



Basi di Dati e Conoscenza

Progetto A.A. 2019/2020

# BASE DI DATI PER GESTIONE DEI CORSI DI LINGUA INGLESE PRESSO UN ISTITUTO DI INSEGNAMENTO

0255041

Alessandro Di Filippo

## Indice

1. Descrizione del Minimondo .....	2
2. Analisi dei Requisiti .....	3
3. Progettazione concettuale .....	7
4. Progettazione logica .....	11
5. Progettazione fisica .....	21
Appendice: Implementazione .....	45

## 1. Descrizione del Minimondo

- 1 Si progetti un sistema informativo per la gestione dei corsi di lingua inglese, tenuti presso un istituto 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.
- 10 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.
- 20 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.
- 30 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
- 34 generare dei report indicanti la propria agenda, su base settimanale.

## 2. Analisi dei Requisiti

### Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
2	Istituto di insegnamento	scuola	Il termine istituto di insegnamento non solo è troppo lungo e complicato ma non è mai usato nel testo, mentre il termine scuola è di uso più comune e presente maggiormente nel testo.
14	Giorni e orari	Giorni corso, orari corso	Per evitare confusione con giorni e orari relative ad altre attività e/o entità.
14	Iscritti	Allievi	Per semplicità unifichiamo tutti i termini riferiti agli studenti sotto il termine di allievi.
18	nome	Nome allievo	Per evitare confusioni con nomi di altre entità.
26	Codice progressivo	Chiave identificativa	L'utilizzo del termine codice progressivo è già utilizzato per identificare il codice del corso.
26	Giorno e l'ora	Giorno attività, ora attività	Come prima per evitare confusioni con giorni e orari relativi ad altro.
29	isciversi	Iscrizione ad attività	Per separare l'iscrizione ad un'attività dall'iscrizione ad un corso.
33	utenti	Allievi	Per semplicità unifichiamo tutti i termini riferiti agli studenti sotto il termine di allievi.
33	Iscrizioni.....ai corsi	Iscrizioni ai corsi	Per rendere il periodo più semplice.
34	report	Report insegnanti	Per differenziare il report effettuato dal personale amministrativo da quello effettuato dagli insegnanti.

### Specifica disambiguata

Si progetti un sistema informativo per la gestione dei corsi di lingua inglese, tenuti presso una **scuola**. 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 **allievi** e l'elenco dei **giorni corso** ed **orari corso** 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 dell'allievo**, 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 una **chiave identificativa**, e sono noti il **giorno attività** e **l'ora attività** 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 effettuare **l'iscrizione all'attività**.

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 ai corsi** degli **allievi** della scuola. Gli insegnanti possono generare dei **report insegnanti** indicanti la propria agenda, su base settimanale.

## Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Scuola	Dove vengono impartiti I corsi di lingue	Istituto di insegnamento	
Corso	Lezioni svolte all'interno della scuola di lingue a cui afferiscono gli allievi e gli insegnanti	Lezioni	Allievi, insegnanti, livelli

Insegnanti	Coloro che impartiscono corsi e lezioni private all'interno della scuola		Corsi, lezioni private,
Giorni corso e orari corso	Quando si svolgono i corsi		Corso
Nome allievo	Nome allievo	Nome	Allievi
Allievi	Gli student della scuola	Iscritti, utenti	Corsi, Attività culturali
Chiave Identificativa	Codice identificativo per le attività culturali extra scolastiche	Codice progressivo	Attività culturali
Giorno attività, ora attività	Quando si svolgono le attività culturali		Attività culturali
Iscrizione attività	Iscrizione alle attività	Iscriversi	Attività culturali, Allievi
Iscrizioni ai corsi	Iscrizione ai corsi		Allievi
Report insegnanti	Resoconto dell'attività degli insegnanti	Report	Insegnanti
Personale amministrativo	Addetti all'inserimento dei dati nella base di dati	Personale di segreteria	
Attività culturali	Tipo di attività culturale	Proiezioni, conferenze	Allievi

## Raggruppamento dei requisiti in insiemi omogenei

### Frasi relative a "Corsi"

Sono organizzati per livelli. Ciascun livello è identificato dal nome del livello stesso (ad esempio Elementary, Intermediate, First Certificate, Advanced, Proficiency).

