | |
| **Indice fk_Docenza_1_idx** | **Tipo: IDX** |
| Colonna 1 | nomeInsegnante |
| | |
| **Indice fk_Docenza_2_idx** | **Tipo: IDX** |
| Colonna 1 | codiceCorso |

- fk_Docenza_1_idx. Per la Foreign Key di Docenza verso l'Insegnante a cui si riferisce.
- fk_Docenza_2_idx. Per la Foreign Key di Docenza verso il Corso cui si riferisce

| **Tabella Insegnante** | |
|---|---|
| **Indice PRIMARY** | **Tipo: PR** |
| Colonna 1 | nomeInsegnante |
| | |

| **Indice username_UNIQUE** | **Tipo: UQ** |
|---|---|
| Colonna 1 | username |
| | |
| **Indice fk_Insegnante_1_idx** | **Tipo: IDX** |
| Colonna 1 | username |

- fk_Insegnante_1_idx. Per la Foreign Key di Insegnante verso il suo Username
- username_UNIQUE. Lo username dell'Insegnante è unico per lui. Quando un insegnante esegue la sua unica operazione, la esegue indicando il suo username e noto questo si possono recuperare le informazioni (in particolare nomeInsegnante) in modo da recuperare le Lezioni che esso svolge. Costruire un'indice su username velocizza il recupero delle informazioni.

| **Tabella LezioneCorso** | |
|---|---|
| **Indice PRIMARY** | **Tipo: PR** |
| Colonna 1 | giornoSettimana |
| Colonna 2 | orarioInizio |
| Colonna 3 | codiceCorso |
| | |
| **Indice fk_LezioneCorso_1_idx** | **Tipo: IDX** |
| Colonna 1 | codiceCorso |
| | |
| **Indice fk_LezioneCorso_2_idx** | **Tipo: IDX** |
| Colonna 1 | insegnanteLezione |

- fk_LezioneCorso_1_idx. Per la Foreign Key di LezioneCorso verso il Corso di cui è lezione
- fk_LezioneCorso_2_idx. Per la Foreign Key di LezioneCorso verso Insegnante che la tiene

| **Tabella LezionePrivata** | |
|---|---|
| **Indice PRIMARY** | **Tipo: PR** |
| Colonna 1 | dataLezione |
| Colonna 2 | orarioLezione |
| Colonna 3 | nomeInsegnante |
| | |
| **Indice fk_LezionePrivata_1_idx** | **Tipo: IDX** |
| Colonna 1 | nomeInsegnante |
| | |
| **Indice fk_LezionePrivata_2_idx** | **Tipo: IDX** |
| Colonna 1 | nomeAllievo |

- fk_LezionePrivata_1_idx. Per la Foreign Key di LezionePrivata verso Insegnante che la impartisce
- fk_LezionePrivata_2_idx. Per la Foreign Key di LezionePrivata verso Allievo che la Prenota

| **Tabella Livello** |
|---|

| **Indice PRIMARY** | **Tipo: PR** |
|---|---|
| Colonna 1 | nomeLivello |

| **Tabella Partecipazione** | |
|---|---|
| **Indice PRIMARY** | **Tipo: PR** |
| Colonna 1 | codiceAttivita |
| Colonna 2 | nomeAllievo |
| | |
| **Indice fk_Partecipazione_2_idx** | **Tipo: IDX** |
| Colonna 1 | nomeAllievo |
| | |
| **Indice fk_Partecipazione_1_idx** | **Tipo: IDX** |
| Colonna 1 | codiceAttivita |

- fk_Partecipazione_2_idx. Per la Foreign Key di Partecipazione verso Allievo che partecipa
- fk_Partecipazione_1_idx. Per la Foreign Key di Partecipazione verso Attività a cui è riferita

| **Tabella Utenti** | |
|---|---|
| **Indice PRIMARY** | **Tipo: PR** |
| Colonna 1 | username |

## Trigger

### Allievo_AFTER_INSERT

Trigger Inserito per mantenere aggiornato il numero di allievi di un corso quando si iscrive un nuovo allievo nella scuola: la necessità di questo trigger è data dal fatto che, avendo scelto di mantenere la ridondanza numeroAllievi nel Corso, ogni volta che si inserisce un allievo nella scuola, bisogna aggiornare il conteggio

```
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`Allievo_AFTER_INSERT` AFTER INSERT ON `Allievo` FOR
EACH ROW

BEGIN
      UPDATE `Corso`
      SET Corso.numeroAllievi = Corso.numeroAllievi + 1
      WHERE Corso.codiceCorso = new.codiceCorso ;
END
```

**Assenza_BEFORE_INSERT**

Trigger introdotto per la verifica dell'esistenza di una LezioneCorso a cui è iscritto l'Allievo che abbia lo stesso giorno della settimana della data dell'assenza e lo stesso orario.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`Assenza_BEFORE_INSERT` BEFORE INSERT ON `Assenza`
FOR EACH ROW
BEGIN

      DECLARE var_esiste_lezione INT DEFAULT 0 ;
      DECLARE var_codice_corso INT ;

      SELECT codiceCorso
      FROM Allievo
      WHERE nomeAllievo = new.nomeAllievo
      INTO var_codice_corso ;

      SELECT count(*)
      FROM LezioneCorso
      WHERE
            codiceCorso = var_codice_corso AND
            giornoSettimana = conversioneDataGiorno(new.dataAssenza) AND
            orarioInizio = new.orarioAssenza
      INTO var_esiste_lezione ;

      IF (var_esiste_lezione = 0) THEN
            SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Non Esistono
Lezioni Del Corso Che Combacino' ;
      END IF ;

END
```

**Assenza_AFTER_INSERT**

Trigger inserito per mantenere coerente il numero di assenze commesse da un allievo. La necessità di questo trigger è data dalla scelta di aver mantenuto la ridondanza numeroAssenze per l'entità allievo e quindi all'inserimento di un'assenza bisogna aggiornare il numero di assenze di quell'allievo.

```
CREATE DEFINER = CURRENT_USER TRIGGER
```

```
`my_english_school`.`Assenza_AFTER_INSERT` AFTER INSERT ON `Assenza` FOR
EACH ROW
BEGIN
      UPDATE Allievo
      SET numeroAssenze = numeroAssenze + 1
      WHERE nomeAllievo = new.nomeAllievo;
END
```

**LezioneCorso_BEFORE_INSERT**

Trigger inserito per verificare che la LezioneCorso che si sta inserendo non entri in conflitto con le LezioniCorso dello stesso Insegnante. Nello specifico si effettua un conteggio delle LezioniCorso dello stesso insegnante in un orario che collide con quello della nuova lezione; se il conteggio è maggiore di 0 esiste almeno una lezione che collide con la nuova e quindi si effettua un rollback della transazione.

*verifica_orari_compatibili* è una funzione che torna TRUE se gli orari NON si sovrappongono e FALSE altrimenti (vedere avanti per l'implementazione).

```
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezioneCorso_BEFORE_INSERT` BEFORE INSERT ON
`LezioneCorso` FOR EACH ROW
BEGIN
      #CONTROLLO NON CONFLITTO DI ORARI CON ALTRE LEZIONI DI CORSI
      DECLARE var_has_conflict INT DEFAULT 0;

      SELECT count(*)
      FROM LezioneCorso
      WHERE
            insegnantelezione = new.insegnanteLezione AND
            giornoSettimana = new.giornoSettimana AND
            NOT verifica_orari_compatibili(new.orarioInizio,
new.orarioInizio + INTERVAL new.durata MINUTE, orarioInizio,orarioInizio
+ INTERVAL durata MINUTE)
      INTO var_has_conflict ;

     ## Gestione Segnale Errore
      IF var_has_conflict > 0 THEN
            SIGNAL SQLSTATE '45005' SET MESSAGE_TEXT = 'Insegnante ha una
lezione di un altro corso' ;
       END IF ;
```

```
END
```

**LezioneCorso_BEFORE_INSERT_1**

Trigger inserito per verificare che all'inserimento di una LezioneCorso, l'Insegnante indicato sia effettivamente assegnato al Corso della Lezione. Anche qui si effettua un conteggio sul numero di occorrenze della docenza di quell'insegnante a quel corso; se questo numero non è 1, allora l'Insegnante non è docente del Corso e si fa rollback.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezioneCorso_BEFORE_INSERT_1` BEFORE INSERT ON
`LezioneCorso` FOR EACH ROW
BEGIN

	## Controllo Docenza Assegnata
	DECLARE var_docenze INT DEFAULT 0;

	SELECT count(*)
	FROM Docenza
	WHERE
		Docenza.codiceCorso = NEW.codiceCorso AND
		Docenza.nomeInsegnante = NEW.insegnanteLezione
	INTO var_docenze ;

	IF (var_docenze <> 1) THEN
		SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Insegnante NON
assegnato al Corso Indicato' ;
	END IF ;
END
```

**LezioneCorso_BEFORE_INSERT_2**

Trigger per verificare che l'Insegnante della LezioneCorso inserita non abbia delle LezioniPrivate che collidono con la nuova lezione.

Si esegue un conteggio di tutte le LezioniPrivate dell'Insegnante che possono collidere con la nuova, ovvero che si tengono in una data il cui giorno della settimana coincide con quello della nuova

lezione. Si noti che le LezioniPrivate considerate sono quelle che hanno una data maggiore di quella di inserimento della nuova LezioneCorso: questo perché la LezionePrivata è una tantum e quindi una volta che è passata non interferisce più con la LezioneCorso che viene aggiunta.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezioneCorso_BEFORE_INSERT_2` BEFORE INSERT ON
`LezioneCorso` FOR EACH ROW
BEGIN
      ## CONTROLLO CONFLITTO DELLA LEZIONE CON ALTRE LEZIONI PRIVATE
      DELL'INSEGNANTE

      DECLARE var_has_conflict INT DEFAULT 0 ;

      SELECT count(*)
      FROM LezionePrivata AS P
      WHERE
            nomeInsegnante = new.insegnanteLezione AND
            dataLezione >= NOW() AND
            new.giornoSettimana = conversioneDataGiorno(dataLezione) AND
            NOT verifica_orari_compatibili(new.orarioInizio,
new.orarioInizio + INTERVAL new.durata MINUTE, P.orarioLezione,
P.orarioLezione + INTERVAL P.durataLezione MINUTE)
      INTO var_has_conflict ;

    ## Gestione Segnale Errore
      IF var_has_conflict > 0 THEN
            SIGNAL SQLSTATE '45006' SET MESSAGE_TEXT = 'Insegnante ha una
Lezione Privata in conflitto' ;
      END IF ;
END
```

**LezioneCorso_BEFORE_INSERT_3**

Trigger usato per verificare che lo stesso corso non abbia Lezioni che collidono l’una con l’altra. Anche qui eseguiamo un conteggio ed eventuale rollback se si trova almeno una lezione dello stesso corso che collide

```
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezioneCorso_BEFORE_INSERT_3` BEFORE INSERT ON
`LezioneCorso` FOR EACH ROW
```

```
BEGIN
	DECLARE var_has_conflict INT DEFAULT 0 ;

	SELECT count(*)
	FROM LezioneCorso
	WHERE
		codiceCorso = new.codiceCorso AND
		giornoSettimana = new.giornoSettimana AND
		NOT verifica_orari_compatibili(new.orarioInizio,
new.orarioInizio + INTERVAL new.durata MINUTE, orarioInizio, orarioInizio
+ INTERVAL new.durata MINUTE)
	INTO var_has_conflict ;

	IF var_has_conflict > 0 THEN
		SIGNAL SQLSTATE '45013' SET MESSAGE_TEXT = 'Esiste una lezione
del corso in un  orario che collide' ;
	END IF ;
END
```

**LezionePrivata_BEFORE_INSERT**

Trigger introdotto per verificare che la nuova LezionePrivata non collida con la LezioneCorso dell'Insegnante che impartisce la lezione. Eseguiamo un conteggio sulle LezioniCorso che collidono con l'orario della nuova LezionePrivata.

La LezioneCorso collide con la LezionePrivata solo se essa è nello stesso giorno della settimana della data della privata: la *conversioneDataGiorno* è una funzione che prende la data e restituisce il suo giorno della settimana così come codificato in *giornoSettimana* di LezioneCorso.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezionePrivata_BEFORE_INSERT` BEFORE INSERT ON
`LezionePrivata` FOR EACH ROW
BEGIN

	DECLARE var_has_conflict INT DEFAULT 0 ;

	SELECT count(*)
	FROM LezioneCorso
	WHERE
		insegnanteLezione = new.nomeInsegnante AND
		giornoSettimana = conversioneDataGiorno(new.dataLezione) AND
```

```
            NOT verifica_orari_compatibili(new.orarioLezione,
new.orarioLezione + INTERVAL new.durataLezione MINUTE, orarioInizio,
orarioInizio + INTERVAL durata MINUTE)
    INTO var_has_conflict ;

    IF (var_has_conflict <> 0) THEN
            SIGNAL SQLSTATE '45009' SET MESSAGE_TEXT = 'La Lezione Collide
Con La Lezione di Qualche Corso' ;
    END IF ;
END
```

**LezionePrivata_BEFORE_INSERT_1**

Trigger per la verifica sul conflitto della nuova LezionePrivata con LezioniPrivate dello stesso Insegnante.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezionePrivata_BEFORE_INSERT_1` BEFORE INSERT ON
`LezionePrivata` FOR EACH ROW
BEGIN
     DECLARE var_has_conflict INT DEFAULT 0 ;

     SELECT count(*)
     FROM LezionePrivata
     WHERE
           dataLezione = new.dataLezione AND
           nomeInsegnante = new.nomeInsegnante AND
           NOT verifica_orari_compatibili(new.orarioLezione,
new.orarioLezione + INTERVAL new.durataLezione MINUTE, orarioLezione,
orarioLezione + INTERVAL durataLezione MINUTE)
     INTO var_has_conflict ;

     IF (var_has_conflict > 0) THEN
           SIGNAL SQLSTATE '45011' SET MESSAGE_TEXT = 'La Lezione Collide
Con Altre Lezioni Private Dello Stesso Insegnante' ;
     END IF ;
END
```

**LezionePrivata_BEFORE_INSERT_2**

Trigger per la verifica della prenotazione di più LezioniPrivate dello stesso Allievo in orari che collidono.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezionePrivata_BEFORE_INSERT_2` BEFORE INSERT ON
`LezionePrivata` FOR EACH ROW
BEGIN
    DECLARE var_has_conflict INT DEFAULT 0;

    SELECT count(*)
    FROM LezionePrivata
    WHERE nomeAllievo = NEW.nomeAllievo AND
           dataLezione = new.dataLezione AND
           NOT verifica_orari_compatibili(new.orarioLezione,
new.orarioLezione + INTERVAL new.durataLezione MINUTE, orarioLezione,
orarioLezione + INTERVAL durataLezione MINUTE)
    INTO var_has_conflict ;

    IF (var_has_conflict > 0) THEN
           SIGNAL SQLSTATE '45007' SET MESSAGE_TEXT = 'Alunno ha già
prenotato una lezione in un orario che collide' ;
    END IF ;
END
```

## Eventi

All’istanziazione del DB vengono inserite due tuple nella tabella Utenti

```
START TRANSACTION;

USE `my_english_school`;

INSERT INTO `my_english_school`.`Utenti` (`username`, `password`,
`ruolo`) VALUES ('amministrazione',
'156b7dfafeeef4435ece5367c0e5a5cfd5ae1390', 'amministrazione');

INSERT INTO `my_english_school`.`Utenti` (`username`, `password`,
`ruolo`) VALUES ('segreteria',
'02cad8ee4fb1cc60f6a485316007c9d709293cbd', 'segreteria');


COMMIT;
```

- Amministrazione
  - username = ‘amministrazione’
  - password = SHA1(‘amministrazione’)
  - ruolo = ‘amministrazione’
- Segreteria
  - username = ‘segreteria’
  - password = SHA1(‘segreteria’)
  - ruolo = ‘segreteria’

## Viste

## Funzioni

### verifica_orari_compatibili

Funzione introdotta per evitare di riscrivere ogni volta la condizione di compatibilità delle fasce orarie. L’input è dato da inizio e fine delle due fasce orarie: esse saranno compatibili se la prima finisce prima dell’inizio della seconda oppure se la seconda termina prima dell’inizio della prima e, in questi casi, si ritorna TRUE; in tutti gli altri casi le fasce orarie sono in conflitto e si torna FALSE.

```
CREATE FUNCTION `verifica_orari_compatibili` (
      var_inizio_1 TIME,
      var_fine_1 TIME,
      var_inizio_2 TIME,
      var_fine_2 TIME)
      RETURNS INT
      DETERMINISTIC

BEGIN
      IF (var_fine_1 <= var_inizio_2 OR var_fine_2 <= var_inizio_1) THEN
            RETURN TRUE ;
      ELSE RETURN FALSE ;
      END IF ;
END
```

**conversioneDataGiorno**

Funzione introdotta per mappare una data nella codifica del giorno della settimana utilizzata per LezioneCorso.

```
CREATE FUNCTION conversioneDataGiorno (
      var_data DATE)
    RETURNS ENUM('Lun', 'Mar', 'Mer', 'Gio', 'Ven', 'Sab', 'Dom')
    DETERMINISTIC
    CONTAINS SQL

BEGIN
      DECLARE var_day_name ENUM('Lun', 'Mar', 'Mer', 'Gio', 'Ven', 'Sab',
'Dom') ;
      DECLARE var_day_num INT DEFAULT DAYOFWEEK(var_data) ;

      IF var_day_num = 1 THEN SET var_day_name = 'Dom' ;
      ELSEIF var_day_num = 2 THEN SET var_day_name = 'Lun' ;
      ELSEIF var_day_num = 3 THEN SET var_day_name = 'Mar' ;
      ELSEIF var_day_num = 4 THEN SET var_day_name = 'Mer' ;
      ELSEIF var_day_num = 5 THEN SET var_day_name = 'Gio' ;
      ELSEIF var_day_num = 6 THEN SET var_day_name = 'Ven' ;
      ELSE SET var_day_name = 'Sab' ;
      END IF ;

      RETURN var_day_name ;

END
```

**verificaOrarioDurataValidi**

Funzione introdotta per la verifica della validità dell'orario inserito, della durata inserita e dello scavallamento delle lezioni al giorno successivo. Infatti il tipo TIME di MySql permette di mantenere nel campo HOUR anche valori maggiori di 24 in quanto questo tipo viene utilizzato per mantenere, oltre che l'orario, anche la durata degli eventi.
Nel dominio stiamo supponendo che la scuola non tenga delle lezioni notturne, quindi non è limitante imporre che l'orario di fine della nuova lezione non superi le 24.

```
CREATE FUNCTION `varificaOrarioDurataValidi` (
      var_orario TIME,
```

```
      var_durata INT)
      RETURNS INT
      DETERMINISTIC

BEGIN

      # Controllo validità di orario e durata
      IF (HOUR(var_orario) >= 24) THEN
            RETURN 0 ;
      ELSEIF (var_durata <= 0) THEN
            RETURN 0 ;
      ELSEIF (HOUR(var_orario + INTERVAL var_durata MINUTE) >= 24) THEN
            RETURN 0 ;
      END IF ;

      RETURN 1 ;
END
```

## Stored Procedures e transazioni

### aggiungi_allievo

```
CREATE PROCEDURE `aggiungi_allievo` (
      IN var_nome_allievo VARCHAR(100),
      IN var_telefono_allievo VARCHAR(15),
      IN var_data_iscrizione DATE,
      IN var_codice_corso INT)
BEGIN

      declare exit handler for sqlexception
      begin
            rollback; ## Annullamento Transazione
            resignal; ## Ridirezione Segnale al Client
      end;

      SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
      START TRANSACTION ;
            INSERT INTO `Allievo`(nomeAllievo, telefonoAllievo,
dataIscrizione, codiceCorso)
                  VALUES(var_nome_allievo, var_telefono_allievo,
```

```
var_data_iscrizione, var_codice_corso) ;
      COMMIT ;
END
```

Procedura per l'aggiunta di un nuovo allievo.
Si è impostata la transazione per avere una politica atomica dell'inserimento dell'allievo e dell'aggiornamento del numero degli allievi del corso a cui l'allievo si è iscritto.

**aggiungi_assenza**

```
CREATE PROCEDURE `aggiungi_assenza` (
      IN var_nome_allievo VARCHAR(100),
      IN var_data_assenza DATE,
      IN var_orario_assenza TIME)

BEGIN

      declare exit handler for sqlexception
      begin
            rollback; ## Annullamento Transazione
            resignal; ## Ridirezione Segnale al Client
      end;

      SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
      START TRANSACTION ;

            INSERT INTO Assenza
                  VALUES(var_data_assenza, var_nome_allievo,
var_orario_assenza) ;

    COMMIT ;
END
```

Procedura per l'inserimento dell'Assenza di un Allievo alla Lezione di un Corso.
Si è impostata transazione per avere atomicità rispetto al controllo fatto dai trigger sull'esistenza di una lezione corso e sull'aggiornamento del numero di assenze dell'allievo.
Il livello di isolamento si è messo a Repeatable Read per evitare che le informazioni lette nei trigger cambino nel corso della transazione.

**aggiungi_corso**

```
CREATE PROCEDURE `aggiungi_corso` (
      IN var_livello_corso VARCHAR(45),
      IN var_data_attivazione DATE,
      OUT var_codice_corso INT)
BEGIN

      declare exit handler for sqlexception
      begin
            rollback; ## Annullamento Transazione
            resignal; ## Ridirezione Segnale al Client
      end;

      SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
      START TRANSACTION ;
            INSERT INTO `Corso`(`nomeLivello`, `dataAttivazione`,
`codiceCorso`, `numeroAllievi`)
                  VALUES(var_livello_corso, var_data_attivazione, NULL,
DEFAULT) ;

            SELECT LAST_INSERT_ID() INTO var_codice_corso ;
    COMMIT ;

END
```

Procedura per l’aggiunta di un nuovo Corso.

Al termine dell’inserimento viene recuperato il codice del nuovo Corso e ritornato come parametro di OUT.

Il numero di allievi di default è 0, mentre il codice viene generato dal sistema. I parametri sono:

- var_livello_corso: livello del nuovo corso
- var_data_attivazione: data di attivazione del nuovo corso

**aggiungi_insegnante**

```
CREATE PROCEDURE `aggiungi_insegnante` (
      IN var_nome_insegnante VARCHAR(100),
      IN var_nazione_insegnante VARCHAR(100),
      IN var_indirizzo_insegnante VARCHAR(100),
```

```
      IN var_username_insegnante VARCHAR(100))
BEGIN

      declare exit handler for sqlexception
      begin
            rollback; ## Annullamento Transazione
            resignal; ## Ridirezione Segnale al Client
      end;

      SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
      START TRANSACTION ;

            INSERT INTO Utenti
                  VALUES(var_username_insegnante, SHA1('insegnante'),
'insegnante') ;

            INSERT INTO `Insegnante`
                  VALUES (var_nome_insegnante, var_nazione_insegnante,
var_indirizzo_insegnante, var_username_insegnante) ;

    COMMIT ;
END
```

Procedura per l'inserimento di un Insegnante.
All'atto dell'inserimento viene richiesto anche l'inserimento dello username associato all'Insegnante: infatti l'Insegnante è uno delle tipologie di utente nel database e quindi inserire l'insegnante richiede anche l'inserimento di un nuovo utente affinché questo nuovo insegnante possa poi accedere alle funzionalità del sistema.
Viene impostata una transazione per avere una politica all-or-nothing sull'inserimento dell'utente e dell'insegnante.
Viene impostata come password di default 'insegnante' con crittografia SHA1 e come ruolo del nuovo utente 'insegnante'.

**aggiungi_lezione**

```
CREATE PROCEDURE `aggiungi_lezione` (
      IN var_giorno_settimana ENUM('Lun', 'Mar', 'Mer', 'Gio', 'Ven',
'Sab', 'Dom'),
    IN var_orario_inizio TIME,
```

```
    IN var_codice_corso INT,
    IN var_durata INT,
    IN var_insegnante_lezione VARCHAR(100))

BEGIN

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

     SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
     START TRANSACTION ;

          IF (verificaOrarioDurataValidi(var_orario_inizio,
var_durata) != 1) THEN
               SIGNAL SQLSTATE '45009' SET MESSAGE_TEXT = 'Orario/Durata
Inseriti Non Validi' ;
          END IF ;

          INSERT INTO `LezioneCorso`
               VALUES (var_giorno_settimana, var_orario_inizio,
var_codice_corso, var_durata, var_insegnante_lezione) ;
    COMMIT ;
END
```

Procedura per l'inserimento di una LezioneCorso.

Si verifica usando la funzione *verificaOrarioDurataValidi* che l'orario e la durata inseriti siano validi.

Prima dell'inserimento della lezione vengono fatti partire tutti I trigger specificati nella sezione precedente che fanno I controlli necessari a fare si che che la nuova lezione sia consistente con l'istanza corrente del DB. Laddove si verifichi una violazione delle condizioni sugli orari o sui controlli fatti dai Trigger, la transazione viene revocata e l'errore segnalato al client.