è specificato il nome del libro di testo e se viene richiesto di sostenere un esame finale.

Sono identificati univocamente dal nome del livello cui afferiscono e da un codice progressivo, necessario per distinguere corsi che fanno riferimento allo stesso livello.

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 ciascun corso sono note la data di attivazione, il numero e le informazioni anagrafiche degli iscritti e l'elenco dei giorni corso ed orari corso 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.

Per gli allievi sono noti il nome, un recapito, il corso a cui sono iscritti.

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.

#### **Frasi relative a "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.

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.

Per poter partecipare alle attività gli allievi devono iscriversi.

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

#### **Frasi relative a "Insegnanti"**

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.

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.

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.

Gli insegnanti possono generare dei report indicanti la propria agenda, su base settimanale.

#### **Frasi relative a "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.

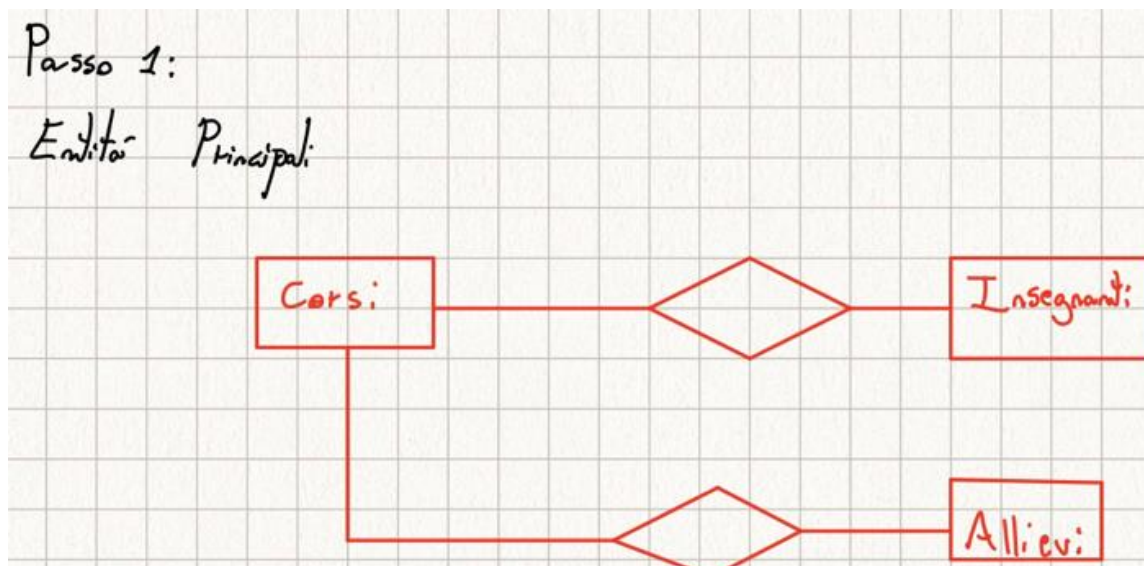
Nel caso di conferenze, sono noti l'argomento che verrà trattato ed il nome del conferenziere.

Per poter partecipare alle attività gli allievi devono effettuare l'iscrizione all'attività.