Come livello di isolamento si è scelto SERIALIZABLE perché quello che interessa prendere al momento dell'inserimento della lezione nel sistema è il lock sulla proprietà 'collisione degli orari': scegliere serializable permette di evitare che mentre vengono eseguiti I vari controlli vengano inserite delle tuple di Lezioni che collidono ma che ricadono in controlli che sono già stati effettuati.

**aggiungi_livello**

```
CREATE PROCEDURE `aggiungi_livello` (
      IN var_nome_livello VARCHAR(45),
      IN var_nome_libro VARCHAR(45),
      IN var_has_exam BOOLEAN)
BEGIN
      INSERT INTO `Livello`(`nomeLivello`, `libro`, `esame`)
            VALUES(var_nome_livello, var_nome_libro, var_has_exam) ;
END
```

**aggiungi_partecipazione**

```
CREATE PROCEDURE `aggiungi_partecipazione` (
      IN var_codice_attivita INT,
      IN var_nome_allievo VARCHAR(100))

BEGIN
      INSERT INTO `Partecipazione`
            VALUES (var_codice_attivita, var_nome_allievo) ;
END
```

**assegna_corso**

```
CREATE PROCEDURE `assegna_corso` (
      IN var_codice_corso INT,
      IN var_nome_insegnante VARCHAR(100))
BEGIN
      INSERT INTO `Docenza`
            VALUES(var_codice_corso, var_nome_insegnante) ;
END
```

**genera_agenda**

```
CREATE PROCEDURE `genera_agenda` (
      IN var_username_insegnante VARCHAR(100),
```

```
      IN var_week_index INT)
BEGIN

      DECLARE var_insegnante VARCHAR(100) ;
      DECLARE var_week DATE DEFAULT DATE_ADD(NOW(), INTERVAL
var_week_index WEEK) ;
      DECLARE var_first_date DATE DEFAULT DATE_SUB(var_week, INTERVAL
WEEKDAY(var_week) DAY) ;
      DECLARE var_last_date DATE DEFAULT DATE_ADD(var_first_date, INTERVAL
6 DAY) ;
      DECLARE var_week_date DATE DEFAULT var_first_date ;


      DROP TABLE IF EXISTS ImpegnoInsegnante ;
      CREATE TEMPORARY TABLE ImpegnoInsegnante (
            dataImpegno DATE ,
            orarioInizio TIME ,
            orarioFine TIME ,
            tipoImpegno ENUM('C','P') , ## C = Corso ; P = Privata
            corso INT ,
            livello VARCHAR(45) ,
            allievo VARCHAR(100)) ;

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION ;

            SELECT nomeInsegnante
            FROM Insegnante
            WHERE username = var_username_insegnante
            INTO var_insegnante ;


            loop_data: LOOP
                  IF (var_week_date > var_last_date) THEN
                        LEAVE loop_data ;
                  END IF ;

                  INSERT INTO ImpegnoInsegnante
                        SELECT var_week_date, L.orarioInizio, L.orarioInizio
+ INTERVAL L.durata MINUTE, 'C', C.codiceCorso, C.nomeLivello, NULL
                        FROM LezioneCorso AS L JOIN Corso AS C ON
L.codiceCorso = C.codiceCorso
```

```
                    WHERE L.insegnanteLezione = var_insegnante AND
conversioneDataGiorno(var_week_date) = L.giornoSettimana ;

                SET var_week_date = DATE_ADD(var_week_date, INTERVAL 1
DAY) ;
          END LOOP ;


          INSERT INTO ImpegnoInsegnante
                SELECT L.dataLezione, L.orarioLezione, L.orarioLezione +
INTERVAL L.durataLezione MINUTE, 'P', NULL, NULL, L.nomeAllievo
                FROM LezionePrivata AS L
                WHERE L.nomeInsegnante = var_insegnante AND L.dataLezione
BETWEEN var_first_date AND var_last_date ;

          SET var_week_date = var_first_date ;
          select_loop : LOOP
                IF (var_week_date > var_last_date) THEN
                     LEAVE select_loop ;
                END IF ;

                SELECT *
                FROM ImpegnoInsegnante
                WHERE dataImpegno = var_week_date
                ORDER BY dataImpegno, orarioInizio ;

                SET var_week_date = DATE_ADD(var_week_date, INTERVAL 1
DAY) ;
          END LOOP ;

    COMMIT ;
    DROP TEMPORARY TABLE ImpegnoInsegnante ;

END
```

Procedura per generare l’agenda dell’insegnante.

I parametri della procedura sono:

- var_username_insegnante. Username dell’insegnante che effettua la richiesta. A partire da questo username viene recuperato il nome dell’insegnante.
- var_week_index. Indice della settimana a cui la richiesta si riferisce.

    - Ecc.
    - -1 = scorsa
    - 0 = attuale
    - 1 = prossima
    - 2 = due settimane a questa parte
    - ecc.

Vengono per prima cosa dichiarate delle variabili per permettere di ciclare sulle data.
Nello specifico:

- var_week. Data qualunque della settimana richiesta. Non importa quale essa sia perché viene utilizzata solo come supporto
- var_first_date. Prima data della settimana richiesta: calcolata partendo da var_week
- var_last_date. Ultima data della settimana richiesta: calcolata da var_first_date aggiungendo 6 giorni
- var_week_date. Variabile utilizzata per ciclare sulle date della settimana
- Si è assunto che il primo giorno della settimana sia lunedì

Viene creata una tabella temporanea per mantenere le informazioni riguardanti le lezioni della settimana:

- dataImpegno
- orarioInizio
- orarioFine
- tipoImpegno. Enum di ‘C’ per Corso e ‘P’ per Privata
- eventuali codice e livello del corso
- eventuale allievo prenotante la lezione

Si recupera il nome dell’insegnante partendo dal suo username.

Abbiamo poi un ciclo che scorre sulle date della settimana richiesta. Per ogni data della settimana richiesta si selezionano tutte le lezioni di corsi tenute da quell’insegnante e che si svolgono in un giorno della settimana uguale a quello della data attuale del ciclo. In questo caso è necessario un ciclo sulla data perché le LezioniCorso hanno come informazione temporale il loro giorno della settimana (essendo state assunte a cadenza settimanale) e non la loro data specifica.

Terminato il ciclo si selezionano tutte le LezioniPrivate impartite dall’insegnante in una data che

ricade nell'intervallo della settimana (BETWEEN comprende gli estremi).

Per finire abbiamo un altro ciclo sulle date della settimana che viene utilizzato per selezionare separatamente le lezioni di ogni data. Il risultato della procedura saranno quindi sette ResultSet, uno per ogni giorno della settimana.

Come livello di isolamento si è scelto Read Committed in quanto ci interessa che tutte le lezioni che leggiamo siano state inserite in maniera corretta rispettando tutti I vincoli, ma non ci interessa se mentre stiamo leggendo venga inserita una nuova lezione. Infatti, anche supponendo di usare un livello di isolamento maggiore, la nuova lezione, non aggiunta durante la lettura delle informazioni sarebbe aggiunta subito dopo, e quindi comunque l'utente dovrebbe rieseguire la procedura per venirne a conoscenza. Non vi sono inoltre conteggi o operatori aggregati particolari tali da causare problemi di inserimento fantasma.

**genera_report_insegnante**

```
CREATE PROCEDURE `genera_report_insegnante` (
      IN var_insegnante VARCHAR(100),
      IN var_month_index INT,
      IN var_year YEAR)

BEGIN

      DECLARE var_date DATE DEFAULT MAKEDATE(var_year, 1 +
32*var_month_index);
      DECLARE var_month_last_date DATE DEFAULT LAST_DAY(var_date);
      DECLARE var_month_first_date DATE DEFAULT
DATE_SUB(var_month_last_date, INTERVAL (DAYOFMONTH(var_month_last_date) -
1) DAY) ;

      DECLARE var_month_date DATE DEFAULT var_month_first_date ;

      declare exit handler for sqlexception
      begin
            rollback; ## Annullamento Transazione
            resignal; ## Ridirezione Segnale al Client
      end;
```

```
	DROP TABLE IF EXISTS ReportInsegnante ;
	CREATE TEMPORARY TABLE ReportInsegnante(
		dataLezione DATE,
		orarioInizio TIME,
		durata INT,
		tipoLezione ENUM('C','P')) ;


	SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
	START TRANSACTION ;

		IF (var_month_index < 0 || var_month_index > 11) THEN
			SIGNAL SQLSTATE '45059' SET MESSAGE_TEXT = 'Indice Mese
Non Valido' ;
		END IF ;

		month_date_loop: LOOP
			IF (var_month_date > var_month_last_date) THEN
				LEAVE month_date_loop ;
			END IF ;

			INSERT INTO ReportInsegnante
				SELECT var_month_date, L.orarioInizio, L.durata, 'C'
				FROM LezioneCorso AS L
				WHERE L.insegnanteLezione = var_insegnante AND
conversioneDataGiorno(var_month_date) = L.giornoSettimana ;

			SET var_month_date = DATE_ADD(var_month_date, INTERVAL 1
DAY) ;
		END LOOP ;

		INSERT INTO ReportInsegnante
			SELECT L.dataLezione, L.orarioLezione, L.durataLezione,
'P'
			FROM LezionePrivata AS L
			WHERE L.nomeInsegnante = var_insegnante AND L.dataLezione
BETWEEN var_month_first_date AND var_month_last_date ;

		SELECT *
		FROM ReportInsegnante
		ORDER BY dataLezione, orarioInizio ;
```

```
    COMMIT ;
    DROP TEMPORARY TABLE ReportInsegnante ;
END
```

Procedura per generare il report delle attività dell'insegnante.
Il funzionamento e lo schema adottato sono molto simili a quelli della procedura *genera_agenda*, con la differenza principale che qui si lavora sui giorni del mese piuttosto che su quelli della settimana.
Come parametri troviamo:

- var_insegnante. Il nome dell'insegnante di cui dobbiamo produrre il report delle attività
- var_month_index. Indice del mese di cui dobbiamo generare il report (0 = Gennaio, 1 = Febraio, ecc.)
- var_year. Numero che serve ad indicare l'anno a cui il mese si riferisce. Questa variable è stata introdotta per permettere di gestire anche situazioni in cui il mese di cui si vuole chiedere il report appartiene all'anno (inteso in senso temporale e non scolastico) precedente: ad esempio, se l'anno scolastico comincia a settembre e termina a giugno, una volta arrivati a gennaio, senza informazioni riguardanti l'anno rispetto a cui si vuole il report, sarebbe impossibile prendere le informazioni, ad esempio, di novembre

**login**

```
CREATE PROCEDURE `login`(
      IN var_username VARCHAR(100),
      IN var_password VARCHAR(45),
      OUT var_role INT)
BEGIN

      DECLARE var_enum_role
ENUM("amministrazione","segreteria","insegnante") ;

      SELECT `ruolo`
      FROM `Utenti`
      WHERE
            `username` = var_username AND
            `password` = SHA1(var_password)
      INTO var_enum_role ;
```

```
      IF var_enum_role = "amministrazione" THEN
            SET var_role = 0 ;
      ELSEIF var_enum_role = "segreteria" THEN
            SET var_role = 1 ;
      ELSEIF var_enum_role = "insegnante" THEN
            SET var_role = 2 ;
      ELSE
            SET var_role = 3 ;
      END IF ;

END
```

Procedura di login.

Viene selezionato il ruolo dell'utente di cui sono date le credenziali. Il ruolo viene poi mappato su un codice numerico tornato come OUT.

Notare che viene confrontata la password presente nel sistema con l'SHA1 della password data in input: quando viene inserito un utente, ad esempio un insegnante, la sua password non viene salvata in chiaro, ma ne viene salvato l'SHA1.

**organizza_attivita_culturale**

```
CREATE PROCEDURE `organizza_attivita_culturale` (
      IN var_data_attivita DATE,
      IN var_orario_attivita TIME,
      IN var_tipo_attivita SMALLINT,
      IN var_titolo_film VARCHAR(100),
      IN var_regista_film VARCHAR(100),
      IN var_conferenziere VARCHAR(100),
      IN var_argomento_conferenza VARCHAR(100),
      OUT var_codice_attivita INT)

BEGIN

      declare exit handler for sqlexception
      begin
            rollback; ## Annullamento Transazione
            resignal; ## Ridirezione Segnale al Client
      end;
```

```
      SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
      START TRANSACTION ;

            IF (HOUR(var_orario_attivita) >= 24) THEN
                  SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Orario Non
Valido: Orario Inizio Maggiore di 24' ;
            END IF ;

            IF (var_tipo_attivita != 0 AND var_tipo_attivita != 1) THEN
                  SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Tipo Attività
Specificato Non Valido' ;
            END IF ;

            IF var_tipo_attivita = 0 AND (var_conferenziere IS NOT NULL OR
var_argomento_conferenza IS NOT NULL OR var_titolo_film IS NULL OR
var_regista_film IS NULL) THEN
                  SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Informazioni
Non Valide Per Proiezione' ;
            ELSEIF var_tipo_attivita = 1 AND (var_conferenziere IS NULL OR
var_argomento_conferenza IS NULL OR var_titolo_film IS NOT NULL OR
var_regista_film IS NOT NULL) THEN
                  SIGNAL SQLSTATE '45004' SET MESSAGE_TEXT = 'Informazioni
Non Valide Per Conferenza' ;
            END IF ;

            INSERT INTO `AttivitaCulturale`
                  VALUES (NULL, var_data_attivita, var_orario_attivita,
var_tipo_attivita, var_titolo_film, var_regista_film, var_conferenziere,
var_argomento_conferenza) ;

            SELECT LAST_INSERT_ID() INTO var_codice_attivita ;
      COMMIT ;
END
```

Procedura per l’inserimento di una nuova AttivitaCulturale nel sistema.
Viene verificato che l’orario inserito non sia superiore a 24: qualora questo non venga soddisfatto la transazione viene abortita.
Viene verificato che il tipo di attività indicato sia un campo valido (ovvero 0 per proiezioni e 1 per conferenze).
Viene verificato che le informazioni siano coerenti con il tipo di attività impostata.

Dopo l'inserimento viene recuperato il codice della nuova attività inserita come parametro di OUT.

**prenota_lezione_privata**

```
CREATE PROCEDURE `prenota_lezione_privata` (
      IN var_data DATE,
      IN var_orario TIME,
      IN var_insegnante VARCHAR(100),
      IN var_durata INT,
      IN var_allievo VARCHAR(100))
BEGIN

      declare exit handler for sqlexception
      begin
            rollback; ## Annullamento Transazione
            resignal; ## Ridirezione Segnale al Client
      end;

      SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
      START TRANSACTION ;

            IF (verificaOrarioDurataValidi(var_orario, var_durata) != 1)
THEN
                  SIGNAL SQLSTATE '45049' SET MESSAGE_TEXT = 'Orario/Durata
Inseriti Non Validi' ;
            END IF ;

            INSERT INTO LezionePrivata
                  VALUES (var_data, var_orario, var_insegnante, var_durata,
var_allievo) ;
    COMMIT ;
END
```

Procedura per inserimeno LezionePrivata.

Viene verificata la correttezza dell'orario e della data inseriti.

Prima di eseguire l'inserimento vengono avviati tutti I trigger per il controllo dei conflitti che possono incorrere all'inserimento di una lezione privata: se uno qualsiasi di questi controlli fallisce la transazione va in abort, altrimenti la LezionePrivata viene inserita.

Come livello di isolamento si è scelto SERIALIZABLE perché quello che interessa prendere al momento dell'inserimento della lezione nel sistema è il lock sulla proprietà 'collisione degli orari':

scegliere serializable permette di evitare che mentre vengono eseguiti I vari controlli vengano inserite delle tuple di Lezioni che collidono ma che ricadono in controlli che sono già stati effettuati.

**recupera_attivita**

```
CREATE PROCEDURE `recupera_attivita` ()
BEGIN

      SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
      START TRANSACTION ;
            SELECT *
            FROM AttivitaCulturale
            WHERE dataAttivita > NOW()
            ORDER BY dataAttivita, orarioAttivita ;
      COMMIT ;
END
```

**recupera_corsi**

```
CREATE PROCEDURE `recupera_corsi` ()
BEGIN
      SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
      START TRANSACTION ;
            SELECT *
            FROM `Corso`
            ORDER BY nomeLivello, codiceCorso;
      COMMIT ;
END
```

**recupera_docenze**

```
CREATE PROCEDURE `recupera_docenze` ()
BEGIN
      SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
      START TRANSACTION ;
            SELECT D.codiceCorso, C.nomeLivello, D.nomeInsegnante FROM
Docenza AS D JOIN Corso AS C ON D.codiceCorso = C.codiceCorso
            ORDER BY C.nomeLivello, D.codiceCorso ;
```

```
      COMMIT ;
END
```

**recupera_insegnanti**

```
CREATE PROCEDURE `recupera_insegnanti` ()
BEGIN

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
    START TRANSACTION ;
           SELECT nomeInsegnante, nazioneInsegnante, indirizzoInsegnante
           FROM Insegnante
           ORDER BY nomeInsegnante ;
    COMMIT ;
END
```

**recupera_insegnanti_liberi**

```
CREATE PROCEDURE `recupera_insegnanti_liberi` (
      IN var_data DATE,
      IN var_orario TIME,
      IN var_durata INT)
BEGIN

      SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
      START TRANSACTION ;

           IF (verificaOrarioDurataValidi(var_orario, var_durata) != 1)
THEN
                SIGNAL SQLSTATE '45009' SET MESSAGE_TEXT = 'Orario/Durata
Inseriti Non Validi' ;
           END IF ;

           SELECT I.nomeInsegnante
           FROM Insegnante AS I
           WHERE
                I.nomeInsegnante NOT IN (
                     SELECT insegnanteLezione
                     FROM LezioneCorso
```

```
                        WHERE
                              giornoSettimana =
conversioneDataGiorno(var_data) AND
                              NOT verifica_orari_compatibili(var_orario,
var_orario + INTERVAL var_durata MINUTE, orarioInizio, orarioInizio +
INTERVAL durata MINUTE)
                        )
                  AND
                  I.nomeInsegnante NOT IN (
                        SELECT nomeInsegnante
                        FROM LezionePrivata
                        WHERE
                              var_data = dataLezione AND
                              NOT verifica_orari_compatibili(var_orario,
var_orario + INTERVAL var_durata MINUTE, orarioLezione, orarioLezione +
INTERVAL durataLezione MINUTE)
                        ) ;

      COMMIT ;

END
```

Procedura per il recupero di insegnanti liberi in una certa fascia oraria. Questa procedura è stata introdotta principalmente come supporto all'operazione di *prenota_lezione_privata()* per evitare che colui che la esegue sia costretto ad andare a tentativi finché non riesce a trovare una fascia oraria libera con quell'insegnante.
I parametri sono le informazioni temporali nella cui fascia si chiedono insegnanti liberi.
Si selezionano quindi I nomi degli insegnanti qualora questi non siano:

- Tra gli insegnanti che hanno lezioni corso in quella fascia e nel giorno della settimana di quella data
- Tra gli insegnanti che hanno lezioni private in quella fascia oraria e in quella data

Il livello di isolamento si è posto a Read Committed per gli stessi motivi di *genera_agenda*

**recupera_livelli**

```
CREATE PROCEDURE `recupera_livelli` ()
BEGIN

    SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
```

```
    START TRANSACTION ;
          SELECT *
          FROM Livello
          ORDER BY nomeLivello ;
    COMMIT ;
END
```

**report_assenze_corso**

```
CREATE PROCEDURE `report_assenze_corso` (IN var_codice_corso INT)
BEGIN

     SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
     START TRANSACTION ;
          SELECT nomeAllievo, numeroAssenze
          FROM Allievo
          WHERE codiceCorso = var_codice_corso ;
     COMMIT ;
END
```

**riavvia_anno**

```
CREATE PROCEDURE `riavvia_anno` ()
BEGIN

     declare exit handler for sqlexception
     begin
          rollback; ## Annullamento Transazione
          resignal; ## Ridirezione Segnale al Client
     end;

     SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
     START TRANSACTION ;
          DELETE FROM `Partecipazione` ;
          DELETE FROM `AttivitaCulturale` ;
          DELETE FROM `Assenza` ;
          DELETE FROM `LezioneCorso` ;
          DELETE FROM `Docenza` ;
          DELETE FROM `LezionePrivata` ;
```

```
            DELETE FROM `Allievo` ;
            DELETE FROM `Insegnante` ;

            DELETE FROM `Corso` ;
            DELETE FROM `Livello` ;

            DELETE FROM `Utenti`
                  WHERE `username` <> 'amministrazione' AND  `username` <>
'segreteria' ;
      COMMIT ;
END
```

Procedura per il riavvio del sistema.

Vengono eliminate le tuple in tutte le tabelle. Le uniche tuple che non vengono eliminate sono quelle dell'utenza 'amministrazione' e 'segreteria' della tabella Utenti: queste infatti sono le tuple che permettono l'identificazione a quella tipologia di utenti e che sono introdotti alla creazione del DB.

Il livello di Isolamento viene posto a Serializable per evitare che mentre viene effettuato il riavvio, altri utenti possano accedere inserendo dei dati in tabelle che sono già state cancellate: livelli di isolamento inferiori infatti prenderebbero il lock sulla singola tupla e non sull'intera relazione, causando incoerenza.

Inoltre si utilizza una transazione per fare in modo che lo stato sia ripristinato a quello di partenza qualora vi fossero dei problemi in una qualche cancellazione.

# Appendice: Implementazione

## Codice SQL per instanziare il database

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_
DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema my_english_school
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `my_english_school` ;

-- -----------------------------------------------------
-- Schema my_english_school
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `my_english_school` DEFAULT CHARACTER SET
utf8mb4 COLLATE utf8mb4_0900_ai_ci ;
USE `my_english_school` ;

-- -----------------------------------------------------
-- Table `my_english_school`.`Livello`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`Livello` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`Livello` (
  `nomeLivello` VARCHAR(45) NOT NULL,
  `libro` VARCHAR(45) NOT NULL,
```

```
  `esame` SMALLINT NOT NULL,
  PRIMARY KEY (`nomeLivello`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `my_english_school`.`Corso`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`Corso` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`Corso` (
  `codiceCorso` INT NOT NULL AUTO_INCREMENT,
  `nomeLivello` VARCHAR(45) NOT NULL,
  `dataAttivazione` DATE NOT NULL,
  `numeroAllievi` INT NOT NULL DEFAULT '0',
  PRIMARY KEY (`codiceCorso`),
  INDEX `fk_Corso_1_idx` (`nomeLivello` ASC) VISIBLE,
  CONSTRAINT `fk_Corso_1`
    FOREIGN KEY (`nomeLivello`)
    REFERENCES `my_english_school`.`Livello` (`nomeLivello`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `my_english_school`.`Allievo`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`Allievo` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`Allievo` (
  `nomeAllievo` VARCHAR(100) NOT NULL,
  `telefonoAllievo` VARCHAR(15) NOT NULL,
  `numeroAssenze` INT NOT NULL DEFAULT 0,
  `dataIscrizione` DATE NOT NULL,
  `codiceCorso` INT NOT NULL,
  PRIMARY KEY (`nomeAllievo`),
  INDEX `fk_Allievo_1_idx` (`codiceCorso` ASC) VISIBLE,
  CONSTRAINT `fk_Allievo_1`
    FOREIGN KEY (`codiceCorso`)
    REFERENCES `my_english_school`.`Corso` (`codiceCorso`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `my_english_school`.`Assenza`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`Assenza` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`Assenza` (
  `dataAssenza` DATE NOT NULL,
  `nomeAllievo` VARCHAR(100) NOT NULL,
  `orarioAssenza` TIME NOT NULL,
  INDEX `fk_Assenza_1_idx` (`nomeAllievo` ASC) VISIBLE,
```

```
  PRIMARY KEY (`nomeAllievo`, `orarioAssenza`, `dataAssenza`),
  CONSTRAINT `fk_Assenza_1`
    FOREIGN KEY (`nomeAllievo`)
    REFERENCES `my_english_school`.`Allievo` (`nomeAllievo`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `my_english_school`.`AttivitaCulturale`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`AttivitaCulturale` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`AttivitaCulturale` (
  `codiceAttivita` INT NOT NULL AUTO_INCREMENT,
  `dataAttivita` DATE NOT NULL,
  `orarioAttivita` TIME NOT NULL,
  `tipoAttivita` SMALLINT NOT NULL COMMENT 'tipoAttività.   - 0 <---> 
proiezione\n                      - 1 <---> conferenza',
  `titoloFilm` VARCHAR(100) NULL,
  `registaFilm` VARCHAR(100) NULL,
  `conferenziere` VARCHAR(100) NULL,
  `argomentoConferenza` VARCHAR(100) NULL,
  PRIMARY KEY (`codiceAttivita`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `my_english_school`.`Utenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`Utenti` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`Utenti` (
  `username` VARCHAR(100) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `ruolo` ENUM('amministrazione', 'segreteria', 'insegnante') NOT NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `my_english_school`.`Insegnante`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`Insegnante` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`Insegnante` (
  `nomeInsegnante` VARCHAR(100) NOT NULL,
  `nazioneInsegnante` VARCHAR(45) NOT NULL,
  `indirizzoInsegnante` VARCHAR(100) NOT NULL,
  `username` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`nomeInsegnante`),
  INDEX `fk_Insegnante_1_idx` (`username` ASC) VISIBLE,
  UNIQUE INDEX `username_UNIQUE` (`username` ASC) VISIBLE,
  CONSTRAINT `fk_Insegnante_1`
    FOREIGN KEY (`username`)
```

```
    REFERENCES `my_english_school`.`Utenti` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `my_english_school`.`Docenza`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`Docenza` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`Docenza` (
  `codiceCorso` INT NOT NULL,
  `nomeInsegnante` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`codiceCorso`, `nomeInsegnante`),
  INDEX `fk_Docenza_1_idx` (`nomeInsegnante` ASC) VISIBLE,
  INDEX `fk_Docenza_2_idx` (`codiceCorso` ASC) VISIBLE,
  CONSTRAINT `fk_Docenza_1`
    FOREIGN KEY (`nomeInsegnante`)
    REFERENCES `my_english_school`.`Insegnante` (`nomeInsegnante`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_Docenza_2`
    FOREIGN KEY (`codiceCorso`)
    REFERENCES `my_english_school`.`Corso` (`codiceCorso`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;



-- -----------------------------------------------------
-- Table `my_english_school`.`LezioneCorso`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`LezioneCorso` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`LezioneCorso` (
  `giornoSettimana` ENUM('Lun', 'Mar', 'Mer', 'Gio', 'Ven', 'Sab', 'Dom')
NOT NULL,
  `orarioInizio` TIME NOT NULL,
  `codiceCorso` INT NOT NULL,
  `durata` INT NOT NULL,
  `insegnanteLezione` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`giornoSettimana`, `orarioInizio`, `codiceCorso`),
  INDEX `fk_LezioneCorso_1_idx` (`codiceCorso` ASC) VISIBLE,
  INDEX `fk_LezioneCorso_2_idx` (`insegnanteLezione` ASC) VISIBLE,
  CONSTRAINT `fk_LezioneCorso_1`
    FOREIGN KEY (`codiceCorso`)
    REFERENCES `my_english_school`.`Corso` (`codiceCorso`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_LezioneCorso_2`
    FOREIGN KEY (`insegnanteLezione`)
    REFERENCES `my_english_school`.`Insegnante` (`nomeInsegnante`)
    ON DELETE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `my_english_school`.`LezionePrivata`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`LezionePrivata` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`LezionePrivata` (
  `dataLezione` DATE NOT NULL,
  `orarioLezione` TIME NOT NULL,
  `nomeInsegnante` VARCHAR(100) NOT NULL,
  `durataLezione` INT NOT NULL,
  `nomeAllievo` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`dataLezione`, `orarioLezione`, `nomeInsegnante`),
  INDEX `fk_LezionePrivata_1_idx` (`nomeInsegnante` ASC) VISIBLE,
  INDEX `fk_LezionePrivata_2_idx` (`nomeAllievo` ASC) VISIBLE,
  CONSTRAINT `fk_LezionePrivata_1`
    FOREIGN KEY (`nomeInsegnante`)
    REFERENCES `my_english_school`.`Insegnante` (`nomeInsegnante`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_LezionePrivata_2`
    FOREIGN KEY (`nomeAllievo`)
    REFERENCES `my_english_school`.`Allievo` (`nomeAllievo`)
    ON DELETE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;


-- -----------------------------------------------------
-- Table `my_english_school`.`Partecipazione`
```

```
-- -----------------------------------------------------
DROP TABLE IF EXISTS `my_english_school`.`Partecipazione` ;

CREATE TABLE IF NOT EXISTS `my_english_school`.`Partecipazione` (
  `codiceAttivita` INT NOT NULL,
  `nomeAllievo` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`codiceAttivita`, `nomeAllievo`),
  INDEX `fk_Partecipazione_2_idx` (`nomeAllievo` ASC) VISIBLE,
  INDEX `fk_Partecipazione_1_idx` (`codiceAttivita` ASC) VISIBLE,
  CONSTRAINT `fk_Partecipazione_1`
    FOREIGN KEY (`codiceAttivita`)
    REFERENCES `my_english_school`.`AttivitaCulturale` (`codiceAttivita`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
  CONSTRAINT `fk_Partecipazione_2`
    FOREIGN KEY (`nomeAllievo`)
    REFERENCES `my_english_school`.`Allievo` (`nomeAllievo`)
    ON DELETE RESTRICT)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

USE `my_english_school` ;



USE `my_english_school`;

DELIMITER $$

USE `my_english_school`$$
DROP TRIGGER IF EXISTS `my_english_school`.`Allievo_AFTER_INSERT` $$
```

```
USE `my_english_school`$$

CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`Allievo_AFTER_INSERT` AFTER INSERT ON `Allievo` FOR
EACH ROW


BEGIN

    UPDATE `Corso`

    SET Corso.numeroAllievi = Corso.numeroAllievi + 1

    WHERE Corso.codiceCorso = new.codiceCorso ;

END$$



USE `my_english_school`$$

DROP TRIGGER IF EXISTS `my_english_school`.`Assenza_BEFORE_INSERT` $$

USE `my_english_school`$$

CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`Assenza_BEFORE_INSERT` BEFORE INSERT ON `Assenza`
FOR EACH ROW

BEGIN


     DECLARE var_esiste_lezione INT DEFAULT 0 ;

    DECLARE var_codice_corso INT ;


    SELECT codiceCorso

    FROM Allievo

    WHERE nomeAllievo = new.nomeAllievo

    INTO var_codice_corso ;


    SELECT count(*)

    FROM LezioneCorso

    WHERE

           codiceCorso = var_codice_corso AND
```

```
            giornoSettimana = conversioneDataGiorno(new.dataAssenza) AND
         orarioInizio = new.orarioAssenza
       INTO var_esiste_lezione ;

    IF (var_esiste_lezione = 0) THEN
            SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = "Non esistono
Lezioni del Corso di Allievo che combacino" ;
    END IF ;

END$$



USE `my_english_school`$$
DROP TRIGGER IF EXISTS `my_english_school`.`Assenza_AFTER_INSERT` $$
USE `my_english_school`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`Assenza_AFTER_INSERT` AFTER INSERT ON `Assenza` FOR
EACH ROW
BEGIN
      UPDATE Allievo
    SET numeroAssenze = numeroAssenze + 1
    WHERE nomeAllievo = new.nomeAllievo;
END$$



USE `my_english_school`$$
DROP TRIGGER IF EXISTS `my_english_school`.`LezioneCorso_BEFORE_INSERT` $
$
USE `my_english_school`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezioneCorso_BEFORE_INSERT` BEFORE INSERT ON
`LezioneCorso` FOR EACH ROW
BEGIN
```

```
      #CONTROLLO NON CONFLITTO DI ORARI CON ALTRE LEZIONI DI CORSI


    DECLARE var_has_conflict INT DEFAULT 0;


    SELECT count(*)
    FROM LezioneCorso
    WHERE
           insegnantelezione = new.insegnanteLezione AND
        giornoSettimana = new.giornoSettimana AND
        NOT verifica_orari_compatibili(new.orarioInizio, new.orarioInizio
+ INTERVAL new.durata MINUTE, orarioInizio, orarioInizio + INTERVAL
durata MINUTE)
      INTO var_has_conflict ;


    IF var_has_conflict > 0 THEN
           SIGNAL SQLSTATE '45005' SET MESSAGE_TEXT = 'Insegnante ha una
lezione di un altro corso' ;
    END IF ;


END$$



USE `my_english_school`$$
DROP TRIGGER IF EXISTS `my_english_school`.`LezioneCorso_BEFORE_INSERT_1`
$$
USE `my_english_school`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezioneCorso_BEFORE_INSERT_1` BEFORE INSERT ON
`LezioneCorso` FOR EACH ROW
BEGIN


      ## Controllo Docenza Assegnata
      DECLARE var_docenze INT DEFAULT 0;
```

```
    SELECT count(*)
    FROM Docenza
    WHERE Docenza.codiceCorso = NEW.codiceCorso AND
Docenza.nomeInsegnante = NEW.insegnanteLezione
    INTO var_docenze ;

    IF (var_docenze <> 1) THEN
           SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'Insegnante NON
assegnato al Corso Indicato' ;
     END IF ;
END$$


USE `my_english_school`$$
DROP TRIGGER IF EXISTS `my_english_school`.`LezioneCorso_BEFORE_INSERT_2`
$$
USE `my_english_school`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezioneCorso_BEFORE_INSERT_2` BEFORE INSERT ON
`LezioneCorso` FOR EACH ROW
BEGIN
     ## CONTROLLO CONFLITTO DELLA LEZIONE CON ALTRE LEZIONI PRIVATE DI
INSEGNANTE

    DECLARE var_has_conflict INT DEFAULT 0 ;

    SELECT count(*)
    FROM LezionePrivata AS P
    WHERE
           nomeInsegnante = new.insegnanteLezione
        AND dataLezione >= NOW() AND
        new.giornoSettimana = conversioneDataGiorno(dataLezione) AND
```

```
        NOT verifica_orari_compatibili(new.orarioInizio, new.orarioInizio
+ INTERVAL new.durata MINUTE, P.orarioLezione, P.orarioLezione + INTERVAL
P.durataLezione MINUTE)
      INTO var_has_conflict ;

    IF var_has_conflict > 0 THEN
           SIGNAL SQLSTATE '45006' SET MESSAGE_TEXT = 'Insegnante ha una
Lezione Privata in conflitto' ;
    END IF ;
END$$



USE `my_english_school`$$
DROP TRIGGER IF EXISTS `my_english_school`.`LezioneCorso_BEFORE_INSERT_3`
$$
USE `my_english_school`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezioneCorso_BEFORE_INSERT_3` BEFORE INSERT ON
`LezioneCorso` FOR EACH ROW
BEGIN
      DECLARE var_has_conflict INT DEFAULT 0 ;

    SELECT count(*)
    FROM LezioneCorso
    WHERE
           codiceCorso = new.codiceCorso AND
           giornoSettimana = new.giornoSettimana AND
        NOT verifica_orari_compatibili(new.orarioInizio, new.orarioInizio
+ INTERVAL new.durata MINUTE, orarioInizio, orarioInizio + INTERVAL
new.durata MINUTE)
      INTO var_has_conflict ;

      IF var_has_conflict > 0 THEN
```

```
            SIGNAL SQLSTATE '45013' SET MESSAGE_TEXT = 'Esiste una lezione
del corso in un orario che collide' ;
    END IF ;
END$$



USE `my_english_school`$$
DROP TRIGGER IF EXISTS `my_english_school`.`LezionePrivata_BEFORE_INSERT`
$$
USE `my_english_school`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezionePrivata_BEFORE_INSERT` BEFORE INSERT ON
`LezionePrivata` FOR EACH ROW
BEGIN
     ## Controllo NO Conflitto con Lezioni Corso di Insegnante

    DECLARE var_has_conflict INT DEFAULT 0 ;

    SELECT count(*)
    FROM LezioneCorso
    WHERE
            insegnanteLezione = new.nomeInsegnante AND
            giornoSettimana = conversioneDataGiorno(new.dataLezione) AND
        NOT verifica_orari_compatibili(new.orarioLezione,
new.orarioLezione + INTERVAL new.durataLezione MINUTE, orarioInizio,
orarioInizio + INTERVAL durata MINUTE)
    INTO var_has_conflict ;

    IF (var_has_conflict <> 0) THEN
            SIGNAL SQLSTATE '45009' SET MESSAGE_TEXT = 'La Lezione Collide
Con La Lezione di Qualche Corso' ;
    END IF ;
END$$
```

```
USE `my_english_school`$$

DROP TRIGGER IF EXISTS
`my_english_school`.`LezionePrivata_BEFORE_INSERT_1` $$

USE `my_english_school`$$

CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezionePrivata_BEFORE_INSERT_1` BEFORE INSERT ON
`LezionePrivata` FOR EACH ROW

BEGIN

    ## Controllo NO Conflitto con Lezioni Private di Insegnante


     DECLARE var_has_conflict INT DEFAULT 0 ;


    SELECT count(*)

    FROM LezionePrivata

    WHERE

           dataLezione = new.dataLezione AND

        nomeInsegnante = new.nomeInsegnante AND

           NOT verifica_orari_compatibili(new.orarioLezione,
new.orarioLezione + INTERVAL new.durataLezione MINUTE, orarioLezione,
orarioLezione + INTERVAL durataLezione MINUTE)

     INTO var_has_conflict ;


     IF (var_has_conflict > 0) THEN

           SIGNAL SQLSTATE '45011' SET MESSAGE_TEXT = 'La Lezione Collide
Con Altre Lezioni Private Dello Stesso Insegnante' ;

     END IF ;

END$$



USE `my_english_school`$$

DROP TRIGGER IF EXISTS
```

```
`my_english_school`.`LezionePrivata_BEFORE_INSERT_2` $$

USE `my_english_school`$$

CREATE DEFINER = CURRENT_USER TRIGGER
`my_english_school`.`LezionePrivata_BEFORE_INSERT_2` BEFORE INSERT ON
`LezionePrivata` FOR EACH ROW

BEGIN

     ## Controllo Stesso Studente non ha più lezioni in orari che
collidono

    DECLARE var_has_conflict INT DEFAULT 0 ;


    ## CONTO IL NUMERO DI LEZIONI CHE COLLIDONO

    SELECT count(*)

    FROM LezionePrivata

    WHERE nomeAllievo = NEW.nomeAllievo AND

           dataLezione = new.dataLezione AND

           NOT verifica_orari_compatibili(new.orarioLezione,
new.orarioLezione + INTERVAL new.durataLezione MINUTE, orarioLezione,
orarioLezione + INTERVAL durataLezione MINUTE)

    INTO var_has_conflict ;


    IF (var_has_conflict > 0) THEN

           SIGNAL SQLSTATE '45007' SET MESSAGE_TEXT = 'Alunno ha già
prenotato una lezione in un orario che collide' ;

    END IF ;

END$$



DELIMITER ;

SET SQL_MODE = '';

DROP USER IF EXISTS login;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_
DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
```

```
CREATE USER 'login' IDENTIFIED BY 'Login@1234';


GRANT EXECUTE ON procedure `my_english_school`.`login` TO 'login';

SET SQL_MODE = '';

DROP USER IF EXISTS amministrazione;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_
DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'amministrazione' IDENTIFIED BY 'Amministrazione@1234';


GRANT EXECUTE ON procedure `my_english_school`.`riavvia_anno` TO
'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`aggiungi_livello` TO
'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`aggiungi_corso` TO
'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`aggiungi_insegnante` TO
'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`assegna_corso` TO
'amministrazione';

GRANT EXECUTE ON procedure
`my_english_school`.`organizza_attivita_culturale` TO 'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`recupera_corsi` TO
'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`aggiungi_lezione` TO
'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`recupera_docenze` TO
'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`genera_report_insegnante`
TO 'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`recupera_insegnanti` TO
'amministrazione';

GRANT EXECUTE ON procedure `my_english_school`.`recupera_livelli` TO
'amministrazione';

SET SQL_MODE = '';
```

```
DROP USER IF EXISTS segreteria;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_
DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'segreteria' IDENTIFIED BY 'Segreteria@1234';


GRANT EXECUTE ON procedure `my_english_school`.`aggiungi_allievo` TO
'segreteria';

GRANT EXECUTE ON procedure `my_english_school`.`aggiungi_partecipazione`
TO 'segreteria';

GRANT EXECUTE ON procedure `my_english_school`.`recupera_corsi` TO
'segreteria';

GRANT EXECUTE ON procedure `my_english_school`.`recupera_attivita` TO
'segreteria';

GRANT EXECUTE ON procedure `my_english_school`.`aggiungi_assenza` TO
'segreteria';

GRANT EXECUTE ON procedure `my_english_school`.`prenota_lezione_privata`
TO 'segreteria';

GRANT EXECUTE ON procedure `my_english_school`.`report_assenze_corso` TO
'segreteria';

GRANT EXECUTE ON procedure
`my_english_school`.`recupera_insegnanti_liberi` TO 'segreteria';

SET SQL_MODE = '';

DROP USER IF EXISTS insegnante;

SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_
DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'insegnante' IDENTIFIED BY 'Insegnante@1234';


GRANT EXECUTE ON procedure `my_english_school`.`genera_agenda` TO
'insegnante';


SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
```

```
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;


-- -----------------------------------------------------
-- Data for table `my_english_school`.`Utenti`
-- -----------------------------------------------------
START TRANSACTION;
USE `my_english_school`;
INSERT INTO `my_english_school`.`Utenti` (`username`, `password`,
`ruolo`) VALUES ('amministrazione',
'156b7dfafeeef4435ece5367c0e5a5cfd5ae1390', 'amministrazione');
INSERT INTO `my_english_school`.`Utenti` (`username`, `password`,
`ruolo`) VALUES ('segreteria',
'02cad8ee4fb1cc60f6a485316007c9d709293cbd', 'segreteria');


COMMIT;
```

## Codice del Front-End

### config

EnvironmentSetter.c

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

#include "EnvironmentSetter.h"

const char *CONFIGURATION_FILE_NAME =
"src/code/config/configuration.properties" ;

bool setEnvironmentVariable(char *environmentToken) ;

bool loadConfiguration() {
    // Open configuration file to prepare env
    FILE *configurationFile = fopen(CONFIGURATION_FILE_NAME, "r") ;
    if (configurationFile == NULL) {
```

```
        fprintf(stderr, "Errore apertura file\n") ;
        return false ;
    }

    char *environmentLine = NULL ;
    size_t len ;
    ssize_t nread = getline(&environmentLine, &len, configurationFile) ;
    while (nread > 0) {
        char *environmentToken = strtok(environmentLine, "\n") ;

        if (setEnvironmentVariable(environmentToken) == false) {
            free(environmentLine) ;
            fclose(configurationFile) ;
            return false ;
        }

        nread = getline(&environmentLine, &len, configurationFile) ;
    }
    //getline alloca memoria quando viene passata un NULL come puntatore
--> libero memoria allocata
    free(environmentLine) ;

    fclose(configurationFile) ;
    return true ;
}

bool setEnvironmentVariable(char *environmentToken) {
    //Function to parse Environment Variable from configuration file
    char *tokenName = strtok(environmentToken, "=") ;
    char *tokenValue = strtok(NULL, "=") ;

    if (setenv(tokenName, tokenValue, 0)) {
        printf("Errore Impostazione Variabile d'Ambiente\n") ;
        return false ;
    }
    return true ;
}
```

EnvironmentSetter.h

```
#pragma once
#include <stdbool.h>
```

```
bool loadConfiguration() ;
```

configuration.properties

```
DB.HOST=localhost
DB.PORT=3306
DB.NAME=my_english_school
LOGIN.USER=login
LOGIN.PASSWD=Login@1234
AMMINISTRAZIONE.USER=amministrazione
AMMINISTRAZIONE.PASSWD=Amministrazione@1234
SEGRETERIA.USER=segreteria
SEGRETERIA.PASSWD=Segreteria@1234
INSEGNANTE.USER=insegnante
INSEGNANTE.PASSWD=Insegnante@1234
```

**controller**

AdministrationController.c

```
#include "AdministrationControllerHeader.h"

#include "../model/Level.h"
#include "../model/Activity.h"
#include "../model/Class.h"
#include "../model/Student.h"
#include "../model/Teacher.h"
#include "../model/Lesson.h"
#include "../model/User.h"

#include "../db/AdministrationDatabaseHeader.h"
#include "../db/CommonDBHeader.h"

#include "../view/AdministrationViewHeader.h"
#include "../view/ViewUtilsHeader.h"


enum AdministrationControllerOptions {
    ADD_LEVEL = 0,
    ADD_CLASS,
    ADD_TEACHER,
    ASSIGN_CLASS,
    ADD_LESSON,
```

```
    ORGANIZE_ACTIVITY,
    TEACHER_REPORT,
    RESTART_YEAR,
    ADMINISTRATION_QUIT,
} ;

void getAllTeachers() {
    DatabaseResult *teachers = selectAllTeachers() ;

    if (teachers == NULL) return ;
    printAllTeachers((Teacher **) teachers->rowsSet, teachers->numRows) ;

    freeDatabaseResult(teachers) ;
}


void addLevel() {
    Level level ;
    if (getLevelInfo(&level)) {
        addLevelToDatabase(&level) ;
    }
}

void addClass() {
    Class newClass ;

    DatabaseResult *result = selectAllLevels() ;
    if (result != NULL) {
        printAllLevels((Level **) result->rowsSet, result->numRows) ;
        freeDatabaseResult(result) ;
    }

    if (!getClassInfo(&newClass)) return ;

    int *newClassCode = addClassToDatabase(&newClass) ;

    if (newClassCode == NULL) return ;

    printNewClassCode(newClassCode) ;

    free(newClassCode) ;
}
```

```
void addTeacher() {
    Teacher newTeacher ;
    char teacherUsername[USERNAME_MAX_SIZE] ;
    if (getTeacherInfo(&newTeacher, teacherUsername)) {
        addTeacherToDatabase(&newTeacher, teacherUsername) ;
    }
}

void assignClass() {
    Teacher teacher ;
    Class class ;

    getAllTeachers() ;
    DatabaseResult *courses = selectAllCourses() ;
    if (courses != NULL) {
        printAllCourses((Class **) courses->rowsSet, courses->numRows) ;
        freeDatabaseResult(courses) ;
    }
    if (getTeacherAndClassInfo(&teacher, &class)) {
        assignTeacherToClass(&teacher, &class) ;
    }
}

void organizeActivity() {
    CuturalActivity newActivity ;
    if (! getActivityInfo(&newActivity)) return ;

    int *activityCode = organizeActivityInDatabase(&newActivity) ;

    printNewActivityCode(activityCode) ;

    free(activityCode) ;
}

void getAllTeaching() {
    DatabaseResult *result = selectAllTeaching() ;
    if (result == NULL) return ;

    printAllTeaching((Teaching **) result->rowsSet, result->numRows) ;

    freeDatabaseResult(result) ;
```

```
}

void addLesson() {
    getAllTeaching() ;
    ClassLesson newLesson ;
    if (getCourseLessonInfo(&newLesson)) {
        addClassLessonToDatabase(&newLesson) ;
    }
}

void generateTeacherReport() {
    char teacherName[TEACHER_NAME_MAX_LEN] ;
    int year ;
    int monthIndex ;

    getAllTeachers() ;
    if (!getTeacherReportInfo(teacherName, &year, &monthIndex)) return ;

    DatabaseResult *result = generateTeacherReportFromDB(teacherName,
&year, &monthIndex) ;
    if (result == NULL) return ;

    printTeacherReport((ReportLesson **) result->rowsSet, result-
>numRows) ;

    freeDatabaseResult(result) ;
}


void restartYear() {
    if (!askRestartConfirm()) return ;
    if (!restartYearDB()) return ;

    printRestartSuccess() ;
}

void administrationController() {

    do {
        int selectedOption = getAdministrationOption() ;

        showOptionHeader() ;
```

```
        switch (selectedOption) {
            case ADD_LEVEL :
                addLevel() ;
                break ;

            case ADD_CLASS :
                addClass() ;
                break ;

            case ADD_TEACHER :
                addTeacher() ;
                break ;

            case ASSIGN_CLASS :
                assignClass() ;
                break ;

            case ADD_LESSON :
                addLesson() ;
                break ;

            case ORGANIZE_ACTIVITY :
                organizeActivity() ;
                break ;

            case TEACHER_REPORT :
                generateTeacherReport() ;
                break ;

            case RESTART_YEAR :
                restartYear() ;
                break ;

            case ADMINISTRATION_QUIT :
                goto exit_loop ;

            default :
                printf("INVALID OPTION\n\n") ;
                break ;
        }
        showOptionHeader() ;
    } while (true) ;
```

```
    exit_loop:
    showOptionHeader() ;
}
```

AdministrationControllerHeader.h

```
#pragma once

#include <stdbool.h>
#include <time.h>
#include <stdbool.h>
#include <stdio.h>

#include "../utils/TimeUtils.h"

void administrationController() ;
```