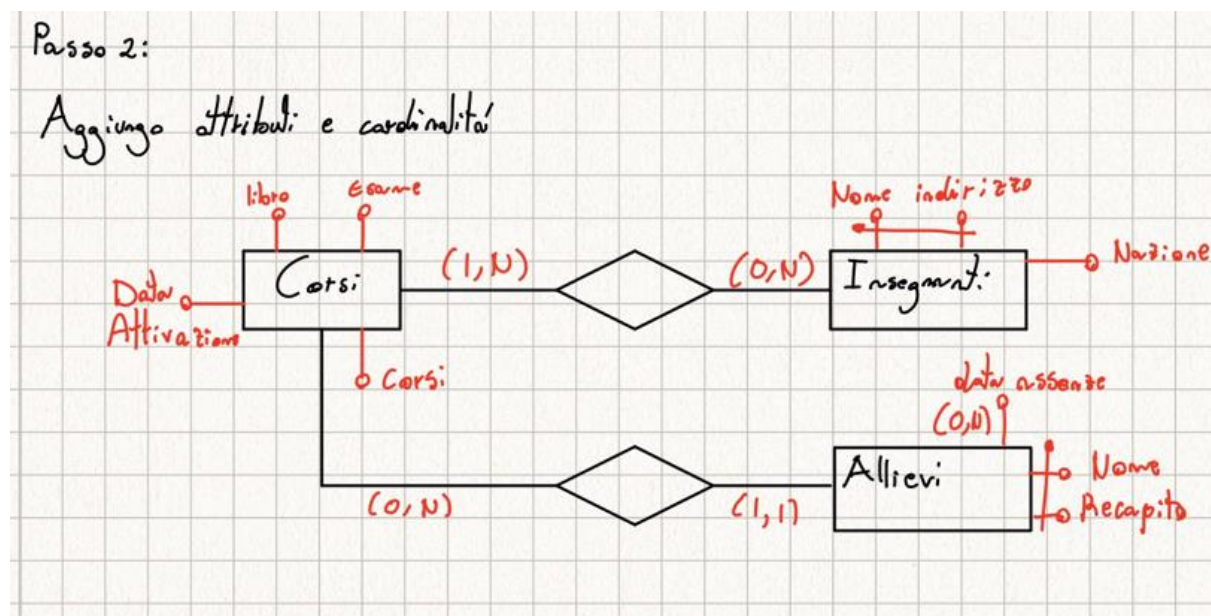
### 3. Progettazione concettuale

#### Costruzione dello schema E-R

##### Passo 1



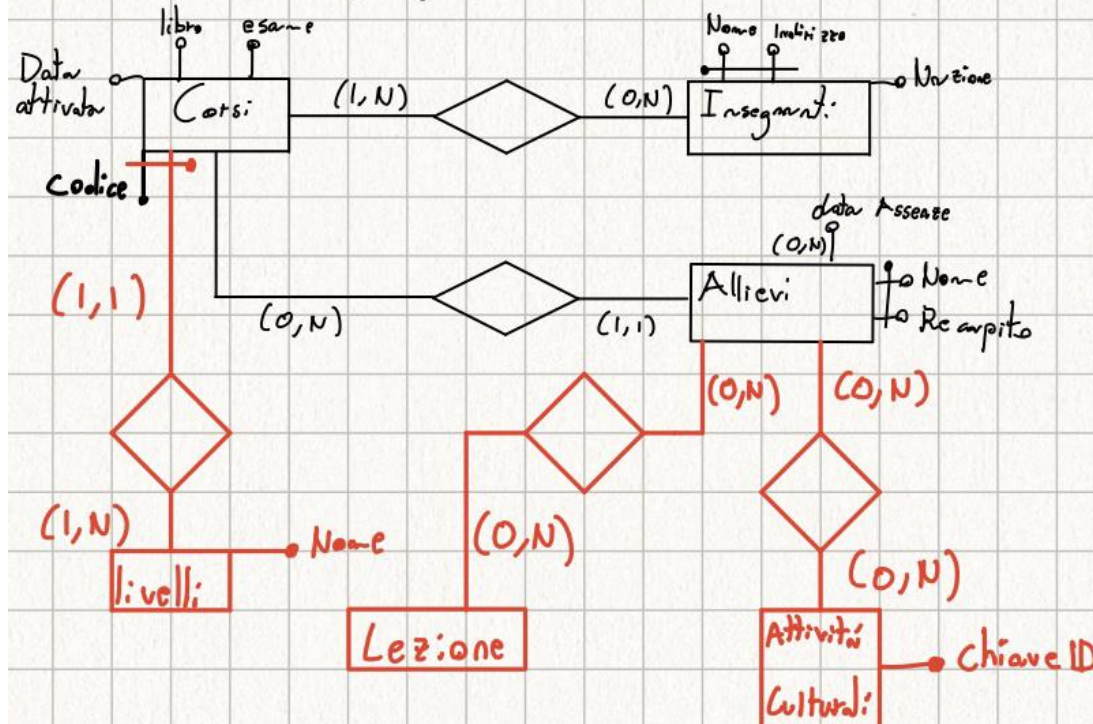
##### Passo 2





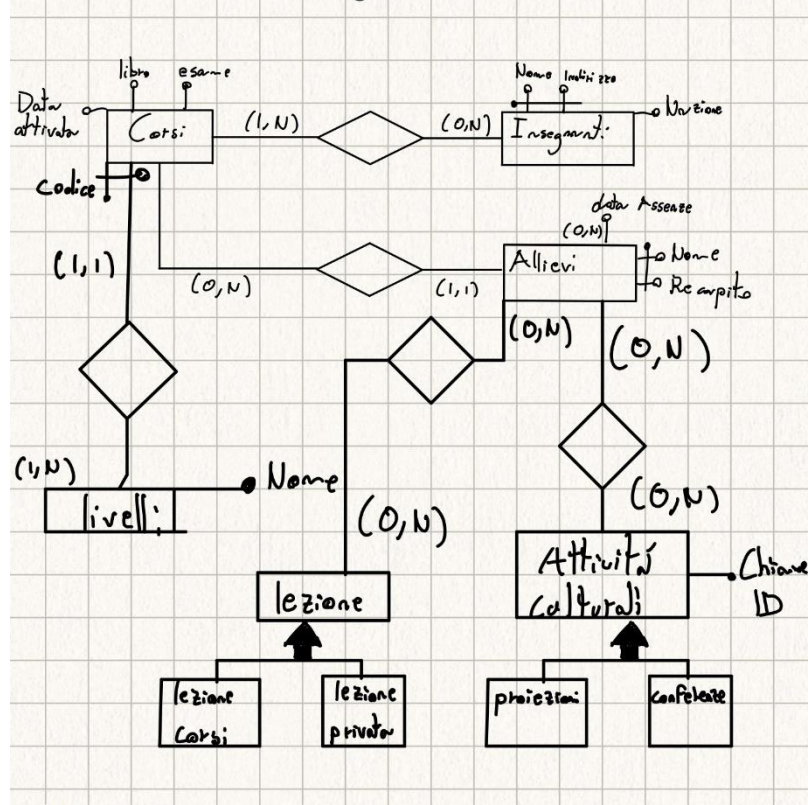
## Passo 3

## Passo 3 Aggiungo Ulteriori Entità



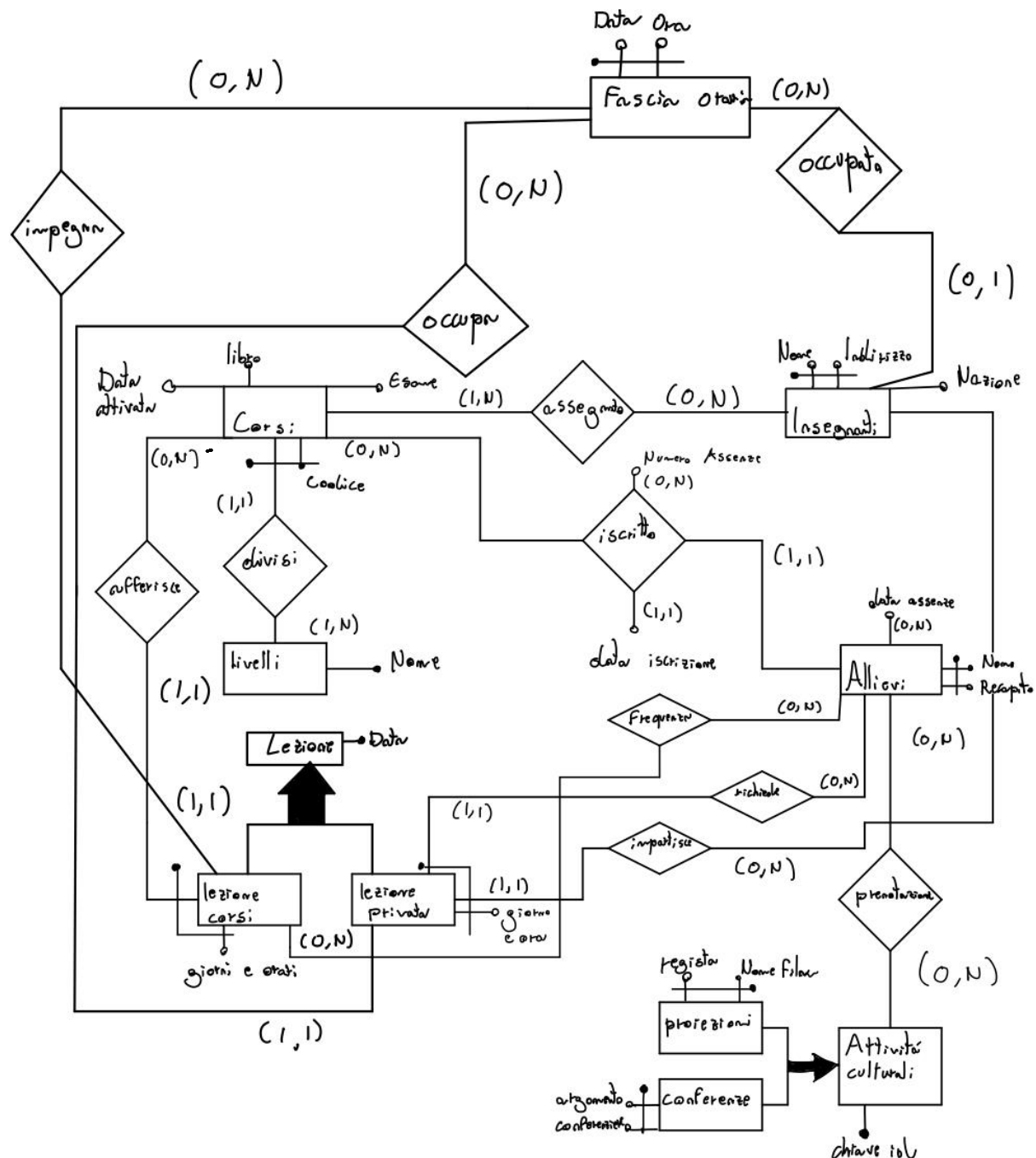
## Passo 4

## Passo 4: effettuo le generalizzazioni





## Integrazione finale



## Regole aziendali

- Assicurarsi che in una determinata fascia oraria un insegnante sia assegnato ad un solo corso.
- La prenotazione di una lezione individuale non può avvenire in concomitanza di un altro impegno di un insegnante.

## Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