LoginController.c

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

#include "../view/LoginView.h"
#include "LoginControllerHeader.h"
#include "../db/DatabaseLoginHeader.h"
#include "../utils/SystemUtilsHeader.h"
#include "../utils/IOUtils.h"
#include "AdministrationControllerHeader.h"
#include "SecretaryControllerHeader.h"
#include "TeacherControllerHeader.h"


void successLogin(Role loginRole, char *username) ;


void loginController() {
    User loginCredentials ;


    do {
        memset(&loginCredentials, 0, sizeof(User)) ;
```

```
        Role loginRole = LOGIN ;

        if (!showLoginView(&loginCredentials)) {
            loginRole = LOGIN ;
        }
        else {
            loginRole = attemptLogin(&loginCredentials) ;
            if (loginRole == LOGIN) {
                colorPrint("Username e/o Password Non Valida\n\n",
RED_TEXT) ;
            }
            else {
                successLogin(loginRole, loginCredentials.username) ;
            }
        }
    } while (true) ;
}

void successLogin(Role loginRole, char *username) {
    clearScreen() ;
    showAppHeader() ;

    if (switchRole(loginRole) == false) {
        return ;
    }

    switch (loginRole) {
        case AMMINISTRAZIONE :
            administrationController() ;
            break ;
        case SEGRETERIA :
            secretaryController() ;
            break ;
        case INSEGNANTE :
            teacherController(username) ;
            break ;
        case LOGIN :
            break ;
    }

    switchRole(LOGIN) ;
```

```
    clearScreen() ;
    showAppHeader() ;
}
```

LoginControllerHeader.h

```
#pragma once

#include <stdbool.h>

void loginController() ;
```

SecretaryController.c

```
#include "SecretaryControllerHeader.h"

void getAllClasses() {
    DatabaseResult *result = selectAllCourses() ;
    if (result == NULL) return ;

    printAllCourses((Class **) result->rowsSet, result->numRows) ;

    freeDatabaseResult(result) ;
}

void addStudent() {
    getAllClasses() ;
    Student newStudent ;
    if (getStudentInfo(&newStudent)) {
        addStudentToDatabase(&newStudent) ;
    }
}

void getAllActivities() {
    DatabaseResult *result = getAllActivitiesFromDatabase() ;
    if (result == NULL) return ;
    printAllActivities((CuturalActivity **) result->rowsSet, result-
>numRows) ;

    freeDatabaseResult(result) ;
}

void addStudentJoinActivity() {
```

```
    getAllActivities() ;
    char studentName[STUDENT_NAME_MAX_LEN] ;
    int activityCode ;
    if (getStudentJoinActivityInfo(studentName, &activityCode)) {
        addStudentJoinActivityToDatabase(studentName, &activityCode) ;
    }
}


void bookPrivateLesson() {
    PrivateLesson privateLesson ;
    if (getPrivateLessonInfo(&privateLesson)) {
        bookPrivateLessonInDatabase(&privateLesson) ;
    }
}

void courseAbsenceReport() {
    getAllClasses() ;
    int courseCode ;

    if (!getCourseAbsenceReportInfo(&courseCode)) return ;

    DatabaseResult *result = getCourseAbsenceReportDB(courseCode) ;

    if (result == NULL) return ;

    printCourseAbsenceReport((Student **) result->rowsSet, result-
>numRows) ;

    freeDatabaseResult(result) ;
}

void addAbsence() {
    Absence absence ;
    if (getAbsenceInfo(&absence)) {
        addAbsenceToDatabase(&absence) ;
    }
}


void freeTeacherReport() {
    Date date ;
```

```
    Time time ;
    int duration ;

    if (!getFreeTeacherReportInfo(&date, &time, &duration)) return ;

    DatabaseResult *result = loadFreeTeachersFromDB(&date, &time,
&duration) ;
    if (result == NULL) return ;

    printFreeTeacherReport((char **) result->rowsSet, result->numRows) ;

    freeDatabaseResult(result) ;
}


void secretaryController() {

    do {
        int selectedOption = getSecretaryOption() ;

        showOptionHeader() ;
        switch (selectedOption) {

            case ADD_STUDENT :
                addStudent() ;
                break; ;

            case ADD_JOIN_ACTIVITY :
                addStudentJoinActivity() ;
                break ;

            case BOOK_LESSON :
                bookPrivateLesson() ;
                break ;

            case ADD_ABSENCE :
                addAbsence() ;
                break ;

            case COURSE_ABSENCE_REPORT :
                courseAbsenceReport() ;
                break ;
```

```
            case FREE_TEACHER_REPORT :
                freeTeacherReport() ;
                break ;

            case SECRETARY_QUIT :
                goto exit_loop ;

            default :
                printf("INVALID OPTION\n\n") ;
                break ;
        }
        showOptionHeader() ;
    } while (true) ;

    exit_loop:
    showOptionHeader() ;
}
```

SecretaryControllerHeader.h

```
#pragma once

#include "../model/Student.h"
#include "../view/SecretaryViewHeader.h"
#include "../db/SecretaryDBHeader.h"
#include "../view/CommonViewHeader.h"
#include <stdio.h>

enum SecretaryOption {
    ADD_STUDENT = 0,
    ADD_JOIN_ACTIVITY ,
    BOOK_LESSON,
    ADD_ABSENCE,
    COURSE_ABSENCE_REPORT,
    FREE_TEACHER_REPORT,
    SECRETARY_QUIT
} ;

void secretaryController() ;
```

TeacherController.c

```
#include "TeacherControllerHeader.h"


void generateAgenda(char *teacherUsername) {
    int weekIndex ;
    if (!getAgendaInfo(&weekIndex)) return ;

    // Esecuzione DB
                              DatabaseResult       **resultArray       =
generateAgendaFromDatabase(teacherUsername, weekIndex) ;

    if (resultArray == NULL) return ;

    for (int i = 0 ; i < 7 ; i++) {
        if (resultArray[i] == NULL) break ;
        DatabaseResult * result = resultArray[i] ;
                printAgenda((GeneralLesson  **)  result->rowsSet,  result-
>numRows) ;
        freeDatabaseResult(result) ;
    }

    free(resultArray) ;
}


void teacherController(char *teacherUsername) {

    do {
        int selectedOption = getTeacherOption() ;

        showOptionHeader() ;
        switch (selectedOption) {

            case GENERATE_AGENDA :
                generateAgenda(teacherUsername) ;
                break ;

            case TEACHER_QUIT :
                goto exit_loop ;

            default :
                printf("INVALID OPTION\n\n") ;
```

```
                break ;
        }
        showOptionHeader() ;
    } while (true) ;

    exit_loop:
    showOptionHeader() ;
}
```

TeacherControllerHeader.h

```
#pragma once

#include "../model/Teacher.h"
#include "../model/Lesson.h"

#include "../view/TeacherViewHeader.h"
#include "../view/ViewUtilsHeader.h"
#include "../view/TablePrinterHeader.h"
#include "../db/TeacherDBHeader.h"

enum TeacherOptions {
    GENERATE_AGENDA = 0,
    TEACHER_QUIT
} ;


void teacherController(char *teacherUsername) ;
```

**db**

AdministrationDatabaseHeader.h

```
#include <stdbool.h>
#include <string.h>

#include "../controller/AdministrationControllerHeader.h"

#include "../utils/SystemUtilsHeader.h"
#include "../utils/IOUtils.h"

#include "../model/Activity.h"
```

```
#include "../model/Class.h"
#include "../model/Level.h"
#include "../model/Student.h"
#include "../model/Teacher.h"
#include "../model/Lesson.h"
#include "../model/User.h"

#include "Connector.h"
#include "DatabaseUtilsHeader.h"


bool addLevelToDatabase(Level *levelPtr) ;

int *addClassToDatabase(Class *classPtr) ;

bool addTeacherToDatabase(Teacher *teacherPtr, char *username) ;

bool assignTeacherToClass(Teacher *teacherPtr, Class *classPtr) ;

int *organizeActivityInDatabase(CuturalActivity *newActivity) ;

bool addClassLessonToDatabase(ClassLesson *newLesson) ;

DatabaseResult *selectAllTeaching() ;

DatabaseResult *generateTeacherReportFromDB(char *teacherName, int *year,
int *monthIndex) ;

bool restartYearDB() ;

DatabaseResult *selectAllLevels() ;

bool createUserDB(User *newUser, Role userRole) ;

DatabaseResult *selectAllTeachers() ;
```

AdministrationController.c

```
#include "AdministrationDatabaseHeader.h"


bool addLevelToDatabase(Level *levelPtr) {
    MYSQL_STMT *addLevelProcedure ;
```

```
    if (!setupPreparedStatement(&addLevelProcedure, "CALL
aggiungi_livello(?,?,?) ", conn)) {
        printMysqlError(conn, "Impossibile Preparare Precedura 'Aggiungi
Livello'") ;
        return false ;
    }

    MYSQL_BIND param[3] ;

    bindParam(&param[0], MYSQL_TYPE_STRING, levelPtr->levelName,
strlen(levelPtr->levelName), false) ;
    bindParam(&param[1], MYSQL_TYPE_STRING, levelPtr->levelBookName,
strlen(levelPtr->levelBookName), false) ;
    bindParam(&param[2], MYSQL_TYPE_SHORT, &(levelPtr->levelHasExam),
sizeof(short int), false) ;

    if (mysql_stmt_bind_param(addLevelProcedure, param) != 0) {
        printStatementError(addLevelProcedure, "Impossibile Fare Il Bind
Dei Parametri per Procedura Aggiunta Livello") ;
        freeStatement(addLevelProcedure, false) ;
        return false ;
    }

    if (mysql_stmt_execute(addLevelProcedure) != 0) {
        printStatementError(addLevelProcedure, "Impossibile Eseguire
Procedura Aggiunta Livello") ;
        freeStatement(addLevelProcedure, false) ;
        return false ;
    }

    freeStatement(addLevelProcedure, false) ;

    return true ;
}


int *addClassToDatabase(Class *classPtr) {
    MYSQL_STMT *addClassProcedure ;
    if (!setupPreparedStatement(&addClassProcedure, "CALL
aggiungi_corso(?,?,?) ", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Aggiungi
Corso'") ;
```

```
        return NULL ;
    }

    MYSQL_BIND param[3] ;

    bindParam(&param[0], MYSQL_TYPE_STRING, classPtr->levelName,
strlen(classPtr->levelName), false) ;

    MYSQL_TIME mysqlDate ;
    prepareDateParam(&(classPtr->activationDate), &mysqlDate) ;

    bindParam(&param[1], MYSQL_TYPE_DATE, &mysqlDate, sizeof(MYSQL_TIME),
false) ;

    int newClassCode = 0 ;
    bindParam(&param[2], MYSQL_TYPE_LONG, &newClassCode, sizeof(int),
false) ;

    if (mysql_stmt_bind_param(addClassProcedure, param) != 0) {
        printStatementError(addClassProcedure, "Impossibile Fare
Parametri Procedura 'Aggiungi Corso'") ;
        freeStatement(addClassProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_execute(addClassProcedure) != 0) {
        printStatementError(addClassProcedure, "Impossibile Eseguire
Procedura 'Aggiungi Corso'") ;
        freeStatement(addClassProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_store_result(addClassProcedure) != 0) {
        printStatementError(addClassProcedure, "Store Risultato
Impossibile per 'Aggiungi Corso'") ;
        freeStatement(addClassProcedure, true) ;
        return NULL ;
    }

    MYSQL_BIND resultParam ;
    bindParam(&resultParam, MYSQL_TYPE_LONG, &newClassCode, sizeof(int),
false) ;
```

```
    if (mysql_stmt_bind_result(addClassProcedure, &resultParam) != 0) {
        printStatementError(addClassProcedure, "Errore Recupero Codice
Nuovo Corso") ;
        freeStatement(addClassProcedure, true) ;
        return NULL ;
    }

    if (mysql_stmt_fetch(addClassProcedure) != 0) {
        printStatementError(addClassProcedure, "Errore Fetch del
Risultato") ;
        freeStatement(addClassProcedure, true) ;
        return NULL ;
    }

    freeStatement(addClassProcedure, true) ;

    int *returnCode = myMalloc(sizeof(int)) ;
    *returnCode = newClassCode ;
    return returnCode ;
}


bool addTeacherToDatabase(Teacher *teacherPtr, char *username) {
    MYSQL_STMT *addTeacherProcedure ;
    if (!setupPreparedStatement(&addTeacherProcedure, "CALL
aggiungi_insegnante(?,?,?,?) ", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Aggiungi
Insegnante'") ;
        return false ;
    }

    MYSQL_BIND param[4] ;
    bindParam(&param[0], MYSQL_TYPE_STRING, teacherPtr->teacherName,
strlen(teacherPtr->teacherName), false) ;
    bindParam(&param[1], MYSQL_TYPE_STRING, teacherPtr-
>teacherNationality, strlen(teacherPtr->teacherNationality), false) ;
    bindParam(&param[2], MYSQL_TYPE_STRING, teacherPtr->teacherAddress,
strlen(teacherPtr->teacherAddress), false) ;
    bindParam(&param[3], MYSQL_TYPE_STRING, username, strlen(username),
false) ;
```

```
    if (mysql_stmt_bind_param(addTeacherProcedure, param) != 0) {
        printStatementError(addTeacherProcedure, "Impossibile Bind
Parametri di Procedura 'Aggiungi Insegnante'") ;
        freeStatement(addTeacherProcedure, false) ;
        return false ;
    }

    if (mysql_stmt_execute(addTeacherProcedure) != 0) {
        printStatementError(addTeacherProcedure, "Impossibile Eseguire
Procedura 'Aggiungi Insegnante'") ;
        freeStatement(addTeacherProcedure, false) ;
        return false ;
    }

    freeStatement(addTeacherProcedure, false) ;

    return true ;
}

bool assignTeacherToClass(Teacher *teacherPtr, Class *classPtr) {
    MYSQL_STMT *assignClassProcedure ;
    if (!setupPreparedStatement(&assignClassProcedure, "CALL
assegna_corso(?,?) ", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Assegna
Corso'") ;
        return false ;
    }

    MYSQL_BIND param[2] ;
    bindParam(&param[0], MYSQL_TYPE_LONG, &(classPtr->classCode),
sizeof(int), false) ;
    bindParam(&param[1], MYSQL_TYPE_STRING, teacherPtr->teacherName,
strlen(teacherPtr->teacherName), false) ;

    if (mysql_stmt_bind_param(assignClassProcedure, param) != 0) {
        printStatementError(assignClassProcedure, "Impossibile Bind
Parametri di Procedura 'Assegna Corso'") ;
        freeStatement(assignClassProcedure, false) ;
        return false ;
    }

    if (mysql_stmt_execute(assignClassProcedure) != 0) {
```

```
        printStatementError(assignClassProcedure, "Impossibile Eseguire
Procedura 'Assegna Corso'") ;
        freeStatement(assignClassProcedure, false) ;
        return false ;
    }

    freeStatement(assignClassProcedure, false) ;

    return true ;
}

int *organizeActivityInDatabase(CuturalActivity *newActivity) {
    MYSQL_STMT *organizeActivityProcedure ;
    if (!setupPreparedStatement(&organizeActivityProcedure, "CALL
organizza_attivita_culturale(?,?,?,?,?,?,?,?) ", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Organizza
Attività'") ;
        return NULL ;
    }

    MYSQL_BIND param[8] ;

    MYSQL_TIME mysqlDate ;
    prepareDateParam(&(newActivity->activityDate), &mysqlDate) ;
    MYSQL_TIME mysqlTime ;
    prepareTimeParam(&(newActivity->activityTime), &mysqlTime) ;

    bindParam(&param[0], MYSQL_TYPE_DATE, &mysqlDate, sizeof(MYSQL_TIME),
false) ;
    bindParam(&param[1], MYSQL_TYPE_TIME, &mysqlTime, sizeof(MYSQL_TIME),
false) ;

    short int activityType ;
    int activityCode = 0 ;
    if (newActivity->type == FILM) {
        activityType = 0 ;
        bindParam(&param[3], MYSQL_TYPE_STRING, newActivity->filmTitle,
strlen(newActivity->filmTitle), false) ;
        bindParam(&param[4], MYSQL_TYPE_STRING, newActivity-
>filmDirector, strlen(newActivity->filmDirector), false) ;

        bindParam(&param[5], MYSQL_TYPE_NULL, NULL, sizeof(NULL), true) ;
```

```
        bindParam(&param[6], MYSQL_TYPE_NULL, NULL, sizeof(NULL), true) ;
    }
    else {
        activityType = 1 ;
        bindParam(&param[3], MYSQL_TYPE_NULL, NULL, sizeof(NULL), true) ;
        bindParam(&param[4], MYSQL_TYPE_NULL, NULL, sizeof(NULL), true) ;

        bindParam(&param[5], MYSQL_TYPE_STRING, newActivity-
>meetingLecturer, strlen(newActivity->meetingLecturer), false) ;
        bindParam(&param[6], MYSQL_TYPE_STRING, newActivity-
>meetingArgument, strlen(newActivity->meetingArgument), false) ;
    }
    bindParam(&param[2], MYSQL_TYPE_SHORT, &activityType, sizeof(short
int), false) ;
    bindParam(&param[7], MYSQL_TYPE_LONG, &activityCode, sizeof(int),
false) ;

    if (mysql_stmt_bind_param(organizeActivityProcedure, param) != 0) {
        printStatementError(organizeActivityProcedure, "Impossibile Bind
Parametri di Procedura 'Organizza Attività'") ;
        freeStatement(organizeActivityProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_execute(organizeActivityProcedure) != 0) {
        printStatementError(organizeActivityProcedure, "Impossibile
Eseguire Procedura 'Organizza Attività'") ;
        freeStatement(organizeActivityProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_store_result(organizeActivityProcedure) != 0) {
        printStatementError(organizeActivityProcedure, "Store Risultato
Impossibile per 'Organizza Attività'") ;
        freeStatement(organizeActivityProcedure, true) ;
        return NULL ;
    }

    MYSQL_BIND resultParam ;
    bindParam(&resultParam, MYSQL_TYPE_LONG, &activityCode, sizeof(int),
false) ;
```

```
    if (mysql_stmt_bind_result(organizeActivityProcedure, &resultParam) !
= 0) {
        printStatementError(organizeActivityProcedure, "Errore Recupero
Codice Nuova Attività") ;
        freeStatement(organizeActivityProcedure, true) ;
        return NULL ;
    }

    if (mysql_stmt_fetch(organizeActivityProcedure) != 0) {
        printStatementError(organizeActivityProcedure, "Errore Fetch del
Risultato") ;
        freeStatement(organizeActivityProcedure, true) ;
        return NULL ;
    }

    freeStatement(organizeActivityProcedure, true) ;

    int *returnCode = (int *)myMalloc(sizeof(int)) ;
    *returnCode = activityCode ;

    return returnCode ;
}

void prepareDayOfWeekParam(enum DayOfWeek dayOfWeek, char
*dayOfWeekString) {
    switch (dayOfWeek)
    {
    case MONDAY :
        strcpy(dayOfWeekString, "Lun") ;
        break;
    case TUESDAY:
        strcpy(dayOfWeekString, "Mar") ;
        break;
    case WEDNESDAY :
        strcpy(dayOfWeekString, "Mer") ;
        break ;
    case THURSDAY :
        strcpy(dayOfWeekString, "Gio") ;
        break ;
    case FRIDAY :
        strcpy(dayOfWeekString, "Ven") ;
        break ;
```

```
    case SATURDAY :
        strcpy(dayOfWeekString, "Sab") ;
        break ;
    case SUNDAY :
        strcpy(dayOfWeekString, "Dom") ;
        break ;
    }
}

bool addClassLessonToDatabase(ClassLesson *newLesson) {
    MYSQL_STMT *addLessonToClassProcedure ;
    if (!setupPreparedStatement(&addLessonToClassProcedure, "CALL
aggiungi_lezione(?,?,?,?,?)", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Aggiungi
Lezione Corso'") ;
        return false ;
    }

    MYSQL_BIND param[5] ;
    MYSQL_TIME mysqlTime ;

    char dayOfWeekString[3 + 1] ;
    prepareDayOfWeekParam(newLesson->dayOfWeek, dayOfWeekString) ;
    bindParam(&param[0], MYSQL_TYPE_STRING, dayOfWeekString,
strlen(dayOfWeekString), false) ;
    prepareTimeParam(&(newLesson->startTime), &mysqlTime) ;
    bindParam(&param[1], MYSQL_TYPE_TIME, &mysqlTime, sizeof(MYSQL_TIME),
false) ;
    bindParam(&param[2], MYSQL_TYPE_LONG, &(newLesson->classCode),
sizeof(int), false) ;
    bindParam(&param[3], MYSQL_TYPE_LONG, &(newLesson->lessonDuration),
sizeof(int), false) ;
    bindParam(&param[4], MYSQL_TYPE_STRING, newLesson->teacherName,
strlen(newLesson->teacherName) , false) ;


    if (mysql_stmt_bind_param(addLessonToClassProcedure, param) != 0) {
        printStatementError(addLessonToClassProcedure, "Bind Parametri
Impossibile per 'Aggiungi Lezione Corso'") ;
        freeStatement(addLessonToClassProcedure, false) ;
        return false ;
    }
```

```
    if (mysql_stmt_execute(addLessonToClassProcedure) != 0) {
        printStatementError(addLessonToClassProcedure, "Esecuzione
Impossibile Per 'Aggiungi Lezione Corso'") ;
        freeStatement(addLessonToClassProcedure, false) ;
        return false ;
    }

    freeStatement(addLessonToClassProcedure, true) ;

    return true ;
}

DatabaseResult *selectAllTeaching() {
    MYSQL_STMT *loadAllTachingProcedure ;
    if (!setupPreparedStatement(&loadAllTachingProcedure, "CALL
recupera_docenze()", conn)) {
        printMysqlError(conn, "Impossibie Preparare Procedura 'Recupera
Docenze'") ;
        return NULL ;
    }

    if (mysql_stmt_execute(loadAllTachingProcedure) != 0) {
        printStatementError(loadAllTachingProcedure, "Errore Esecuzione
Recupero Docenze") ;
        freeStatement(loadAllTachingProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_store_result(loadAllTachingProcedure) != 0) {
        printStatementError(loadAllTachingProcedure, "Store Risultato
Impossibile per 'Recupera Docenze'") ;
        freeStatement(loadAllTachingProcedure, true) ;
        return NULL ;
    }

    int classCode ;
    char levelName[LEVEL_NAME_MAX_LEN] ;
    char teacherName[TEACHER_NAME_MAX_LEN] ;
    MYSQL_BIND returnParam[3] ;
    bindParam(&returnParam[0], MYSQL_TYPE_LONG, &classCode, sizeof(int),
```

```
false) ;
    bindParam(&returnParam[1], MYSQL_TYPE_STRING, levelName,
LEVEL_NAME_MAX_LEN, false) ;
    bindParam(&returnParam[2], MYSQL_TYPE_STRING, teacherName,
TEACHER_NAME_MAX_LEN, false) ;

    if (mysql_stmt_bind_result(loadAllTachingProcedure, returnParam) !=
0) {
        printStatementError(loadAllTachingProcedure, "Errore Bind Del
Risultato") ;
        freeStatement(loadAllTachingProcedure, false) ;
        return NULL ;
    }

    DatabaseResult *result = myMalloc(sizeof(DatabaseResult)) ;
    result->numRows = mysql_stmt_num_rows(loadAllTachingProcedure) ;
    result->rowsSet = myMalloc(sizeof(Teaching *) * result->numRows) ;

    int hasResult = mysql_stmt_fetch(loadAllTachingProcedure) ;
    int i = 0 ;
    while (hasResult != 1 && hasResult != MYSQL_NO_DATA) {
        Teaching *teaching = (Teaching *) myMalloc(sizeof(Teaching)) ;
        teaching->classCode = classCode ;
        strcpy(teaching->levelName, levelName) ;
        strcpy(teaching->teacherName, teacherName) ;

        result->rowsSet[i] = teaching ;
        hasResult = mysql_stmt_fetch(loadAllTachingProcedure) ;
        i++ ;
    }

    if (hasResult == 1) {
        printStatementError(loadAllTachingProcedure, "Fetch Risultato
Impossibile per 'Recupera Docenze'") ;
        freeStatement(loadAllTachingProcedure, true) ;
        freeDatabaseResult(result) ;
        return NULL ;
    }

    freeStatement(loadAllTachingProcedure, true) ;

    return result ;
```

```
}


DatabaseResult *generateTeacherReportFromDB(char *teacherName, int
*yearPtr, int *monthIndexPtr) {
    MYSQL_STMT *generateTeacherReportProcedure ;
    if (!setupPreparedStatement(&generateTeacherReportProcedure, "CALL
genera_report_insegnante(?,?,?)", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Genera
Report Insegnante'") ;
        return NULL ;
    }

    MYSQL_BIND param[3] ;
    bindParam(&param[0], MYSQL_TYPE_STRING, teacherName,
strlen(teacherName), false) ;
    bindParam(&param[1], MYSQL_TYPE_LONG, monthIndexPtr, sizeof(int),
false) ;
    bindParam(&param[2], MYSQL_TYPE_SHORT, yearPtr, sizeof(int), false) ;

    if (mysql_stmt_bind_param(generateTeacherReportProcedure, param) !=
0) {
        printStatementError(generateTeacherReportProcedure, "Bind
Parametri Impossibile per 'Genera Report Insegnante'") ;
        freeStatement(generateTeacherReportProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_execute(generateTeacherReportProcedure) != 0) {
        printStatementError(generateTeacherReportProcedure, "Esecuzione
Impossibiler Per 'Genera Report Insegnate'") ;
        freeStatement(generateTeacherReportProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_store_result(generateTeacherReportProcedure) != 0) {
        printStatementError(generateTeacherReportProcedure, "Store
Risultato Impossibile per 'Genera Report Insegnante'") ;
        freeStatement(generateTeacherReportProcedure, true) ;
        return NULL ;
    }
```

```
    MYSQL_BIND returnParam[4] ;
    MYSQL_TIME mysqlDate ;
    MYSQL_TIME mysqlTime ;
    int lessonDuration ;
    char lessonType[5] ;

    bindParam(&returnParam[0], MYSQL_TYPE_DATE, &mysqlDate,
sizeof(MYSQL_TIME), false) ;
    bindParam(&returnParam[1], MYSQL_TYPE_TIME, &mysqlTime,
sizeof(MYSQL_TIME), false) ;
    bindParam(&returnParam[2], MYSQL_TYPE_LONG, &lessonDuration,
sizeof(int), false) ;
    bindParam(&returnParam[3], MYSQL_TYPE_STRING, lessonType, 5, false) ;

    if (mysql_stmt_bind_result(generateTeacherReportProcedure,
returnParam) != 0) {
        printStatementError(generateTeacherReportProcedure, "Bind
Risultato Impossibile Per 'Genera Report Insegnante'") ;
        freeStatement(generateTeacherReportProcedure, true) ;
        return NULL ;
    }

    DatabaseResult *result = (DatabaseResult *)
myMalloc(sizeof(DatabaseResult)) ;
    result->numRows = mysql_stmt_num_rows(generateTeacherReportProcedure)
;
    result->rowsSet = myMalloc(sizeof(ReportLesson *) * result-
>numRows) ;

    int hasResult = mysql_stmt_fetch(generateTeacherReportProcedure) ;
    int i = 0 ;
    while (hasResult != 1 && hasResult != MYSQL_NO_DATA) {
        ReportLesson *lesson = myMalloc(sizeof(ReportLesson)) ;
        getDateParam(&(lesson->lessonDate), &mysqlDate) ;
        getTimeParam(&(lesson->startTime), &mysqlTime) ;
        lesson->duration = lessonDuration ;
        lesson->lessonType = (strcmp(lessonType, "C") == 0) ? COURSE :
PRIVATE ;

        result->rowsSet[i] = lesson ;
        i++ ;
        hasResult = mysql_stmt_fetch(generateTeacherReportProcedure) ;
```

```
    }

    if (hasResult == 1) {
        printStatementError(generateTeacherReportProcedure, "Fetch
Impossibile per 'Report Insegnante'") ;
        freeStatement(generateTeacherReportProcedure, true) ;
        freeDatabaseResult(result) ;
        return NULL ;
    }

    freeStatement(generateTeacherReportProcedure, true) ;

    return result ;
}


bool restartYearDB() {
    MYSQL_STMT *restartYearProcedure ;
    if (!setupPreparedStatement(&restartYearProcedure, "CALL
riavvia_anno() ", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Riavvio
Anno'") ;
        return false ;
    }

    if (mysql_stmt_execute(restartYearProcedure) != 0) {
        printStatementError(restartYearProcedure, "Impossile Eseguire
Procedura 'Riavvio Anno'") ;
        return false ;
    }

    freeStatement(restartYearProcedure, false) ;

    return true ;
}


DatabaseResult *selectAllLevels() {
    MYSQL_STMT *preparedStatement ;
    if (!setupPreparedStatement(&preparedStatement, "CALL
recupera_livelli()", conn)) {
        printMysqlError(conn, "Errore Inizializzazione Procedura
```

```
'Recupera Livelli'") ;
        return NULL ;
    }

    if (mysql_stmt_execute(preparedStatement) != 0) {
        printStatementError(preparedStatement, "Impossibile Eseguire
Procedura 'Recupera Livelli'") ;
        freeStatement(preparedStatement, false) ;
        return NULL ;
    }


    if (mysql_stmt_store_result(preparedStatement) != 0) {
        printStatementError(preparedStatement, "Store Risultato
Impossibile per 'Recupera Livelli'") ;
        freeStatement(preparedStatement, true) ;
        return NULL ;
    }

    MYSQL_BIND returnParam[3] ;
    char levelName[LEVEL_NAME_MAX_LEN] ;
    char bookName[LEVEL_BOOK_NAME_MAX_LEN] ;
    int hasExam ;

    bindParam(&returnParam[0], MYSQL_TYPE_STRING, levelName,
LEVEL_NAME_MAX_LEN, false) ;
    bindParam(&returnParam[1], MYSQL_TYPE_STRING, bookName,
LEVEL_BOOK_NAME_MAX_LEN, false) ;
    bindParam(&returnParam[2], MYSQL_TYPE_LONG, &hasExam, sizeof(int),
false) ;

    if (mysql_stmt_bind_result(preparedStatement, returnParam) != 0) {
        printStatementError(preparedStatement, "Impossibile Recuperare
Risultato 'Recupera Livelli'") ;
        freeStatement(preparedStatement, false) ;
        return NULL ;
    }

    DatabaseResult *result = myMalloc(sizeof(DatabaseResult)) ;
    result->numRows = mysql_stmt_num_rows(preparedStatement) ;
    result->rowsSet = myMalloc(sizeof(Level *) * result->numRows) ;
```

```
    int hasResult = mysql_stmt_fetch(preparedStatement) ;
    int i = 0 ;
    while (hasResult != 1 && hasResult != MYSQL_NO_DATA) {
        Level *level = (Level *) myMalloc(sizeof(Level)) ;
        strcpy(level->levelName, levelName) ;
        strcpy(level->levelBookName, bookName) ;
        level->levelHasExam = hasExam ;

        result->rowsSet[i] = level ;

        hasResult = mysql_stmt_fetch(preparedStatement) ;
        i++ ;
    }

    if (hasResult == 1) {
        printStatementError(preparedStatement, "Fetch Risultato
Impossibile per 'Recupera Livelli'") ;
        freeStatement(preparedStatement, true) ;
        freeDatabaseResult(result) ;
        return NULL ;
    }

    freeStatement(preparedStatement, true) ;

    return result ;
}


DatabaseResult *selectAllTeachers() {
    MYSQL_STMT *preparedStatement ;
    if (!setupPreparedStatement(&preparedStatement, "CALL
recupera_insegnanti()", conn)) {
        printMysqlError(conn, "Errore Inizializzazione Procedura
'Recupera Insegnanti'") ;
        return NULL ;
    }

    if (mysql_stmt_execute(preparedStatement) != 0) {
        printStatementError(preparedStatement, "Impossibile Eseguire
Procedura 'Recupera Insegnanti'") ;
        freeStatement(preparedStatement, false) ;
```

```
        return NULL ;
    }

    if (mysql_stmt_store_result(preparedStatement) != 0) {
        printStatementError(preparedStatement, "Store Result Impossibile
per 'Recupera Insegnanti'") ;
        freeStatement(preparedStatement, true) ;
        return NULL ;
    }

    char teacherName[TEACHER_NAME_MAX_LEN] ;
    char teacherAddress[TEACHER_ADDRESS_MAX_LEN] ;
    char teacherNation[TEACHER_NATIONALITY_MAX_LEN] ;

    MYSQL_BIND returnParam[3] ;
    bindParam(&returnParam[0], MYSQL_TYPE_STRING, teacherName,
TEACHER_NAME_MAX_LEN, false) ;
    bindParam(&returnParam[1], MYSQL_TYPE_STRING, teacherNation,
TEACHER_NATIONALITY_MAX_LEN, false) ;
    bindParam(&returnParam[2], MYSQL_TYPE_STRING, teacherAddress,
TEACHER_ADDRESS_MAX_LEN, false) ;

    if (mysql_stmt_bind_result(preparedStatement, returnParam) != 0) {
        printStatementError(preparedStatement, "Impossibile Recuperare
Risultato 'Recupera Insegnanti'") ;
        freeStatement(preparedStatement, true) ;
        return NULL ;
    }

    DatabaseResult *result = myMalloc(sizeof(DatabaseResult)) ;
    result->numRows = mysql_stmt_num_rows(preparedStatement) ;
    result->rowsSet = myMalloc(sizeof(Teacher *) * result->numRows) ;

    int hasResult = mysql_stmt_fetch(preparedStatement) ;
    int i = 0 ;
    while (hasResult != 1 && hasResult != MYSQL_NO_DATA) {
        Teacher *teacher = myMalloc(sizeof(Teacher)) ;
        strcpy(teacher->teacherName, teacherName) ;
        strcpy(teacher->teacherAddress, teacherAddress) ;
        strcpy(teacher->teacherNationality, teacherNation) ;

        result->rowsSet[i] = teacher ;
```

```
        i++ ;

        hasResult = mysql_stmt_fetch(preparedStatement) ;
    }

    if (hasResult == 1) {
        printStatementError(preparedStatement, "Fetch Impossibile per
'Recupera Insegnanti'") ;
        freeStatement(preparedStatement, true) ;
        freeDatabaseResult(result) ;
        return NULL ;
    }

    freeStatement(preparedStatement, true) ;

    return result ;
}
```

CommonDB.c

```
#include "CommonDBHeader.h"

DatabaseResult *selectAllCourses() {
    MYSQL_STMT *loadClassesProcedure ;
    if (!setupPreparedStatement(&loadClassesProcedure, "CALL
recupera_corsi()", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Recupera
Corsi'") ;
        return NULL ;
    }

    if (mysql_stmt_execute(loadClassesProcedure) != 0) {
        printStatementError(loadClassesProcedure, "Impossibile Eseguire
'Recupera Corsi'") ;
        freeStatement(loadClassesProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_store_result(loadClassesProcedure) != 0) {
        printStatementError(loadClassesProcedure, "Store Risultato
Impossibile per 'Recupera Corsi'") ;
        freeStatement(loadClassesProcedure, true) ;
        return NULL ;
```

```
    }

    int classCode ;
    char classLevelName[LEVEL_NAME_MAX_LEN] ;
    int classStudentsNumber ;
    MYSQL_TIME mysqlTime ;
    memset(&mysqlTime, 0, sizeof(MYSQL_TIME)) ;

    MYSQL_BIND resultParam[4] ;
    bindParam(&resultParam[0], MYSQL_TYPE_LONG, &classCode, sizeof(int),
false) ;
    bindParam(&resultParam[1], MYSQL_TYPE_VAR_STRING, classLevelName,
LEVEL_NAME_MAX_LEN, false) ;
    bindParam(&resultParam[3], MYSQL_TYPE_LONG, &classStudentsNumber,
sizeof(int), false) ;
    bindParam(&resultParam[2], MYSQL_TYPE_DATE, &mysqlTime,
sizeof(MYSQL_TIME), false) ;

    if (mysql_stmt_bind_result(loadClassesProcedure, resultParam) != 0)
{
        printStatementError(loadClassesProcedure, "Impossibile Recuperare
Parametri") ;
        freeStatement(loadClassesProcedure, true) ;
        return NULL ;
    }

    DatabaseResult *result = myMalloc(sizeof(DatabaseResult)) ;
    result->numRows = mysql_stmt_num_rows(loadClassesProcedure) ;
    result->rowsSet = myMalloc(sizeof(Class *) * result->numRows) ;

    int i = 0 ;
    int hasResult = mysql_stmt_fetch(loadClassesProcedure) ;
    while (hasResult != 1 && hasResult != MYSQL_NO_DATA) {
        Class *class = (Class *) myMalloc(sizeof(Class)) ;
        class->classCode = classCode ;
        strcpy(class->levelName, classLevelName) ;
        class->studentsNumber = classStudentsNumber ;

        class->activationDate.year = mysqlTime.year ;
        class->activationDate.month = mysqlTime.month ;
        class->activationDate.day = mysqlTime.day ;
```

```
        result->rowsSet[i] = class ;
        hasResult = mysql_stmt_fetch(loadClassesProcedure) ;
        i++ ;
    }

    if (hasResult == 1) {
        printStatementError(loadClassesProcedure, "Fetch Impossibile Per
'Recupera Corsi'") ;
        freeDatabaseResult(result) ;
        freeStatement(loadClassesProcedure, true) ;
        return NULL ;
    }

    freeStatement(loadClassesProcedure, true) ;

    return result ;
}
```

CommonDBHeader.h

```
#pragma once

#include "DatabaseUtilsHeader.h"
#include "../model/Class.h"
#include "Connector.h"
#include "../utils/SystemUtilsHeader.h"

DatabaseResult *selectAllCourses() ;
```

Connector.c

```
#include <stdbool.h>
#include <stdio.h>
#include <mysql/mysql.h>
#include "../utils/IOUtils.h"
#include "Connector.h"
#include "../controller/LoginControllerHeader.h"
#include "DatabaseUtilsHeader.h"



MYSQL *conn ;
```

```
bool connectToDatabase() {
    char *host = getenv(DB_HOST) ;
    char *port = getenv(DB_PORT) ;
    char *databaseName = getenv(DB_NAME) ;
    char *loginUser = getenv(DB_LOGIN_USER) ;
    char *loginPassword = getenv(DB_LOGIN_PASSWD) ;

    if (host == NULL || port == NULL || loginUser == NULL ||
loginPassword == NULL || databaseName == NULL) {
        fprintf(stderr, "Variabili d'Ambiente non Trovate\n") ;
        return false ;
    }

    conn = mysql_init(NULL) ;
    if (conn == NULL) {
        fprintf(stderr, "Errore in Inizializzazione Librerie\n") ;
        return false ;
    }

    if (mysql_real_connect(
        conn,
        host,
        loginUser,
        loginPassword,
        databaseName,
        atoi(port),
        NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS |
CLIENT_COMPRESS | CLIENT_INTERACTIVE | CLIENT_REMEMBER_OPTIONS) == NULL)
{
            fprintf(stderr, "Errore di Connessione al Database\n%s\n\n",
mysql_error(conn)) ;
            return false ;
        }

    return true ;
}
```

Connector.h

```
#pragma once

#include <stdbool.h>
```

```
#include <mysql/mysql.h>
#include "../controller/LoginControllerHeader.h"
#include "../model/User.h"

#define DB_HOST "DB.HOST"
#define DB_PORT "DB.PORT"
#define DB_NAME "DB.NAME"

#define DB_LOGIN_USER "LOGIN.USER"
#define DB_LOGIN_PASSWD "LOGIN.PASSWD"

#define DB_AMMINISTRAZIONE_USER "AMMINISTRAZIONE.USER"
#define DB_AMMINISTRAZIONE_PASSWD "AMMINISTRAZIONE.PASSWD"

#define DB_SEGRETERIA_USER "SEGRETERIA.USER"
#define DB_SEGRETERIA_PASSWD "SEGRETERIA.PASSWD"

#define DB_INSEGNANTE_USER "INSEGNANTE.USER"
#define DB_INSEGNANTE_PASSWD "INSEGNANTE.PASSWD"

extern MYSQL *conn ;


bool connectToDatabase() ;

bool initializePreparedStatement(Role role) ;

bool closeAllStatement() ;
```

DatabaseLoginHeader.h

```
#pragma once

#include <stdbool.h>
#include "../controller/LoginControllerHeader.h"
#include "DatabaseUtilsHeader.h"
#include "../model/User.h"

Role attemptLogin(User *loginDredentialsPtr) ;

bool switchRole(Role newRole) ;
```

DatabaseUtils.c

```
#include "DatabaseUtilsHeader.h"

bool is_null = true ;

/*
    Funzione per preparare un paramentro MYSQL_BIND.
    - mysqlParam : Puntatore al MYSQL_BIND da preparare
    - mysqlType : Tipo di dato MYSQL da legare
    - paramPtr : Puntatore al buffer da cui prendere/in cui inserire
l'informazione
    - paramSize : Lunghezza del buffer puntato da paramPtr (strlen /
sizeof)
    - nullable : Indica se il paramPtr è NULL
*/
void bindParam(MYSQL_BIND *mysqlParam, enum enum_field_types mysqlType,
void *paramPtr, unsigned long paramSize, bool nullable) {

    memset(mysqlParam, 0, sizeof(MYSQL_BIND)) ;

    mysqlParam->buffer = paramPtr ;
    mysqlParam->buffer_type = mysqlType ;
    mysqlParam->buffer_length = paramSize ;

    if (nullable) mysqlParam->is_null = &is_null ;
}

//Fa la Conversione da Tipo Date a MYSQL_TIME
void prepareDateParam(Date *datePtr , MYSQL_TIME *mysqlTime) {

    memset(mysqlTime, 0, sizeof(MYSQL_TIME)) ;

    mysqlTime->day = datePtr->day ;
    mysqlTime->month = datePtr->month ;
    mysqlTime->year = datePtr->year ;
}



//Conversione da Time a MYSQL_TIME
void prepareTimeParam(Time *timePtr, MYSQL_TIME *mysqlTime) {
```

```
    memset(mysqlTime, 0, sizeof(MYSQL_TIME)) ;

    mysqlTime->hour = timePtr->hour ;
    mysqlTime->minute = timePtr->minute ;
    //mysqlTime->second = timePtr->second ;
}

void getDateParam(Date *datePtr, MYSQL_TIME *mysqlTime) {
    memset(datePtr, 0, sizeof(Date)) ;
    datePtr->year = mysqlTime->year ;
    datePtr->month = mysqlTime->month ;
    datePtr->day = mysqlTime->day ;
}

void getTimeParam(Time *timePtr, MYSQL_TIME *mysqlTime) {
    memset(timePtr, 0, sizeof(Time)) ;
    timePtr->hour = mysqlTime->hour ;
    timePtr->minute = mysqlTime->minute ;
    //timePtr->second = mysqlTime->second ;
}

void printMysqlError(MYSQL *conn, char *errorMessage) {
    char sqlErrorMessage[500] ;
    sprintf(sqlErrorMessage, "%s\nErrore %d : %s", errorMessage,
mysql_errno(conn), mysql_error(conn)) ;
    printError(sqlErrorMessage) ;
}

void printStatementError(MYSQL_STMT *statement, char *errorMessage) {
    char statementErrorMessage[500] ;
    sprintf(statementErrorMessage, "%s\nErrore %d : %s", errorMessage,
mysql_stmt_errno(statement), mysql_stmt_error(statement)) ;
    printError(statementErrorMessage) ;
}


/*
 Funzione per liberare uno statement dopo l'esecuzione
    - statement : statement da liberare
    - freeSet : Indica se deve essere consumato il result set ritornato
dallo statement
 */
```

```
void freeStatement(MYSQL_STMT *statement, bool freeSet) {

    if (freeSet) while (mysql_stmt_next_result(statement) == 0) ;
    mysql_stmt_free_result(statement) ;
    mysql_stmt_reset(statement) ;

    mysql_stmt_close(statement) ;
}

//Inizializza uno statement MYSQL
bool setupPreparedStatement(MYSQL_STMT **statement, char
*statementCommand, MYSQL *conn) {

    *statement = mysql_stmt_init(conn) ;

    if (*statement == NULL) {
        printMysqlError(conn, "Impossibile Inizializzare Statement") ;
        return false ;
    }

    if (mysql_stmt_prepare(*statement, statementCommand,
strlen(statementCommand)) != 0) {
        printStatementError(*statement, "Impossibile Associare Comando e
Statement") ;
        return false ;
    }

    bool update = true ;
    mysql_stmt_attr_set(*statement, STMT_ATTR_UPDATE_MAX_LENGTH, &update)
;

    return true ;
}

void freeDatabaseResult(DatabaseResult *databaseResultPtr) {
    if (databaseResultPtr == NULL) {
        free(databaseResultPtr) ;
        return ;
    }
    int numRows = databaseResultPtr->numRows ;
    for (int i = 0 ; i < numRows ; i++) {
        free(databaseResultPtr->rowsSet[i]) ;
```

```
    }
    free(databaseResultPtr->rowsSet) ;
    free(databaseResultPtr) ;
}
```

DatabaseUtilsHeader.h

```
#pragma once

#include <mysql/mysql.h>
#include <stdbool.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "../utils/TimeUtils.h"
#include "../utils/IOUtils.h"
#include "../utils/TimeUtils.h"

//Introduco per evitare di andare a riprendere il numero di righe ogni
volta che devo stampare una tabella
typedef struct {
    int numRows ;
    void **rowsSet ;
} DatabaseResult ;

void freeDatabaseResult(DatabaseResult *databaseResultPtr) ;


void bindParam(MYSQL_BIND *mysqlParam, enum enum_field_types mysqlType,
void *paramPtr, unsigned long paramSize, bool nullable) ;

void prepareDateParam(Date *datePtr , MYSQL_TIME *mysqlTime) ;

void prepareTimeParam(Time *timePtr, MYSQL_TIME *mysqlTime) ;

void printMysqlError(MYSQL *conn, char *errorMessage) ;

void printStatementError(MYSQL_STMT *statement, char *errorMessage) ;

void freeStatement(MYSQL_STMT *statement, bool freeResultSet) ;

bool setupPreparedStatement(MYSQL_STMT **statement, char
*statementCommand, MYSQL *conn) ;
```

```
void getDateParam(Date *datePtr, MYSQL_TIME *mysqlTime) ;

void getTimeParam(Time *timePtr, MYSQL_TIME *mysqlTime) ;
```

LoginDB.c

```
#include "DatabaseLoginHeader.h"
#include <mysql/mysql.h>
#include "../controller/LoginControllerHeader.h"
#include "Connector.h"
#include "DatabaseUtilsHeader.h"
#include <string.h>
#include <stdio.h>
#include "../utils/IOUtils.h"
#include "../model/User.h"

Role attemptLogin(User *loginCredentials) {
    MYSQL_STMT *loginProcedure ;
    if (!setupPreparedStatement(&loginProcedure, "CALL login(?,?,?) ",
conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura
'Login'") ;
        return false ;
    }
    /*
        La procedura di Login ritorna un'unica tabella nel result set
contenente
        il Terzo parametro che è un parametro di OUT: questo parametro è
un codice numerico associato
        all'utente e che indica il ruolo dell'utente.
        Il codice numerico tornato dal DB è mappato nell'enum Role
    */

    MYSQL_BIND param[3] ;
    int role = LOGIN ; // Default set to LOGIN

    bindParam(&param[0], MYSQL_TYPE_VAR_STRING, loginCredentials-
>username, strlen(loginCredentials->username), false) ; //Set username
Param
    bindParam(&param[1], MYSQL_TYPE_VAR_STRING, loginCredentials-
>password, strlen(loginCredentials->password), false) ; // Bind password
Param
```

```
    bindParam(&param[2], MYSQL_TYPE_LONG, &role, sizeof(role), false) ;

    //Binding parametri della procedura
    if (mysql_stmt_bind_param(loginProcedure, param) != 0) {
        printStatementError(loginProcedure, "Impossibile Fare Binding Dei
Parametri di Login") ;
        freeStatement(loginProcedure, false) ;
        return role ;
    }

    //Esecuzione della procedura
    if (mysql_stmt_execute(loginProcedure) != 0) {
        printStatementError(loginProcedure, "Impossibile Eseguire
Procedeura di Login") ;
        freeStatement(loginProcedure, false) ;
        return role ;
    }

    //Set Parametro di OUT
    bindParam(&param[2], MYSQL_TYPE_LONG, &role, sizeof(role), false) ;

    //Bind del parametro di OUT
    if (mysql_stmt_bind_result(loginProcedure, &param[2]) != 0) {
        printStatementError(loginProcedure, "Impossibile Fare il Bind del
risultato di Login") ;
        freeStatement(loginProcedure, true) ;
        return role ;
    }

    //Presa del parametro di OUT

    if (mysql_stmt_fetch(loginProcedure) != 0) {
        printStatementError(loginProcedure, "Impossibile Recuperare
Ruolo") ;
        return role ;
    }

    //Consumo resto del resul set attuale
    while (mysql_stmt_fetch(loginProcedure) == 0) ;

    freeStatement(loginProcedure, true) ;
```

```
    return role ;

}


bool switchRole(Role role) {
    char *databaseName = getenv("DB.NAME") ;
    char *username ;
    char *password ;

    switch(role) {
        case AMMINISTRAZIONE :
            username = getenv(DB_AMMINISTRAZIONE_USER) ;
            password = getenv(DB_AMMINISTRAZIONE_PASSWD) ;
            break ;
        case SEGRETERIA :
            username = getenv(DB_SEGRETERIA_USER) ;
            password = getenv(DB_SEGRETERIA_PASSWD) ;
            break ;
        case INSEGNANTE :
            username = getenv(DB_INSEGNANTE_USER) ;
            password = getenv(DB_INSEGNANTE_PASSWD) ;
            break ;
        case LOGIN :
            username = getenv(DB_LOGIN_USER) ;
            password = getenv(DB_LOGIN_PASSWD) ;
            break ;
    }

    if (username == NULL || password == NULL || databaseName == NULL) {
        printError("Errore : Variabili d'Ambiente Non Trovate") ;
        return false ;
    }

    if (mysql_change_user(conn, username, password, databaseName) != 0) {
        printMysqlError(conn, "Impossibile Cambiare Privilegi Utente") ;
        return false ;
    }

    return true ;
}
```

SecretaryDB.c

```
#include "SecretaryDBHeader.h"

bool addStudentToDatabase(Student *studentPtr) {
    MYSQL_STMT *addStudentProcedure ;
    if (!setupPreparedStatement(&addStudentProcedure, "CALL
aggiungi_allievo(?,?,?,?)", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Aggiungi
Allievo'") ;
        return false ;
    }