<b>Corsi</b>	I corsi a cui afferiscono gli student e gli insegnanti assegnati	Libro, Data attivazione, esame, codice	Codice/Nome <b>Livello</b>
<b>Insegnanti</b>	Coloro che tengono I corsi nell'istituto	Nome, Indirizzo, Nazione	Nome/Indirizzo
<b>Allievi</b>	Coloro che seguono I corsi nell'istituto di insegnamento	Data assenze, Nome, Recapito	Nome/Recapito
<b>Livelli</b>	I vari livelli in cui sono suddivisi I corsi	Nome	Nome
<b>Attività culturali</b>	Si riferisce alle varie attività organizzate dall'istituto al di fuori dei corsi	Chiave Id	Chiave Id
<b>Lezione</b>	Si riferisce alle singole lezioni a cui afferiscono gli allievi	Data	Data
<b>Proiezioni</b>	Proiezioni di film in lingua inglese	Regista, nome film	Chiave ID <b>Attività Culturali</b>
<b>Conferenze</b>	Tipo di attività culturale in cui gli allievi seguono un "corso" esterno tenuto da un non insegnante della scuola	Argomento, conferenziere	Chiave ID <b>Attività Culturali</b>
<b>Lezione corsi</b>	Le lezioni relative ai corsi	Giorni e orari	Giorni e Orari/ Data <b>Lezioni</b>
<b>Lezioni private</b>	Le lezioni relative alle lezioni individuali con I Prof	Giorni e orari	Giorni e Orari/ Data <b>Lezioni</b>
<b>Fascia Oraria</b>	Entità che tiene conto delle fasce orarie occupate per ogni insegnante dalle lezioni per I corsi e quelle private	Data, Ora	Data, Ora

## 4. Progettazione logica

### Volume dei dati

Dati a Regime:

Concetto nello schema	Tipo <sup>1</sup>	Volume atteso
Corsi	E	15
Insegnanti	E	20
Livelli	E	5
Allievi	E	250
Lezione	E	1160
Lezione Corsi	E	960
Lezione Privata	E	200
Attività Culturali	E	30
Proiezioni	E	15
Conferenze	E	15
Fascia Oraria	E	1160
Assegnato	R	40
Iscritto	R	250
Divisi	R	15
Afferisce	R	960
Frequenza	R	16000
Richiede	R	200
Impartisce	R	200
Prenotazione	R	3000
Impegna	R	960
Occupa	R	200
Occupato	R	960

---

<sup>1</sup> Indicare con E le entità, con R le relazioni

Il volume delle lezioni è stato calcolato tenendo conto di almeno 2 ore settimanali per ogni corso, considerando che ogni corso ha la durata di almeno 8 mesi (32 settimane ) =  $2 \cdot 42 \cdot 15$

### Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
Op.1	Assegnazione Insegnanti ad un Corso	20/Anno
Op.2	Insegnante assegnato ad una lezione Privata	20/mese
Op.3	Iscrizione Allievi ai corsi	250/Anno
Op.4	Prenotazione di una lezione privata da parte di un Allievo	20/mese
Op.5	Inserimento degli allievi presenti ad ogni Lezione	500/settimana
Op.6	Prenotazione da parte di un Allievo ad un'attività culturale	100/settimana
Op.7	Creazione Corsi in base ai Livelli	15/Anno
Op.8	Inserimento studente assente ad una Lezione	30/Giorno
Op.9	Controllo assenze studenti	1/settimana
Op.10	Generazione Report settimanali agenda Insegnanti	20/settimana
Op.11	Generazione report mensile attività Insegnanti	20/mese
Op.12	Creazione attività culturale	1/settimana
Op.13	Aggiungere Allievo	250/Anno
Op.14	Aggiungere Insegnante	2-3/Anno
Op.15	Aggiungere Corso	15/Anno
Op.16	Eliminare Allievo	250/Anno
Op.17	Eliminare Insegnante	2-3/Anno
Op.18	Eliminare Corso	15/Anno
Op.19	Eliminare Attività Culturale	1/settimana

## Costo delle operazioni

Operazione 1 → costo 6			
Concetto	Costrutto	Accessi	Tipo
Insegnante	E	1	L
Corso	E	1	L
Assegnato	R	1	S
Fascia Oraria	E	1	S

Operazione 2 → costo 6			
Concetto	Costrutto	Accessi	Tipo
Insegnante	E	1	L
Lezione Privata	E	1	S
Allievi	E	1	L
Fascia Oraria	R	1	S

Operazione 3 → costo 765			
Concetto	Costrutto	Accessi	Tipo
Allievi	E	250	L
Corsi	E	15	L
Iscritto	R	250	S

Operazione 4 → costo 3			
Concetto	Costrutto	Accessi	Tipo
Allievi	E	1	L
Lezione Privata	E	1	S

Operazione 5 → costo 1030			
Concetto	Costrutto	Accessi	Tipo
Allievi	E	500	L
Lezione Corsi	E	30	L
Frequenza	R	250	S

Operazione 6 → costo 4			
Concetto	Costrutto	Accessi	Tipo
Allievi	E	1	L
Attività	E	1	L
Prenotazioni	R	1	S

Operazione 7 → costo 45			
Concetto	Costrutto	Accessi	Tipo
Corsi	E	15	S
Livelli	E	15	L

Operazione 8 → costo 4			
Concetto	Costrutto	Accessi	Tipo
Allievi	E	1	S
Iscritto	R	1	S

Operazione 9→ costo 500			
Concetto	Costrutto	Accessi	Tipo
Allievi	E	250	L
Iscritto	R	250	L

Operazione 10 → costo 60			
Concetto	Costrutto	Accessi	Tipo
Insegnanti	E	20	L
Assegnato	R	20	L
Lezione Privata	E	20	L

Operazione 11→ costo 60			
Concetto	Costrutto	Accessi	Tipo
Insegnanti	E	20	L
Assegnato	R	20	L
Lezione Privata	E	20	L

Operazione 12→ costo 2			
Concetto	Costrutto	Accessi	Tipo
Attività Culturale	E	1	S

Operazione 13→ costo 500			
Concetto	Costrutto	Accessi	Tipo
Allievi	E	250	S



Operazione 14→ costo 40			
Concetto	Costrutto	Accessi	Tipo
Insegnanti	E	20	S

Operazione 15→ costo 30			
Concetto	Costrutto	Accessi	Tipo
Corsi	E	15	S

Operazione 16→ costo 500			
Concetto	Costrutto	Accessi	Tipo
Attività Culturale	E	250	S

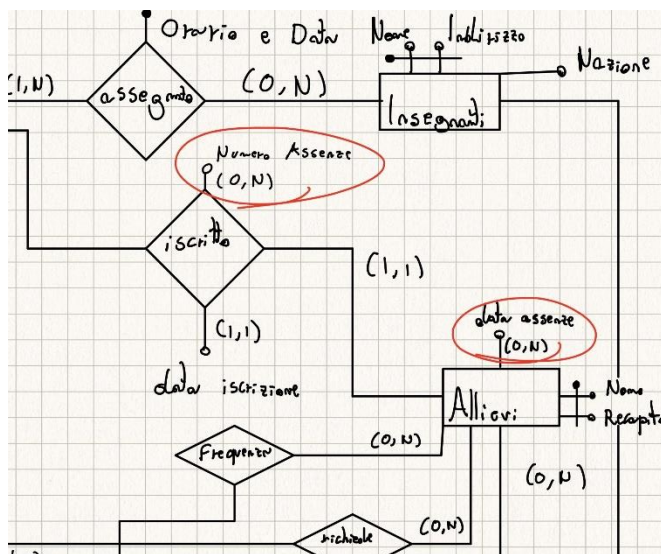
Operazione 17→ costo 40			
Concetto	Costrutto	Accessi	Tipo
Insegnante	E	20	S

Operazione 18→ costo 30			
Concetto	Costrutto	Accessi	Tipo
Corsi	E	15	S

Operazione 19→ costo 2			
Concetto	Costrutto	Accessi	Tipo
Attività Culturale	E	1	S

## Ristrutturazione dello schema E-R

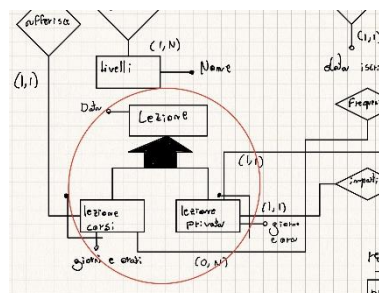
### Analisi Delle Ridondanze:



l'attributo chiave "Giorni e orari" per l'entità **Lezione corsi**.

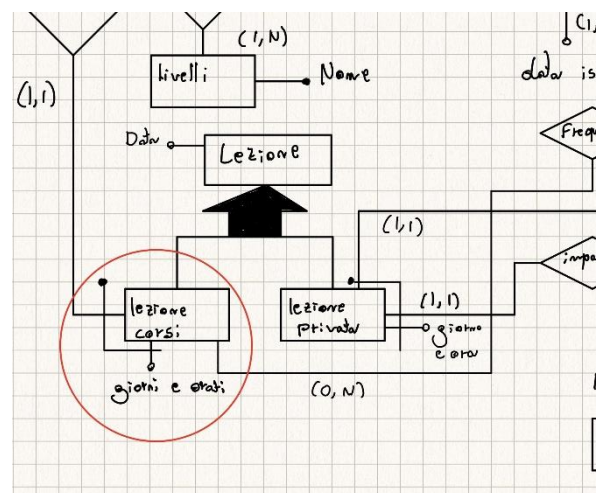
Come primi dati ridondanti abbiamo la presenza di ben due attributi derivabili da operazioni di conteggio di occorrenze. Gli attributi "Numero Assenze" relativo all'associazione **Iscritto** e "Data Assenze" relativo all'entità **Allievi** sono infatti ridondanti dal momento che con delle operazioni di conteggio tramite l'associazione "**Frequenza**" si può risalire non solo al numero di assenze effettuate da un allievo per un determinato corso ma anche alle date essendoci