    MYSQL_BIND param[4] ;

    bindParam(&param[0], MYSQL_TYPE_STRING, studentPtr->studentName,
strlen(studentPtr->studentName), false) ;
    bindParam(&param[1], MYSQL_TYPE_STRING, studentPtr->studentTelephone,
strlen(studentPtr->studentTelephone), false) ;

    MYSQL_TIME mysqlSubscribeDate ;
    prepareDateParam(&(studentPtr->studentSubscribeDate),
&mysqlSubscribeDate) ;
    bindParam(&param[2], MYSQL_TYPE_DATE, &mysqlSubscribeDate,
sizeof(MYSQL_TIME), false) ;

    bindParam(&param[3], MYSQL_TYPE_LONG, &(studentPtr-
>studentClass.classCode), sizeof(int), false) ;

    if (mysql_stmt_bind_param(addStudentProcedure, param) != 0) {
        printStatementError(addStudentProcedure, "Bind Parametri
Impossibile Per 'Aggiungi Allievo'") ;
        freeStatement(addStudentProcedure, false) ;
        return false ;
    }

    if (mysql_stmt_execute(addStudentProcedure) != 0) {
        printStatementError(addStudentProcedure, "Esecuzione Impossibile
Per 'Aggiungi Allievo'") ;
        freeStatement(addStudentProcedure, false) ;
        return false ;
    }
```

```
    freeStatement(addStudentProcedure, false) ;

    return true ;
}

DatabaseResult *getAllActivitiesFromDatabase() {

    MYSQL_STMT *loadAllActivitiesProcedure ;
    if (!setupPreparedStatement(&loadAllActivitiesProcedure, "CALL
recupera_attivita()", conn)) {
        printStatementError(loadAllActivitiesProcedure, "Impossibile
Preparare Procedura 'Recupera Attività'") ;
        return NULL ;
    }

    if (mysql_stmt_execute(loadAllActivitiesProcedure) != 0) {
        printStatementError(loadAllActivitiesProcedure, "Impossibile
Eseguire Procedura Recupera Attività") ;
        freeStatement(loadAllActivitiesProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_store_result(loadAllActivitiesProcedure) != 0) {
        printStatementError(loadAllActivitiesProcedure, "Store Risultato
Impossibile per 'Recupera Attività'") ;
        freeStatement(loadAllActivitiesProcedure, false) ;
        return NULL ;
    }

    int activityCode ;
    char activityFilmTitle[FILM_TITLE_MAX_LEN] ;
    char activityFilmDirector[FILM_DIRECTOR_NAME_MAX_LEN] ;
    char activityMeetingLecturer[MEETING_LECTURER_NAME_MAX_LEN] ;
    char activityMeetingArgument[MEETING_ARGUMENT_MAX_LEN] ;

    int activityType ;

    MYSQL_TIME mysqlDate ;
    MYSQL_TIME mysqlTime ;
    MYSQL_BIND resultParam[8] ;
    bindParam(&resultParam[0], MYSQL_TYPE_LONG, &activityCode,
sizeof(int), false) ;
```

```
    bindParam(&resultParam[1], MYSQL_TYPE_DATE, &mysqlDate,
sizeof(MYSQL_TIME), false) ;
    bindParam(&resultParam[2], MYSQL_TYPE_TIME, &mysqlTime,
sizeof(MYSQL_TIME), false) ;
    bindParam(&resultParam[3], MYSQL_TYPE_LONG, &activityType,
sizeof(int), false) ;
    bindParam(&resultParam[4], MYSQL_TYPE_STRING, activityFilmTitle,
FILM_TITLE_MAX_LEN, true) ;
    bindParam(&resultParam[5], MYSQL_TYPE_STRING, activityFilmDirector,
FILM_DIRECTOR_NAME_MAX_LEN, true) ;
    bindParam(&resultParam[6], MYSQL_TYPE_STRING,
activityMeetingLecturer, MEETING_LECTURER_NAME_MAX_LEN, true) ;
    bindParam(&resultParam[7], MYSQL_TYPE_STRING,
activityMeetingArgument, MEETING_ARGUMENT_MAX_LEN, true) ;

    if (mysql_stmt_bind_result(loadAllActivitiesProcedure, resultParam) !
= 0) {
        printStatementError(loadAllActivitiesProcedure, "Impossibile Bind
Risultato di Recupera Attività") ;
        freeStatement(loadAllActivitiesProcedure, true) ;
        return NULL ;
    }

    DatabaseResult *result = myMalloc(sizeof(DatabaseResult)) ;
    result->numRows =
mysql_stmt_num_rows(loadAllActivitiesProcedure) ; //Senza Store non si
può usare
    result->rowsSet = myMalloc(sizeof(CuturalActivity *) * result-
>numRows) ;

    int i = 0 ;
    int hasResult = mysql_stmt_fetch(loadAllActivitiesProcedure) ;
    while (hasResult != 1 && hasResult != MYSQL_NO_DATA) {
        CuturalActivity *activity = (CuturalActivity *)
myMalloc(sizeof(CuturalActivity)) ;
        activity->activityCode = activityCode ;
        getDateParam(&(activity->activityDate), &mysqlDate) ;
        getTimeParam(&(activity->activityTime), &mysqlTime) ;

        if (activityType == 0) {
            activity->type = FILM ;
            strcpy(activity->filmTitle, activityFilmTitle) ;
```

```
            strcpy(activity->filmDirector, activityFilmDirector) ;
            strcpy(activity->meetingLecturer, "") ;
            strcpy(activity->meetingArgument, "") ;
        }
        else {
            activity->type = MEETING ;
            strcpy(activity->filmTitle, "") ;
            strcpy(activity->filmDirector, "") ;
            strcpy(activity->meetingLecturer, activityMeetingLecturer) ;
            strcpy(activity->meetingArgument, activityMeetingArgument) ;
        }

        result->rowsSet[i] = activity ;
        hasResult = mysql_stmt_fetch(loadAllActivitiesProcedure) ;
        i++ ;
    }

    if (hasResult == 1) {
        printStatementError(loadAllActivitiesProcedure, "Fetch
Impossibile Per 'Recupera Attività'") ;
        freeDatabaseResult(result) ;
        freeStatement(loadAllActivitiesProcedure, true) ;
        return NULL ;
    }

    freeStatement(loadAllActivitiesProcedure, true) ;

    return result ;
}


bool addStudentJoinActivityToDatabase(char *studentName, int
*activityCode) {
    MYSQL_STMT *addJoinProcedure ;
    if (!setupPreparedStatement(&addJoinProcedure, "CALL
aggiungi_partecipazione(?,?)", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Aggiungi
Partecipazione'") ;
        return false ;
    }

    MYSQL_BIND param[2] ;
```

```
    bindParam(&param[0], MYSQL_TYPE_LONG, activityCode, sizeof(int),
false) ;
    bindParam(&param[1], MYSQL_TYPE_STRING, studentName,
strlen(studentName), false) ;

    if (mysql_stmt_bind_param(addJoinProcedure, param) != 0) {
        printStatementError(addJoinProcedure, "Bind Parametri Impossibile
per 'Aggiungi Partecipazione'") ;
        freeStatement(addJoinProcedure, false) ;
        return false ;
    }

    if (mysql_stmt_execute(addJoinProcedure) != 0) {
        printStatementError(addJoinProcedure, "Esecuzione Impossibile per
'Aggiungi Partecipazione'") ;
        freeStatement(addJoinProcedure, false) ;
        return false ;
    }

    freeStatement(addJoinProcedure, false) ;

    return true ;
}

bool addAbsenceToDatabase(Absence *newAbsence) {
    MYSQL_STMT *addAbsenceProcedure ;
    if (!setupPreparedStatement(&addAbsenceProcedure, "CALL
aggiungi_assenza(?,?,?)", conn)) {
        printStatementError(addAbsenceProcedure, "Impossibile Preparare
Procedura 'Aggiungi Assenza'") ;
        return false ;
    }

    MYSQL_BIND param[3] ;

    bindParam(&param[0], MYSQL_TYPE_STRING, newAbsence->studentName,
strlen(newAbsence->studentName), false) ;

    MYSQL_TIME mysqlDate ;
    prepareDateParam(&(newAbsence->absenceDate), &mysqlDate) ;
    bindParam(&param[1], MYSQL_TYPE_DATE, &mysqlDate, sizeof(MYSQL_TIME),
false) ;
```

```
    MYSQL_TIME mysqlTime ;
    prepareTimeParam(&(newAbsence->startTime), &mysqlTime) ;
    bindParam(&param[2], MYSQL_TYPE_TIME, &mysqlTime, sizeof(MYSQL_TIME),
false) ;

    if (mysql_stmt_bind_param(addAbsenceProcedure, param) != 0) {
        printStatementError(addAbsenceProcedure, "Bind Parametri
Impossibile per 'Aggiungi Assenza'") ;
        freeStatement(addAbsenceProcedure, false) ;
        return false ;
    }

    if (mysql_stmt_execute(addAbsenceProcedure) != 0) {
        printStatementError(addAbsenceProcedure, "Esecuzione Impossibile
per 'Aggiungi Assenza'") ;
        freeStatement(addAbsenceProcedure, false) ;
        return false ;
    }

    freeStatement(addAbsenceProcedure, true) ;

    return true ;
}


bool bookPrivateLessonInDatabase(PrivateLesson *lesson) {

    MYSQL_STMT *bookPrivateLessonProcedure ;
    if (!setupPreparedStatement(&bookPrivateLessonProcedure, "CALL
prenota_lezione_privata(?,?,?,?,?)", conn)) {
        printStatementError(bookPrivateLessonProcedure, "Impossibile
Preparare Procedura 'Prenota Lezione Privata'") ;
        return false ;
    }

    MYSQL_BIND param[5] ;
    MYSQL_TIME mysqlTime ;
    MYSQL_TIME mysqlDate ;

    prepareDateParam(&(lesson->lessonDate), &mysqlDate) ;
    bindParam(&param[0], MYSQL_TYPE_DATE, &mysqlDate, sizeof(MYSQL_TIME),
```

```
false) ;

    prepareTimeParam(&(lesson->startTime), &mysqlTime) ;
    bindParam(&param[1], MYSQL_TYPE_TIME, &mysqlTime, sizeof(MYSQL_TIME),
false) ;

    bindParam(&param[2], MYSQL_TYPE_STRING, lesson->lessonTeacher,
strlen(lesson->lessonTeacher), false) ;
    bindParam(&param[3], MYSQL_TYPE_LONG, &(lesson->lessonDurability),
sizeof(int), false) ;
    bindParam(&param[4], MYSQL_TYPE_STRING, lesson->lessonStudent,
strlen(lesson->lessonStudent), false) ;

    if (mysql_stmt_bind_param(bookPrivateLessonProcedure, param) != 0) {
        printStatementError(bookPrivateLessonProcedure, "Impossibile
Preparare I Parametri Per 'Prenota Lezione'") ;
        freeStatement(bookPrivateLessonProcedure, false) ;
        return false ;
    }

    if (mysql_stmt_execute(bookPrivateLessonProcedure) != 0) {
        printStatementError(bookPrivateLessonProcedure, "Impossibile
Eseguire Procedura 'Prenota Lezione'") ;
        freeStatement(bookPrivateLessonProcedure, false) ;
        return false ;
    }

    freeStatement(bookPrivateLessonProcedure, true) ;

    return true ;
}


DatabaseResult *getCourseAbsenceReportDB(int courseCode) {
    MYSQL_STMT *courseAbsenceReportProcedure ;
    if (!setupPreparedStatement(&courseAbsenceReportProcedure, "CALL
report_assenze_corso(?)", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Report
Assenze Corso'") ;
        return NULL ;
    }
```

```
    MYSQL_BIND param[1] ;
    bindParam(&param[0], MYSQL_TYPE_LONG, &courseCode, sizeof(int),
false) ;

    if (mysql_stmt_bind_param(courseAbsenceReportProcedure, param) != 0)
{
        printStatementError(courseAbsenceReportProcedure, "Impossibile
Bind Parametri per 'Report Assenze Corso'") ;
        freeStatement(courseAbsenceReportProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_execute(courseAbsenceReportProcedure) != 0) {
        printStatementError(courseAbsenceReportProcedure, "Impossibile
Eseguire 'Report Assenze Corso'") ;
        freeStatement(courseAbsenceReportProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_store_result(courseAbsenceReportProcedure) != 0) {
        printStatementError(courseAbsenceReportProcedure, "Impossibile
Salvare Risultato 'Report Assenze Corso'") ;
        freeStatement(courseAbsenceReportProcedure, true) ;
        return NULL ;
    }

    char studentName[STUDENT_NAME_MAX_LEN] ;
    int absenceNumber ;

    MYSQL_BIND returnParam[2] ;
    bindParam(&returnParam[0], MYSQL_TYPE_STRING, studentName,
STUDENT_NAME_MAX_LEN, false) ;
    bindParam(&returnParam[1], MYSQL_TYPE_LONG, &absenceNumber,
sizeof(int), false) ;
    if (mysql_stmt_bind_result(courseAbsenceReportProcedure, returnParam)
!= 0) {
        printStatementError(courseAbsenceReportProcedure, "Impossibile
Recuperare Risultato 'Report Assenze Corso'") ;
        freeStatement(courseAbsenceReportProcedure, true) ;
        return NULL ;
    }
```

```
    DatabaseResult *result = myMalloc(sizeof(DatabaseResult)) ;
    result->numRows = mysql_stmt_num_rows(courseAbsenceReportProcedure) ;
    result->rowsSet = myMalloc(sizeof(Student) * result->numRows) ;

    int hasResult = mysql_stmt_fetch(courseAbsenceReportProcedure) ;
    int i = 0 ;
    while (hasResult != 1 && hasResult != MYSQL_NO_DATA) {
        Student *student = myMalloc(sizeof(Student)) ;
        strcpy(student->studentName, studentName) ;
        student->studentAbsenceNumber = absenceNumber ;

        result->rowsSet[i] = student ;
        hasResult = mysql_stmt_fetch(courseAbsenceReportProcedure) ;
        i++ ;
    }

    if (hasResult == 1) {
        printStatementError(courseAbsenceReportProcedure, "Fetch
Impossibile Per 'Recupera Attività'") ;
        freeDatabaseResult(result) ;
        freeStatement(courseAbsenceReportProcedure, true) ;
        return NULL ;
    }

    freeStatement(courseAbsenceReportProcedure, true) ;

    return result ;
}


DatabaseResult *loadFreeTeachersFromDB(Date *date, Time *time, int
*duration) {
    MYSQL_STMT *loadFreeTeachersProcedure ;
    if (!setupPreparedStatement(&loadFreeTeachersProcedure, "CALL
recupera_insegnanti_liberi(?,?,?)", conn)) {
        printMysqlError(conn, "Impossibile Preparare Procedura 'Report
Assenze Corso'") ;
        return NULL ;
    }

    MYSQL_TIME mysqlDate ;
    MYSQL_TIME mysqlTime ;
```

```
    prepareDateParam(date, &mysqlDate) ;
    prepareTimeParam(time, &mysqlTime) ;

    MYSQL_BIND param[3] ;
    bindParam(&param[0], MYSQL_TYPE_DATE, &mysqlDate, sizeof(MYSQL_TIME),
false) ;
    bindParam(&param[1], MYSQL_TYPE_TIME, &mysqlTime, sizeof(MYSQL_TIME),
false) ;
    bindParam(&param[2], MYSQL_TYPE_LONG, duration, sizeof(int), false) ;

    if (mysql_stmt_bind_param(loadFreeTeachersProcedure, param) != 0) {
        printStatementError(loadFreeTeachersProcedure, "Bind Parametri
Impossibile Per 'Recupera Insegnanti Liberi'") ;
        freeStatement(loadFreeTeachersProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_execute(loadFreeTeachersProcedure) != 0) {
        printStatementError(loadFreeTeachersProcedure, "Esecuzione
Impossibile Per 'Recupera Insegnanti Liberi'") ;
        freeStatement(loadFreeTeachersProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_store_result(loadFreeTeachersProcedure) != 0) {
        printStatementError(loadFreeTeachersProcedure, "Store Risultato
Impossibile per 'Recupera Insegnanti Liberi'") ;
        freeStatement(loadFreeTeachersProcedure, true) ;
        return NULL ;
    }

    DatabaseResult *result = (DatabaseResult *)
myMalloc(sizeof(DatabaseResult)) ;
    result->numRows = mysql_stmt_num_rows(loadFreeTeachersProcedure) ;
    result->rowsSet = (void **) myMalloc(sizeof(char *) * result-
>numRows) ;

    char teacherName[TEACHER_NAME_MAX_LEN] ;
    MYSQL_BIND resultParam[1] ;
    bindParam(&resultParam[0], MYSQL_TYPE_STRING, teacherName,
TEACHER_NAME_MAX_LEN, false) ;
```

```
    if (mysql_stmt_bind_result(loadFreeTeachersProcedure, resultParam) !=
0) {
        printStatementError(loadFreeTeachersProcedure, "Bind Risultato
Impossibile Per 'Recupera Insegnanti Liberi'") ;
        freeStatement(loadFreeTeachersProcedure, false) ;
        return NULL ;
    }

    int hasResult = mysql_stmt_fetch(loadFreeTeachersProcedure) ;
    int i = 0 ;
    while (hasResult != 1 && hasResult != MYSQL_NO_DATA) {
        char *teacher = (char *) myMalloc(sizeof(char) *
(TEACHER_NAME_MAX_LEN)) ;
        strcpy(teacher, teacherName) ;

        result->rowsSet[i] = teacher ;
        hasResult = mysql_stmt_fetch(loadFreeTeachersProcedure) ;
        i++ ;
    }

    if (hasResult == 1) {
        printStatementError(loadFreeTeachersProcedure, "Fetch Impossibile
Per 'Recupera Insegnanti Liberi'") ;
        freeDatabaseResult(result) ;
        freeStatement(loadFreeTeachersProcedure, true) ;
        return NULL ;
    }

    freeStatement(loadFreeTeachersProcedure, true) ;

    return result ;
}
```

SecretaryDBHeader.h

```
#pragma once

#include "Connector.h"
#include "DatabaseUtilsHeader.h"
#include <stdbool.h>
#include <string.h>
```

```
#include "../utils/SystemUtilsHeader.h"


#include "../model/Student.h"
#include "../model/Activity.h"
#include "../model/Absence.h"
#include "../model/Partecipation.h"

bool addStudentToDatabase(Student *studentPtr) ;

bool addStudentJoinActivityToDatabase(char *studentName, int
*activityCode) ;



DatabaseResult *selectAllCourses() ;

DatabaseResult *getAllActivitiesFromDatabase() ;

bool addAbsenceToDatabase(Absence *newAbsence) ;


bool bookPrivateLessonInDatabase(PrivateLesson *privateLesson) ;

DatabaseResult *getCourseAbsenceReportDB(int courseCode) ;

DatabaseResult *loadFreeTeachersFromDB(Date *date, Time *time, int
*duration) ;
```

TeacherDB.c

```
#include "TeacherDBHeader.h"



DatabaseResult **generateAgendaFromDatabase(char *teacherUsername, int
weekIndex) {
    MYSQL_STMT *generateAgendaProcedure ;

    if (!setupPreparedStatement(&generateAgendaProcedure, "CALL
genera_agenda(?,?)", conn)) {
        printStatementError(generateAgendaProcedure, "Impossibile
Preparare Procedura 'Genera Agenda'") ;
```

```
        return false ;
    }

    MYSQL_BIND param[2] ;
    bindParam(&param[0], MYSQL_TYPE_STRING, teacherUsername,
strlen(teacherUsername), false) ;
    bindParam(&param[1], MYSQL_TYPE_LONG, &weekIndex, sizeof(int), false)
;

    if (mysql_stmt_bind_param(generateAgendaProcedure, param) != 0) {
        printStatementError(generateAgendaProcedure, "Impossibile Bind
Parametri Per 'Genera Agenda'") ;
        freeStatement(generateAgendaProcedure, false) ;
        return NULL ;
    }

    if (mysql_stmt_execute(generateAgendaProcedure) != 0) {
        printStatementError(generateAgendaProcedure, "Impossibile
Eseguire Procedure Per 'Genera Agenda'") ;
        freeStatement(generateAgendaProcedure, false) ;
        return NULL ;
    }

     int resultSetIndex = 0 ;
     DatabaseResult **resultArray = (DatabaseResult **)
myMalloc(sizeof(DatabaseResult *) * 7) ;
    while(resultSetIndex < 7) {

          // Gestione Errore Store Result
          if (mysql_stmt_store_result(generateAgendaProcedure) != 0) {
               printStatementError(generateAgendaProcedure, "Store
Risultato Impossibile per 'Genera Agenda'") ;
               for (int i = 0 ; i < 7 ; i++) {
                    freeDatabaseResult(resultArray[i]) ;
                    free(resultArray) ;
                    return NULL ;
               }
          }

          DatabaseResult *result = myMalloc(sizeof(DatabaseResult)) ;
          result->numRows = mysql_stmt_num_rows(generateAgendaProcedure)
;
```

```
          result->rowsSet = myMalloc(sizeof(GeneralLesson) * result-
>numRows) ;

          MYSQL_BIND returnParam[7] ;
          char lessonType[5] ;
          int classCode ;
          char classLevel[LEVEL_NAME_MAX_LEN] ;
          char student[STUDENT_NAME_MAX_LEN] ;
          MYSQL_TIME mysqlDate ;
          MYSQL_TIME mysqlStartTime ;
          MYSQL_TIME mysqlEndTime ;

          bindParam(&returnParam[0], MYSQL_TYPE_DATE, &mysqlDate,
sizeof(MYSQL_TIME), false) ;
          bindParam(&returnParam[1], MYSQL_TYPE_TIME, &mysqlStartTime,
sizeof(MYSQL_TIME), false) ;
          bindParam(&returnParam[2], MYSQL_TYPE_TIME, &mysqlEndTime,
sizeof(MYSQL_TIME), false) ;
          bindParam(&returnParam[3], MYSQL_TYPE_STRING, lessonType, 5,
false) ;
          bindParam(&returnParam[4], MYSQL_TYPE_LONG, &classCode,
sizeof(int), true) ;
          bindParam(&returnParam[5], MYSQL_TYPE_STRING, classLevel,
LEVEL_NAME_MAX_LEN, true) ;
          bindParam(&returnParam[6], MYSQL_TYPE_STRING, student,
STUDENT_NAME_MAX_LEN, true) ;

          if (mysql_stmt_bind_result(generateAgendaProcedure,
returnParam) != 0) {
               printStatementError(generateAgendaProcedure, "Impossibile
Recuperare il risultato da 'Genera Agenda'") ;
               freeStatement(generateAgendaProcedure, true) ;
               return NULL ;
          }

          int i = 0 ;
          int hasResult = mysql_stmt_fetch(generateAgendaProcedure) ;
          while (hasResult != 1 && hasResult != MYSQL_NO_DATA) {
               GeneralLesson *lesson = (GeneralLesson *)
myMalloc(sizeof(GeneralLesson)) ;

               getDateParam(&(lesson->lessonDate), &mysqlDate) ;
```

```
                getTimeParam(&(lesson->startTime), &mysqlStartTime) ;
                getTimeParam(&(lesson->endTime), &mysqlEndTime) ;
                if (strcmp(lessonType, "C") == 0) lesson->lessonType =
COURSE ;
                else lesson->lessonType = PRIVATE ;
                lesson->classCode = classCode ;
                strcpy(lesson->levelName, classLevel) ;
                strcpy(lesson->studentName, student) ;

                result->rowsSet[i] = lesson ;
                hasResult = mysql_stmt_fetch(generateAgendaProcedure) ;
                i++ ;
           }

           //Gestione Errore Del Fetch del Risultato
           if (hasResult == 1) {
                printStatementError(generateAgendaProcedure, "Fetch
Risultato Impossibile per 'Genera Agenda'") ;
                for (int i = 0 ; i < 7 ; i++) {
                     freeDatabaseResult(resultArray[i]) ;
                     free(resultArray) ;
                     return NULL ;
                }
           }

           resultArray[resultSetIndex] = result ;
           resultSetIndex++ ;

           mysql_stmt_free_result(generateAgendaProcedure) ;
           mysql_stmt_next_result(generateAgendaProcedure) ;

    }


    freeStatement(generateAgendaProcedure, true) ;

    return resultArray ;

}
```

TeacherDBHeader.h

```
#pragma once
```

```
#include "Connector.h"
#include "DatabaseUtilsHeader.h"

#include "../model/Lesson.h"
#include "../utils/SystemUtilsHeader.h"

DatabaseResult **generateAgendaFromDatabase(char *teacherUsername, int
weekIndex) ;
```

**Main**

Main.c

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

#include "MainHeader.h"
#include "../config/EnvironmentSetter.h"
#include "../db/Connector.h"
#include "../controller/LoginControllerHeader.h"
#include "../utils/TimeUtils.h"
#include "../utils/SystemUtilsHeader.h"
#include "../view/ViewUtilsHeader.h"

int main() {
    showAppHeader() ;
    if (!loadConfiguration()) {
        exitWithError("Errore Configurazione Ambiente") ;
    }
    if (connectToDatabase() && compileTimeRegex()) {
        loginController() ;
    }
}
```

MainHeader.h

```
#include <stdbool.h>

const char *APP_TITLE = "English School Management System" ;
```

```
extern bool loadConfiguration() ;
```

**Model**

Absence.h

```
#pragma once

#include "Student.h"
#include "Lesson.h"

typedef struct {
    int absenceCode ;
    Date absenceDate ;
    char studentName[STUDENT_NAME_MAX_LEN] ;
    enum DayOfWeek dayOfWeek ;
    Time startTime ;
    int classCode ;
} Absence ;
```

Activity.h

```
#pragma once

#include "../utils/TimeUtils.h"

enum ActivityType {
    FILM = 0,
    MEETING
} ;

#define FILM_TITLE_MAX_LEN 100 + 1
#define FILM_DIRECTOR_NAME_MAX_LEN 100 + 1
#define MEETING_LECTURER_NAME_MAX_LEN 100 + 1
#define MEETING_ARGUMENT_MAX_LEN 100 + 1

typedef struct {
    int activityCode ;
    Date activityDate ;
    Time activityTime ;
    enum ActivityType type ;
    char filmTitle[FILM_TITLE_MAX_LEN] ;
```

```
    char filmDirector[FILM_DIRECTOR_NAME_MAX_LEN] ;
    char meetingLecturer[MEETING_LECTURER_NAME_MAX_LEN] ;
    char meetingArgument[MEETING_ARGUMENT_MAX_LEN] ;
} CuturalActivity ;
```

Class.h

```
#pragma once


#include "Level.h"
#include "../utils/TimeUtils.h"

typedef struct {
    int classCode ;
    char levelName[LEVEL_NAME_MAX_LEN] ;
    Date activationDate ;
    int studentsNumber ;
} Class ;
```

Lesson.h

```
#pragma once

#include "Class.h"
#include "Teacher.h"
#include "Student.h"
#include "../utils/TimeUtils.h"


enum DayOfWeek {
    MONDAY = 0 ,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY,
    SUNDAY
} ;


typedef struct {
```

```
    enum DayOfWeek dayOfWeek ;
    Time startTime ;
    int classCode ;
    char classLevel[LEVEL_NAME_MAX_LEN] ;
    int lessonDuration ;
    char teacherName[TEACHER_NAME_MAX_LEN] ;
} ClassLesson ;


typedef struct {
    Date lessonDate ;
    Time startTime ;
    int lessonDurability ;
    char lessonTeacher[TEACHER_NAME_MAX_LEN] ;
    char lessonStudent[STUDENT_NAME_MAX_LEN] ;
} PrivateLesson ;


enum LessonType {
    COURSE = 0 ,
    PRIVATE = 1
} ;

typedef struct {
    Date lessonDate ;
    Time startTime ;
    Time endTime ;
    enum LessonType lessonType ;
    int classCode ;
    char levelName[LEVEL_NAME_MAX_LEN] ;
    char studentName[STUDENT_NAME_MAX_LEN] ;
} GeneralLesson ;

typedef struct {
    Date lessonDate ;
    Time startTime ;
    int duration ;
    enum LessonType lessonType ;
} ReportLesson ;
```

Level.h

```
#pragma once
```

```
#define LEVEL_NAME_MAX_LEN 45 + 1
#define LEVEL_BOOK_NAME_MAX_LEN 45 + 1

typedef struct {
    char levelName[LEVEL_NAME_MAX_LEN] ;
    char levelBookName[LEVEL_BOOK_NAME_MAX_LEN] ;
    short int levelHasExam ;
} Level ;
```

Partecipation.h

```
#pragma once

#include "Student.h"

typedef struct {
    char studentName[STUDENT_NAME_MAX_LEN] ;
    char studentTelephone[STUDENT_TELEPHONE_MAX_LEN] ;
    int activityCode ;
} Partecipation ;
```

Student.h

```
#pragma once

#include "Class.h"
#include "../utils/TimeUtils.h"

#define STUDENT_NAME_MAX_LEN 100 + 1
#define STUDENT_TELEPHONE_MAX_LEN 10 + 1


typedef struct {
    char studentName[STUDENT_NAME_MAX_LEN] ;
    char studentTelephone[STUDENT_TELEPHONE_MAX_LEN] ;
    int studentAbsenceNumber ;
    Date studentSubscribeDate ;
    Class studentClass ;
} Student ;
```

Teacher.h

```
#pragma once

#include "Class.h"


#define TEACHER_NAME_MAX_LEN 100 + 1
#define TEACHER_ADDRESS_MAX_LEN 100 + 1
#define TEACHER_NATIONALITY_MAX_LEN 100 + 1

typedef struct {
    char teacherName[TEACHER_NAME_MAX_LEN] ;
    char teacherNationality[TEACHER_ADDRESS_MAX_LEN] ;
    char teacherAddress[TEACHER_NATIONALITY_MAX_LEN] ;
} Teacher ;

typedef struct {
    char teacherName[TEACHER_NAME_MAX_LEN] ;
    char levelName[LEVEL_NAME_MAX_LEN] ;
    int classCode ;
} Teaching ;
```

User.h

```
#pragma once

#define USERNAME_MAX_SIZE 100 + 1
#define PASSWORD_MAX_SIZE 45 + 1

typedef struct {
    char username[USERNAME_MAX_SIZE] ;
    char password[PASSWORD_MAX_SIZE] ;
} User ;

typedef enum {
    AMMINISTRAZIONE = 0,
    SEGRETERIA,
    INSEGNANTE,
    LOGIN
} Role ;
```

**utils**

IOUtils.c

```
#include "IOUtils.h"


void printError(char *errorMessage) {
    colorPrint(errorMessage, RED_TEXT) ;
    printf("\n") ;
}


bool getUserInput(char *requestString, char *resultBuffer, int
bufferSize) {
    /*
        Function to get user input from stdinput.
        Takes max buffer size from stdin and put them in resultBuffer.
        bufferSize comprende nel conto lo \0
    */

    printf("%s", requestString) ;

    //Alloco un buffer di dimensione pari alla massima dimensione valida
per l'input più uno per lo \n
    char inputBuffer[bufferSize + 1] ;

    //Lettura al massimo di inputMaxSize - 1 caratteri incluso, se lo
trova, il \n
    if (fgets(inputBuffer, bufferSize + 1, stdin) == NULL) {
        printError("Errore Scansione Input") ;
        return false ;
    }

    //Rimozione eventuale \n letto da fgets
    for (int i = 0 ; i < (int) strlen(inputBuffer) ; i++) if
(inputBuffer[i] == '\n') inputBuffer[i] = '\0' ;

    /*
        Se ho una lunghezza di ciò che ho letto pari al massimo
leggibile,
        significa che sul canale di input è rimasto ALMENO lo \n,
        quindi devo rimuovere il resto dell'input per non leggerlo alla
```

```
lettura successiva.
        Significa inoltre che ho letto almeno un carattere in più della
dimensione massima prevista e quindi l'input
        non è valido
    */
    if ((int) strlen(inputBuffer) == bufferSize) {
        while(getchar() != '\n') ;
        printError("Input Inserito Troppo Lungo") ;
        return false ;
    }

    strcpy(resultBuffer, inputBuffer) ;

    /*
        Aggiunta controllo input non vuoto.
        Vantaggi:
            - evito di ricontrollare ogni volta che la funzione viene
chiamata
            - Se viene chiesto un input all'utente ci si aspetta che esso
venga inserito.
              Se non lo fa c'è un errore e posso ritornare false
    */
    if (strlen(resultBuffer) == 0) return false ;

    return true ;
}


void colorPrint(char *printText , TextColorEnum colorEnum) {
    int color = 0 ;
    switch (colorEnum) {
        case (RED_TEXT) :
            color = RED_TEXT_MARK ;
            break;
        case (GREEN_TEXT) :
            color = GREEN_TEXT_MARK ;
            break ;
        case (RED_HIGH) :
            color = RED_HIGH_TEXT_MARK ;
            break ;
        case (GREEN_HIGH) :
            color = GREEN_HIGH_TEXT_MARK ;
```

```
            break;
    }

    printf("\033[%dm%s\033[m", color, printText) ;
}


bool getDateFromUser(Date *datePtr, char *requestString) {
    char dateString[strlen("yyyy-mm-dd") + 1] ;
    if (!(getUserInput(requestString, dateString, strlen("yyyy-mm-dd") +
1))) {
        printError("Errore Inserimento Data") ;
        return false ;
    }

    if (!verifyAndParseDate(datePtr, dateString)) {
        printError("Formato della Data Inserita non Valido") ;
        return false ;
    }

    return true ;
}

bool getTimeFromUser(Time *timePtr, char *requestString) {

    char timeString[strlen("hh:mm") + 1] ;
    if (!(getUserInput(requestString, timeString, strlen("hh:mm") + 1)))
{
        printError("Errore Inserimento Orario") ;
        return false ;
    }

    if (!verifyAndParseTime(timePtr, timeString)) {
        printError("Formato Orario Inserito non Valido") ;
        return false ;
    }

    return true ;
}


bool getIntegerFromUser(int *integerPtr, char *inputMessage) {
```

```
    char integerStringBuff[11 + 1] ;
    if (!getUserInput(inputMessage, integerStringBuff, 11 + 1)) {
        printError("Errore Inserimento Codice Numerico") ;
        return false ;
    }

    char *checkString = "\0" ;
    long longInput = strtol(integerStringBuff, &checkString, 10) ;
    if (errno != 0) {
        printError("Numero Non Valido") ;
        errno = 0 ;
        return false ;
    }
    if (*checkString != '\0') {
        printError("Numero Non Valido") ;
        errno = 0 ;
        return false ;
    } ;
    if (longInput > INT_MAX || longInput < INT_MIN) return false ;
    *integerPtr = (int) longInput ;

    return true ;
}
```

IOUtils.h

```
#pragma once

#include <stdbool.h>
#include "TimeUtils.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stdbool.h>
#include <errno.h>
#include <ctype.h>
#include <limits.h>
#include "SystemUtilsHeader.h"


#define RED_TEXT_MARK 31
#define GREEN_TEXT_MARK 32
```

```
#define RED_HIGH_TEXT_MARK 41
#define GREEN_HIGH_TEXT_MARK 42

typedef enum {
    RED_TEXT ,
    GREEN_TEXT ,
    RED_HIGH ,
    GREEN_HIGH
} TextColorEnum ;


bool getUserInput(char *requestString, char *inputBuffer, int
inputMaxSize) ;

bool getDateFromUser(Date *datePtr, char *requestString) ;

bool getTimeFromUser(Time *timePtr, char *requestString) ;

bool getIntegerFromUser(int *classCodePtr, char *requestString) ;

void colorPrint(char *printText, TextColorEnum colorEnum) ;

void printError(char *errorMessage) ;
```

SystemUtils.c

```
#include "IOUtils.h"
#include <stdlib.h>


void exitWithError(char *errorMessage) {
    printError(errorMessage) ;
    exit(-1) ;
}

void *myMalloc(size_t allocSize) {
    void *returnPtr = malloc(allocSize) ;
    if (returnPtr == NULL) {
        exitWithError("ERRORE ALLOCAZIONE MEMORIA\n") ;
    }
    memset(returnPtr, 0, allocSize) ;
    return returnPtr ;
```

```
}
```

SystemUtilsHeader.h

```
#pragma once

void exitWithError(char *errorMessage) ;

void *myMalloc(size_t allocSize) ;
```

TimeUtils.c

```
#include "TimeUtils.h"
#include "IOUtils.h"

regex_t dateRegex ;
regex_t timeRegex ;


bool compileTimeRegex() {
    if (regcomp(&dateRegex, "[0-9]{4}-[0-9]{2}-[0-9]{2}", REG_EXTENDED) !
= 0) {
        printError("Impossibile Inizializzare Regex di Controllo Data") ;
        return false ;
    }
    if (regcomp(&timeRegex, "[0-9]{2}:[0-9]{2}", REG_EXTENDED) != 0) {
        printError("Impossibile Inizializzare Regex di Controllo Orario")
;
        return false ;
    }
    return true ;
}



bool verifyAndParseDate(Date *datePtr, char *dateString) {
    //Verifica su Correttezza rapporto tra mese-giorno e anno è fatta dal
DBMS
    if (regexec(&dateRegex, dateString, 0, NULL, 0) != 0) {
        return false ;
    }

    char *yearString = strtok(dateString, "-") ;
```

```
    char *monthString = strtok(NULL, "-") ;
    char *dayString = strtok(NULL, "-") ;

    if (strlen(yearString) == 4 && strlen(monthString) == 2 &&
strlen(dayString) == 2) {
        datePtr->year = atoi(yearString) ;
        datePtr->month = atoi(monthString) ;
        datePtr->day = atoi(dayString) ;
        return true ;
    }

    return false ;
}

bool verifyAndParseTime(Time *timePtr, char *timeStr) {
    if (regexec(&timeRegex, timeStr, 0, NULL, 0) != 0) {
        return false ;
    }

    char *hourString = strtok(timeStr, ":") ;
    char *minuteString = strtok(NULL, ":") ;

    if (strlen(hourString) == 2 && strlen(minuteString) == 2) {
        timePtr->hour = atoi(hourString) ;
        timePtr->minute = atoi(minuteString) ;
        return true ;
    }

    return false ;
}
```

TimeUtils.h

```
#pragma once

#include <stdbool.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <regex.h>


typedef struct {
```

```
    int year ;
    int month ;
    int day ;
} Date ;

typedef struct {
    int hour ;
    int minute ;
    int second ;
} Time ;


bool verifyAndParseDate(Date *datePtr, char *stringDate) ;

bool verifyAndParseTime(Time *timePtr, char *stringTime) ;

bool compileTimeRegex() ;
```

**view**

AdministrationView.c

```
#include "AdministrationViewHeader.h"



char *administrationMenuOptions[] = {
    "Aggiungi Livello",
    "Aggiungi Corso",
    "Aggiungi Insegnante",
    "Assegna Corso",
    "Aggiungi Lezione Corso",
    "Organizza Attività",
    "Report Insegnante" ,
    "Riavvio Anno Scolastico",
    "Quit"
    } ;


int getAdministrationOption() {
    return getUserOption(administrationMenuOptions, 9) ;
```

```
}

bool getLevelInfo(Level *levelPtr) {
    printOptionTitle(administrationMenuOptions[0]) ;

    if (!getUserInput("Inserire Nome Livello >>> ", levelPtr->levelName,
LEVEL_NAME_MAX_LEN)) {
        printError("Errore Lettura Nome Livello") ;
        return false ;
    }
    if (!getUserInput("Inserire Nome Libro >>> ", levelPtr-
>levelBookName, LEVEL_BOOK_NAME_MAX_LEN)) {
        printError("Errore Lettura Nome Libro") ;
        return false ;
    }

    char hasExamString[2] ;
    if (!getUserInput("Il Livello Ha Un Esame?? [y-n] >>> ",
hasExamString, 2)) {
        printError("Errore Lettura Esame") ;
        return false ;
    }

    if (strcmp(hasExamString, "y") == 0) levelPtr->levelHasExam = true ;
    else if (strcmp(hasExamString, "n") == 0) levelPtr->levelHasExam =
false ;
    else {
        printError("Opzione Inserita NON Valida") ;
        return false ;
    }

    return true ;
}


bool getClassInfo(Class *classPtr) {
    printOptionTitle(administrationMenuOptions[1]) ;

    if (!(getUserInput("Inserire Livello del Nuovo Corso >>> ", classPtr-
>levelName, LEVEL_NAME_MAX_LEN))) {
        printError("Errore Inserimento Nome Livello") ;
        return false ;
```

```
    }

    if (!getDateFromUser(&(classPtr->activationDate), "Inserire Data
Attivazione Nuovo Corso [yyyy-mm-dd] >>> ")) {
        return false ;
    }

    return true ;
}

bool getTeacherInfo(Teacher *teacherPtr, char *teacherUsername) {
    printOptionTitle(administrationMenuOptions[2]) ;

    if (!getUserInput("Inserire Nome Insegnante >>> ", teacherPtr-
>teacherName, TEACHER_NAME_MAX_LEN)) {
        printError("Errore Inserimento Nome Insegnante") ;
        return false ;
    }

    if (!getUserInput("Inserire Nazionalità Insegnante >>> ", teacherPtr-
>teacherNationality, TEACHER_NATIONALITY_MAX_LEN)) {
        printError("Errore Inserimento Nazionalità") ;
        return false ;
    }

    if (!getUserInput("Inserire Indirizzo Insegnante >>> ", teacherPtr-
>teacherAddress, TEACHER_ADDRESS_MAX_LEN)) {
        printError("Errore Inserimento Indirizzo") ;
        return false ;
    }

    if (!getUserInput("Inserire Username Insegnante >>> ",
teacherUsername, USERNAME_MAX_SIZE)) {
        printError("Errore Inserimento Username") ;
        return false ;
    }

    return true ;
}

bool getTeacherAndClassInfo(Teacher *teacherPtr, Class *classPtr) {
    printOptionTitle(administrationMenuOptions[3]) ;
```

```
    if (!getUserInput("Inserire Nome Insegnante >>> ", teacherPtr-
>teacherName, TEACHER_NAME_MAX_LEN)) {
        printError("Errore Inserimento Nome Insegnante") ;
        return false ;
    }

    if (!getIntegerFromUser(&(classPtr->classCode), "Inserire Codice
Corso >>> ")) {
        printError("Errore Lettura Codice Corso") ;
        return false ;
    }

    return true ;
}


bool getActivityInfo(CuturalActivity *activityPtr) {
    printOptionTitle(administrationMenuOptions[5]) ;

    char activityTypeString[2] ;
    if (!getUserInput("Inserire Tipo Di Attivita [P-C] >>> ",
activityTypeString, 2)) {
        printError("Errore Inserimento Tipo Attività") ;
        return false ;
    }

    if (strcmp(activityTypeString, "P") != 0 &&
strcmp(activityTypeString, "C") != 0) {
        printError("Tipo di Attività Non Valido") ;
        return false ;
    }

    if (strcmp(activityTypeString, "P") == 0) {
        activityPtr->type = FILM ;

        if (!getUserInput("Inserire Titolo Film >>> ", activityPtr-
>filmTitle, FILM_TITLE_MAX_LEN)) {
            printError("Errore Inserimento Titolo Film") ;
            return false ;
        }
        if (!getUserInput("Inserire Regista Film >>> ", activityPtr-
```

```
>filmDirector, FILM_DIRECTOR_NAME_MAX_LEN)) {
            printError("Errore Inserimento Regista Film") ;
            return false ;
        }
    }
    else {
        activityPtr->type = MEETING ;

        if (!getUserInput("Inserire Nome Conferenziere >>> ",
activityPtr->meetingLecturer, MEETING_LECTURER_NAME_MAX_LEN)) {
            printError("Errore Inserimento Conferenziere") ;
            return false ;
        }
        if (!getUserInput("Inserire Argomento Conferenza >>> ",
activityPtr->meetingArgument, MEETING_ARGUMENT_MAX_LEN)) {
            printError("Errore Inserimento Argomento Conferenza") ;
            return false ;
        }
    }

    if (!getDateFromUser(&(activityPtr->activityDate), "Inserire Data
Attività [yyyy-mm-dd] >>> ")) {
        return false ;
    }

    if (!getTimeFromUser(&(activityPtr->activityTime), "Inserire Orario
Attività [hh:mm] >>> ")) {
        return false ;
    }

    return true ;

}

bool getCourseLessonInfo(ClassLesson *newLesson) {
    printOptionTitle(administrationMenuOptions[4]) ;

    if (!getTimeFromUser(&(newLesson->startTime), "Inserire Orario Inizio
Lezione [hh:mm] >>> ")) {
        printError("Errore Presa Data") ;
        return false ;
    }
```

```
    if (!getIntegerFromUser(&(newLesson->lessonDuration), "Inserire
Durata Lezione In Minuti >>> ")) {
        printError("Impossibile Prendere Durata Lezione") ;
        return false ;
    }

    if (!getUserInput("Inserire Nome Insegnante >>> ", newLesson-
>teacherName,  TEACHER_NAME_MAX_LEN)) {
        printError("Impossibile Prendere Nome Insegnante") ;
        return false ;
    }

    if (!getIntegerFromUser(&(newLesson->classCode), "Inserire Codice
Corso >>> ")) {
        printError("Impossibile Leggere Codice Corso") ;
        return false ;
    }

    char dayOfWeekStr[3 + 1] ;
    if (!getUserInput("Inserire Giorno Settimana [Lun-Mar-Mer-Gio-Ven-
Sab-Dom] >>> ", dayOfWeekStr, 4)) {
        printError("Errore Lettura Giorno della Settimana") ;
        return false ;
    }

    if (!(strcmp(dayOfWeekStr, "Lun") || strcmp(dayOfWeekStr, "Mar") ||
strcmp(dayOfWeekStr, "Mer") || strcmp(dayOfWeekStr, "Gio") ||
strcmp(dayOfWeekStr, "Ven") || strcmp(dayOfWeekStr, "Sab") ||
strcmp(dayOfWeekStr, "Dom"))) {
        printError("Errore Giorno Settimana Inserito") ;
        return false ;
    }

    return true ;
}


bool getTeacherReportInfo(char *teacherName, int *yearPtr, int
*monthIndexPtr) {
    printOptionTitle(administrationMenuOptions[6]) ;
    if (!getUserInput("Inserire Nome Insegnante >>> ", teacherName,
```

```
TEACHER_NAME_MAX_LEN)) {
        printError("Errore Lettura Nome Insegnante") ;
        return false ;
    }

    if (!getIntegerFromUser(yearPtr, "Inserire Anno >>> ")) {
        printError("Errore Lettura Anno") ;
        return false ;
    }

    if (!getIntegerFromUser(monthIndexPtr, "Inserire Indice Mese [0=Gen ,
1=Feb , 2=Mar ecc] >>> ")) {
        printError("Errore Lettura Indice Mese") ;
        return false ;
    }

    return true ;
}


void printTeacherReport(ReportLesson **lessonArray, int arrayLen) {
    char *header[] = {"Data Lezione", "Orario Inizio", "Durata", "Tipo
Lezione"} ;
    enum TableFieldType types[] = {DATE, TIME, INT, STRING} ;
    Table *table = createTable(arrayLen, 4, header, types) ;

    for (int i = 0 ; i < arrayLen ; i++) {
        ReportLesson *lesson = lessonArray[i] ;
        setTableElem(table, i, 0, &(lesson->lessonDate)) ;
        setTableElem(table, i, 1, &(lesson->startTime)) ;
        setTableElem(table, i, 2, &(lesson->duration)) ;
        setTableElem(table, i, 3, (lesson->lessonType == COURSE) ?
"Corso" : "Privata") ;
    }

    printTable(table) ;
    freeTable(table) ;
}

void printAllTeaching(Teaching **teachingArray, int arrayLen) {
    char *header[] = {"Codice Corso", "Livello Corso", "Nome Insegnante"}
;
```

```
    enum TableFieldType types[] = {INT, STRING, STRING} ;
    Table *table = createTable(arrayLen, 3, header, types) ;

    for (int i = 0 ; i < arrayLen ; i++) {
        Teaching *teaching = teachingArray[i] ;
        setTableElem(table, i, 0, &(teaching->classCode)) ;
        setTableElem(table, i, 1, teaching->levelName) ;
        setTableElem(table, i, 2, teaching->teacherName) ;
    }

    printTable(table) ;
    freeTable(table) ;
}

void printNewClassCode(int *newClassCode) {
    char *header[] = {"Codice Nuovo Corso"} ;
    enum TableFieldType types[] = {INT} ;

    Table *table = createTable(1, 1, header, types) ;
    setTableElem(table, 0, 0, newClassCode) ;

    printTable(table) ;
    freeTable(table) ;
}

bool askRestartConfirm() {
    printOptionTitle(administrationMenuOptions[7]) ;
    char confirm[2] ;
    if (!getUserInput("Confermi di voler riavviare l'anno scolastico [y-
n] >>> ", confirm, 2)) {
        printError("Errore lettura conferma") ;
        return false ;
    }

    if (strcmp(confirm, "y") == 0) return true ;
    else return false ;
}


void printRestartSuccess() {
    colorPrint("Riavvio Anno Avvenuto Con Successo", GREEN_TEXT) ;
}
```

```
void printAllLevels(Level **levelsArray , int num) {

    char *header[] = {"Nome Livello", "Esame"} ;
    enum TableFieldType types[] = {STRING, STRING} ;
    Table *table = createTable(num, 2, header, types) ;

    for (int i = 0 ; i < num ; i++) {
        Level *level = levelsArray[i] ;
        setTableElem(table, i, 0, level->levelName) ;

        if (level->levelHasExam == 1) setTableElem(table, i, 1, "SI") ;
        else setTableElem(table, i, 1, "NO") ;

    }

    printTable(table) ;
    freeTable(table) ;
}

void printAllCoursesAdministration(Class **classArray, int arrayLen) {
    printAllCourses(classArray, arrayLen) ;
}

void printAllTeachers(Teacher **teachersArray, int arrayLen) {

    char *header[] = {"Nome", "Nazione"} ;
    enum TableFieldType types[] = {STRING, STRING} ;
    Table *table = createTable(arrayLen, 2, header, types) ;

    for (int i = 0 ; i < arrayLen ; i++) {
        Teacher *teacher = teachersArray[i] ;
        setTableElem(table, i, 0, teacher->teacherName) ;
        setTableElem(table, i, 1, teacher->teacherNationality) ;
    }

    printTable(table) ;

    freeTable(table) ;
}
```

```
void printNewActivityCode(int *activityCode) {
    char *header[] = {"Codice Attivita"} ;
    enum TableFieldType types[] = {INT} ;
    Table *table = createTable(1, 1, header, types) ;

    setTableElem(table, 0, 0, activityCode) ;

    printTable(table) ;
    freeTable(table) ;
}
```

AdministrationViewHeader.h

```
#pragma once

#include <stdbool.h>

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

#include "CommonViewHeader.h"
#include "ViewUtilsHeader.h"

#include "../utils/IOUtils.h"

#include "../model/Lesson.h"
#include "../model/Activity.h"
#include "../model/User.h"

int getAdministrationOption() ;

bool getLevelInfo(Level *levelPtr) ;

bool getClassInfo(Class *classPtr) ;

bool getTeacherInfo(Teacher *teacherPtr, char *teacherUsername) ;

bool getTeacherAndClassInfo(Teacher *teacherPtr, Class *classPtr) ;

bool getActivityInfo(CuturalActivity *activityPtr) ;
```

```
bool getCourseLessonInfo(ClassLesson *newLesson) ;

bool getTeacherReportInfo(char *teacherName, int *year, int
*monthIndex) ;

void printTeacherReport(ReportLesson **lessonArray, int arrayLen) ;

void printAllTeaching(Teaching **teachingArray, int arrayLen) ;

void printNewClassCode(int *newClassCode) ;

bool askRestartConfirm() ;

void printRestartSuccess() ;

void printAllLevels(Level **levelArray, int num) ;

void printAllCoursesAdministration(Class **classArray, int arrayLen) ;

void printAllTeachers() ;

void printNewActivityCode(int *activityCode) ;
```

CommonView.c

```
#include "CommonViewHeader.h"

void printAllCourses(Class **classArray, int arrayLen) {
     char *headerArray[] = {"Codice", "Livello", "Numero Allievi", "Data
Attivazione"} ;
    enum TableFieldType tableField[] = {INT, STRING, INT, DATE} ;

    Table *table = createTable(arrayLen, 4, headerArray, tableField) ;
    for(int i = 0 ; i < arrayLen ; i++) {
        Class *classPtr = classArray[i] ;
        setTableElem(table, i, 0, &(classPtr->classCode)) ;
        setTableElem(table, i, 1, classPtr->levelName) ;
        setTableElem(table, i, 2, &(classPtr->studentsNumber)) ;
        setTableElem(table, i, 3, &(classPtr->activationDate)) ;
    }

    printTable(table) ;
    freeTable(table) ;
```

```
}
```

CommonViewHeader.h

```
#pragma once

#include "TablePrinterHeader.h"
#include "../model/Class.h"

void printAllCourses(Class **classArray, int arrayLen) ;
```

LoginView.c

```
#include "LoginView.h"



bool showLoginView(User *loginCredentialsPtr) {

    /*
        ATTENTO!! Per permettere la lettura di esattamente un massimo di
x byte, imposta una dimensione +2 da riservare
        per il terminatore di stringa \0
    */
    printf("Inserire Credenziali Di Accesso\n") ;

    if (!getUserInput("Username >>> ", loginCredentialsPtr->username,
USERNAME_MAX_SIZE)) {
        printError("Errore Lettura Username") ;
        return false ;
    }
    if (!getUserInput("Password >>> ", loginCredentialsPtr->password,
PASSWORD_MAX_SIZE)) {
        printError("Errore Lettura Password") ;
        return false ;
    }

    return true ;

}
```

LoginView.h

```
#pragma once

#include "../model/User.h"

#include "../utils/IOUtils.h"
#include "../utils/SystemUtilsHeader.h"

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

bool showLoginView(User *loginCredentialsPtr) ;
```

SecretaryView.c

```
#include "SecretaryViewHeader.h"


char *secretaryMenuOption[] = {
    "Aggiungi Allievo",
    "Aggiungi Partecipazione Ad Attività",
    "Prenota Lezione Privata",
    "Aggiungi Assenza",
    "Report Assenze Corso",
    "Report Insegnanti Liberi",
    "Quit"
    } ;


int getSecretaryOption() {
    return getUserOption(secretaryMenuOption, 7) ;
}

bool getStudentInfo(Student *studentPtr) {
    printOptionTitle(secretaryMenuOption[0]) ;

    if (!getUserInput("Inserire Nome Nuovo Studente >>> ", studentPtr-
>studentName, STUDENT_NAME_MAX_LEN)) {
        printError("Errore Lettura Nome Studente") ;
        return false ;
    }
```

```
    if (!getUserInput("Inserire Numero di Telefono >>> ", studentPtr-
>studentTelephone, STUDENT_TELEPHONE_MAX_LEN)) {
        printError("Errore Lettura Numero di Telefono") ;
        return false ;
    }
    for (int i = 0 ; i < (int) strlen(studentPtr->studentTelephone) ; i+
+) {
        if (!isdigit(studentPtr->studentTelephone[i])) return false ;
    }
    if ((int) strlen(studentPtr->studentTelephone) != 10) {
        printError("I Numeri di Telefono Hanno 10 Cifre") ;
        return false ;
    }

    if (!getDateFromUser(&(studentPtr->studentSubscribeDate), "Inserire
Data di Iscrizione [yyyy-mm-dd] >>> ")) {
        return false ;
    }

    if (!getIntegerFromUser(&(studentPtr->studentClass.classCode),
"Inserire Codice Corso >>> ")) {
        printError("Errore Inserimento Codice Corso") ;
        return false ;
    }

    return true ;
}

bool getStudentJoinActivityInfo(char *studentName, int *activityCodePtr)
{
    printOptionTitle(secretaryMenuOption[1]) ;

    if (!getUserInput("Inserire Nome Allievo >>> ", studentName,
STUDENT_NAME_MAX_LEN)) {
        printError("Errore Presa Nome Allievo") ;
        return false ;
    }

    if (!getIntegerFromUser(activityCodePtr, "Inserire Codice Attività
>>> ")) {
        printError("Errore Inserimento Codice Attivita") ;
```

```
        return false ;
    }

    return true ;
}


bool getPrivateLessonInfo(PrivateLesson *lesson) {
    printOptionTitle(secretaryMenuOption[2]) ;

    if (!getUserInput("Inserire Nome Allievo >>> ", lesson-
>lessonStudent, STUDENT_NAME_MAX_LEN)) {
        printError("Errore Lettura Studente") ;
        return false ;
    }

    if (!getUserInput("Inserire Nome Insegnante >>> ", lesson-
>lessonTeacher, TEACHER_NAME_MAX_LEN)) {
        printError("Errore Lettura Insegnante") ;
        return false ;
    }

    if (!getDateFromUser(&(lesson->lessonDate), "Inserire Data Lezione
[yyyy-mm-dd] >>> ")) {
        printError("Errore Lettura Data Lezione") ;
        return false ;
    }

    if (!(getTimeFromUser(&(lesson->startTime), "Inserire Orario Lezione
[hh:mm] >>> "))) {
        printError("Errore Lettura Orario Lezione") ;
        return false ;
    }

    if (!getIntegerFromUser(&(lesson->lessonDurability), "Inserire Durata
Lezione In Minuti >>> ")) {
        printError("Errore Lettura Durata Lezione") ;
        return false ;
    }

    return true ;
}
```

```
bool getCourseAbsenceReportInfo(int *courseCode) {
    printOptionTitle(secretaryMenuOption[4]) ;

    if (!getIntegerFromUser(courseCode, "Inserire Codice Corso >>> ")) {
        printError("Errore Lettura Codice Corso") ;
        return false ;
    }

    return true ;
}

bool getAbsenceInfo(Absence *absencePtr) {
    printOptionTitle(secretaryMenuOption[3]) ;

    if (!getUserInput("Inserire Nome Allievo >>> ", absencePtr-
>studentName, STUDENT_NAME_MAX_LEN)) {
        printError("Errore Lettura Nome Allievo") ;
        return false ;
    }

    if (!getDateFromUser(&(absencePtr->absenceDate), "Inserire Data
Assenza [yyyy-mm-dd] >>> ")) {
        printError("Errore Lettura Data Assenza") ;
        return false ;
    }

    if (!getTimeFromUser(&(absencePtr->startTime), "Inserire Orario
Assenza [hh:mm] >>> ")) {
        printError("Errore Lettura Orario Assenza") ;
        return false ;
    }

    return true ;
}


bool getFreeTeacherReportInfo(Date *datePtr, Time *timePtr, int
*durationPtr) {
    printOptionTitle(secretaryMenuOption[5]) ;
    if (!getDateFromUser(datePtr, "Inserire Data [yyyy-mm-dd] >>> ")) {
```

```
        printError("Errore Lettura Data Lezione") ;
        return false ;
    }

    if (!getTimeFromUser(timePtr, "Inserire Orario [hh:mm] >>> ")) {
        printError("Errore Lettura Orario Lezione") ;
        return false ;
    }

    if (!getIntegerFromUser(durationPtr, "Inserire Durata Impegno >>> "))
{
        printError("Errore Lettura Durata") ;
        return false ;
    }

    return true ;
}



void printFreeTeacherReport(char **teacherNameArray, int arrayLen) {
    char *header[] = {"Nome Insegnante"} ;
    enum TableFieldType types[] = {STRING} ;
    Table *table = createTable(arrayLen, 1, header, types) ;
    for (int i = 0 ; i < arrayLen ; i++) {
        setTableElem(table, i, 0, teacherNameArray[i]) ;
    }

    printTable(table) ;
    freeTable(table) ;
}



void printCourseAbsenceReport(Student **studentArray, int arrayLen) {
    char *header[] = {"Nome Allievo", "Numero Assenze"} ;
    enum TableFieldType types[] = {STRING, INT} ;
    Table *table = createTable(arrayLen, 2, header, types) ;

    for (int i = 0 ; i < arrayLen ; i++) {
        Student *student = studentArray[i] ;
        setTableElem(table, i, 0, student->studentName) ;
        setTableElem(table, i, 1, &(student->studentAbsenceNumber)) ;
```

```
    }

    printTable(table) ;
    freeTable(table) ;
}

void printAllActivities(CuturalActivity **activityArray, int arrayLen) {
    char *headerArray[] = {"Codice", "Data", "Orario", "Tipo", "Titolo
Film", "Regista", "Conferenziere", "Argomento Conferenza"} ;
    enum TableFieldType typesArray[] = {INT, DATE, TIME, STRING, STRING,
STRING, STRING, STRING} ;
    Table *table = createTable(arrayLen, 8, headerArray, typesArray) ;

    for (int i = 0 ; i < arrayLen ; i++) {
        CuturalActivity *activity = activityArray[i] ;
        setTableElem(table, i, 0, &(activity->activityCode)) ;
        setTableElem(table, i, 1, &(activity->activityDate)) ;
        setTableElem(table, i, 2, &(activity->activityTime)) ;
        setTableElem(table, i, 4, activity->filmTitle) ;
        setTableElem(table, i, 5, activity->filmDirector) ;
        setTableElem(table, i, 6, activity->meetingLecturer) ;
        setTableElem(table, i, 7, activity->meetingArgument) ;

        if (activity->type == FILM) setTableElem(table, i, 3, "F") ;
        else setTableElem(table, i, 3, "C") ;
    }

    printTable(table) ;
    freeTable(table) ;
}
```

SecretaryViewHeader.h

```
#pragma once


#include "../utils/IOUtils.h"

#include "ViewUtilsHeader.h"
#include "TablePrinterHeader.h"

#include <errno.h>
#include <stdbool.h>
```

```
#include <ctype.h>

#include "../model/Student.h"
#include "../model/Lesson.h"
#include "../model/Absence.h"
#include "../model/Partecipation.h"
#include "../model/Activity.h"
#include "../model/Class.h"


int getSecretaryOption() ;

bool getStudentInfo(Student *studentPtr) ;

bool getStudentJoinActivityInfo(char *studentName, int
*activityCodePtr) ;

bool getPrivateLessonInfo(PrivateLesson *lessonPtr) ;

bool getCourseAbsenceReportInfo(int *courseCode) ;

bool getAbsenceInfo(Absence *absencePtr) ;

bool getFreeTeacherReportInfo(Date *datePtr, Time *timePtr, int
*durationPtr) ;

bool getActivityParticipantsReportInfo(int *activityCode) ;

void printActivityParticipantsReport(Partecipation **partecipationArray,
int arrayLen) ;

void printFreeTeacherReport(char **teacherNameArray, int arrayLen) ;

void printCourseAbsenceReport(Student **studentArray, int arrayLen) ;

void printAllActivities(CuturalActivity **activityArray, int arrayLen) ;

void printAllCourses(Class **classArray, int arrayLen) ;
```

TablePrinter.c

```
#include "TablePrinterHeader.h"

const char *nullString = "" ;

int findColMaxSize(void ***table, char **headerArray, int rowsNum, int
colIndex, enum TableFieldType colType) {
    int maxSize = strlen(headerArray[colIndex]) ;

    for (int rowIndex = 0 ; rowIndex < rowsNum ; rowIndex++) {
        void *currentElem = table[rowIndex][colIndex] ;
        int currentSize ;

        if (currentElem == NULL) {
            currentSize = strlen(nullString) ;
            if (maxSize < currentSize) maxSize = currentSize ;
            continue;
        }

        char numBuff[50] ;
        switch (colType) {
            case STRING :
                currentSize = strlen((char *) currentElem) ;
                break ;
            case INT :
                sprintf(numBuff, "%d", *((int *)currentElem)) ;
                currentSize = strlen(numBuff) ;
                break ;
            case FLOAT :
                sprintf(numBuff, "%.2f", *((float *)currentElem)) ;
                currentSize = strlen(numBuff) ;
                break ;
            case DATE :
                currentSize = strlen("yyyy-mm-dd") ;
                break ;
            case TIME :
                currentSize = strlen("hh:mm") ;
                break ;

        }

        if (maxSize < currentSize) maxSize = currentSize ;
```

```
    }

    return maxSize ;
}

void printSeparator(int *widthArray, int colNum) {
    putchar('+') ;
    for (int colIndex = 0 ; colIndex < colNum ; colIndex++) {
        for (int i = 0 ; i < widthArray[colIndex] ; i++) putchar('-') ;
        putchar('+') ;
    }
    putchar('\n') ;
}

void printHeader(char **headerArray, int *widthArray, int colsNum) {
    putchar('|') ;
    for (int colIndex = 0 ; colIndex < colsNum ; colIndex++) {
        printf(" \033[96m%-*s\033[m", widthArray[colIndex] - 1,
headerArray[colIndex]) ;
        putchar('|') ;
    }
    putchar('\n') ;
}

void printRow(void ***table, int rowIndex, int colNum, int *widthArray,
enum TableFieldType *rowStruct) {
    putchar('|') ;

    char dateBuff[50] ;
    char timeBuff[50] ;
    Date *datePtr ;
    Time *timePtr ;
    for (int colIndex = 0 ; colIndex < colNum ; colIndex++) {
        void *currentElem = table[rowIndex][colIndex] ;
        enum TableFieldType currentType = rowStruct[colIndex] ;
        int fieldSize = widthArray[colIndex] - 1 ;

        if (currentElem == NULL) {
            printf(" %-*s", fieldSize, nullString) ;
            putchar('|') ;
            continue;
        }
```

```
        switch (currentType) {
            case STRING :
                printf(" %-*s", fieldSize, (char *) currentElem) ;
                break ;
            case INT :
                printf(" %-*d", fieldSize, *((int *) currentElem)) ;
                break; ;
            case FLOAT :
                printf(" %-*.2f", fieldSize, *((float *) currentElem)) ;
                break ;
            case DATE :
                datePtr = (Date *) currentElem ;
                sprintf(dateBuff, "%d-%02d-%02d", datePtr->year, datePtr-
>month, datePtr->day) ;
                printf(" %-*s", fieldSize, dateBuff) ;
                break ;
            case TIME :
                timePtr = (Time *) currentElem ;
                sprintf(timeBuff, "%02d:%02d", timePtr->hour, timePtr-
>minute) ;
                printf(" %-*s", fieldSize, timeBuff) ;
                break ;
        }
        //Usato -1 per inserire lo spazio aggiuntivo a dx e dare più
visibilità all'output
        putchar('|') ;

    }
    putchar('\n') ;
}


void printTable(Table *tablePtr) {

    int rowsNum = tablePtr->rowsNum ;
    int colsNum = tablePtr->colsNum ;
    char **headerArray = tablePtr->headerArray ;
    enum TableFieldType *rowTypes = tablePtr->rowStruct ;

    void ***table = tablePtr->table ;
```

```
    int maxWidthArray[colsNum] ;

    for (int colIndex = 0 ; colIndex < colsNum ; colIndex++) {
        int colMaxSize = findColMaxSize(table, headerArray, rowsNum,
colIndex, rowTypes[colIndex]) ;
        maxWidthArray[colIndex] = 5 + colMaxSize ;
    }

    printSeparator(maxWidthArray, colsNum) ;
    printHeader(headerArray, maxWidthArray, colsNum) ;

    for (int rowIndex = 0 ; rowIndex < rowsNum ; rowIndex++) {
        printSeparator(maxWidthArray, colsNum) ;
        printRow(table, rowIndex, colsNum, maxWidthArray, rowTypes) ;
    }
    printSeparator(maxWidthArray, colsNum) ;
}


Table *createTable(int rowsNum, int colsNum, char **headerArray, enum
TableFieldType *rowTypes) {

    Table *tablePtr = myMalloc(sizeof(Table)) ;

    tablePtr->table = (void ***) myMalloc(sizeof(void **) * rowsNum) ;

    for (int rowIndex = 0 ; rowIndex < rowsNum ; rowIndex++) tablePtr-
>table[rowIndex] = (void **) myMalloc(sizeof(void *) * colsNum) ;

    tablePtr->colsNum = colsNum ;
    tablePtr->rowsNum = rowsNum ;
    tablePtr->headerArray = headerArray ;
    tablePtr->rowStruct = rowTypes ;

    return tablePtr ;
}

void freeTable(Table *tablePtr) {

    for (int rowIndex = 0 ; rowIndex < tablePtr->rowsNum ; rowIndex++) {
        free(tablePtr->table[rowIndex]) ;
    }
```

```
    free(tablePtr) ;
}

void setTableElem(Table *tablePtr, int rowIndex, int colIndex, void
*elemPtr) {
    (tablePtr->table)[rowIndex][colIndex] = elemPtr ;
}
```

TablePrinterHeader.h

```
#pragma once

#include <stdio.h>
#include <string.h>
#include "../utils/TimeUtils.h"
#include "../utils/SystemUtilsHeader.h"

enum TableFieldType {
    STRING = 0,
    INT,
    DATE,
    TIME,
    FLOAT
} ;

typedef struct {
    int rowsNum ;
    int colsNum ;
    char **headerArray ;
    void ***table ;
    enum TableFieldType *rowStruct ;
} Table ;


void printTable(Table *tablePtr) ;

Table *createTable(int rowsNum, int colsNum, char **headerArray, enum
TableFieldType *rowTypes) ;

void freeTable(Table *tablePtr) ;

void setTableElem(Table *tablePtr, int rowIndex, int colIndex, void
```

```
*elemPtr) ;
```

TeacherView.c

```
#include "TeacherViewHeader.h"


char *teacherMenuOptions[] = {"Generazione Agenda", "Quit"} ;

int getTeacherOption() {
    return getUserOption(teacherMenuOptions, 2) ;
}

bool getAgendaInfo(int *weekIndexPtr) {
    printOptionTitle(teacherMenuOptions[0]) ;
    if (!getIntegerFromUser(weekIndexPtr, "Inserire Indice Settimana
[0=attuale, 1=prossima, 2=due settimane ecc] >>> ")) {
        printError("Errore Lettura Indice Settimana") ;
        return false ;
    }

    return true ;
}

void printAgenda(GeneralLesson **lessonArray, int arrayLen) {
    char *header[] = {"Data", "Inizio", "Fine", "Tipo", "Codice Corso",
"Livello Corso", "Allievo"} ;
    enum TableFieldType types[] = {DATE, TIME, TIME, STRING, INT, STRING,
STRING} ;
    Table *table = createTable(arrayLen, 7, header, types) ;

    for (int i = 0 ; i < arrayLen ; i++) {
        GeneralLesson *lesson = lessonArray[i] ;
        setTableElem(table, i, 0, &(lesson->lessonDate)) ;
        setTableElem(table, i, 1, &(lesson->startTime)) ;
        setTableElem(table, i, 2, &(lesson->endTime)) ;
        setTableElem(table, i, 3, &(lesson->lessonType)) ;
        if (lesson->lessonType == COURSE) {
            setTableElem(table, i, 3, "Corso") ;
            setTableElem(table, i, 4, &(lesson->classCode)) ;
            setTableElem(table, i, 5, lesson->levelName) ;
            setTableElem(table, i, 6, NULL) ;
        }
```

```
        else {
            setTableElem(table, i, 3, "Privata") ;
            setTableElem(table, i, 4, NULL) ;
            setTableElem(table, i, 5, NULL) ;
            setTableElem(table, i, 6, lesson->studentName) ;
        }
    }

    printTable(table) ;
    freeTable(table) ;
    printf("\n") ;
}
```

TeacherViewHeader.h

```
#pragma once

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "../utils/SystemUtilsHeader.h"
#include "../utils/IOUtils.h"

#include "ViewUtilsHeader.h"
#include "TablePrinterHeader.h"

#include "../model/Lesson.h"


int getTeacherOption() ;

bool getAgendaInfo(int *weekIndexPtr) ;

void printAgenda(GeneralLesson **lessonArray, int arrayLen) ;
```

ViewUtils.c

```
#include "ViewUtilsHeader.h"

void showMenu(char **menuOptionsArray, int optionsNumber) {
```

```
    for (int i = 0 ; i < optionsNumber ; i++) {
        printf("%d. %s\n", i, menuOptionsArray[i]) ;
    }
    printf("Scegliere Opzione [%d-%d] >>> ", 0, optionsNumber - 1) ;
}

void showOptionHeader() {
    int headerSize = 75 ;
    printf("\n") ;
    for (int i = 0 ; i < headerSize ; i ++) putchar('*') ;
    printf("\n") ;
}

void printOptionTitle(char *optionName) {
    colorPrint(optionName, GREEN_TEXT) ;
    printf("\n") ;
}

int getUserOption(char *menuOption[], int menuLen) {
    printf("\n") ;
    colorPrint("Cosa Posso Fare Per Te??", GREEN_HIGH) ;
    printf("\n") ;
    showMenu(menuOption, menuLen) ;

    int selectedOption ;
    if (!getIntegerFromUser(&selectedOption, "")) {
        selectedOption = -1 ;
    }

    return selectedOption ;
}

void clearScreen() {
    printf("\033[2J\033[H");
}

void printHeaderLine(int headerLenght) {
    for (int i = 0 ; i < headerLenght ; i++) {
        if (i == 0 || i == headerLenght - 1) putchar('+') ;
        else putchar('-') ;
    }
}
```

```
void showAppHeader() {
    char *appTitleHeader = "| English School Management System |" ;
    printHeaderLine(strlen(appTitleHeader)) ;
    printf("\n%s\n", appTitleHeader) ;
    printHeaderLine(strlen(appTitleHeader)) ;
    printf("\n") ;
}
```

ViewUtilsHeader.h

```
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include "../utils/IOUtils.h"

void showMenu(char **menuOptionsArray, int optionsNumber) ;
void showOptionHeader() ;

void printOptionTitle(char *optionName) ;

int getUserOption(char *menuOption[], int menuLen) ;

void clearScreen() ;

void showAppHeader() ;
```