### Eliminazione delle generalizzazioni:

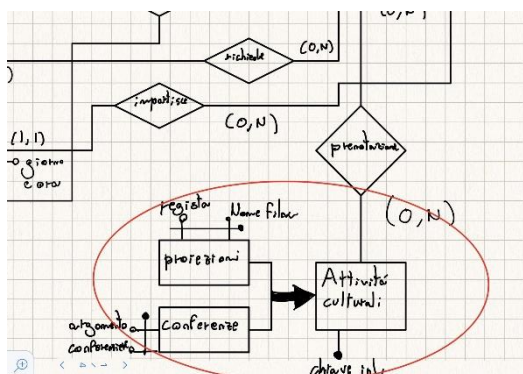


Nel caso delle generalizzazioni ci troviamo di fronte a due generalizzazioni totali da dover eliminare.

La prima di esse è la generalizzazione dell'entità **Lezione** in due entità figlie "Lezione corsi" e "Lezione privata", in questo specifico caso essendo l'entità **Lezione** del tutto superflua dal momento che anche il suo attributo chiave "Data" è già presente nelle entità figlie sotto il nome di "Giorni e orari" quindi non c'è neanche bisogno di "tramandare" gli attributi del padre ai figli.



La seconda generalizzazione di cui ci vogliamo liberare è quella dell'entità “**Attività culturali**” che dà vita alle due entità “**Proiezioni**” e “**Conferenze**”. In questo caso invece è meglio, per semplicità e



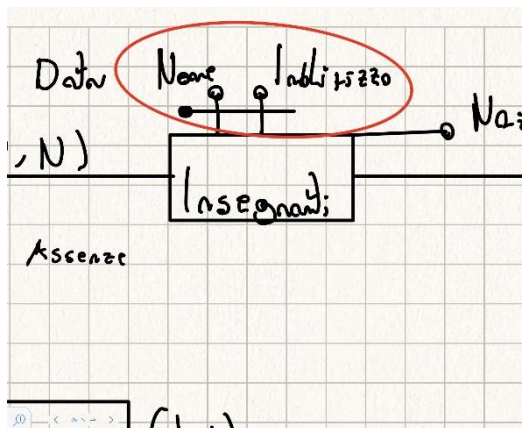
compattezza, ricorrere ad un accorpamento delle entità figlie al genitore e, grazie all'introduzione di un attributo ulteriore oltre quelli che acquisisce il genitore dai figli, possiamo distinguere quando in presenza di determinate occorrenze ci si riferisce a “**Proiezioni**” o “**Conferenze**”.

L'attributo in questione è “**Tipologia**”.

### Scelta degli identificatori principali

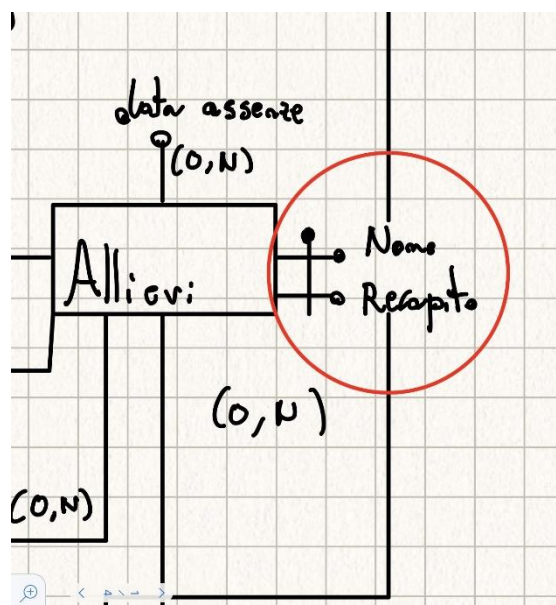
Sotto questo punto di vista, nel caso di questa Base di Dati ci troviamo di fronte a tre identificatori con chiave esterna purtroppo due di questi tre non possono essere ristrutturati, dal momento che questa scelta è stata compiuta per evitare ambiguità al livello di identificazione dal momento che alcune **Date** e **orari** potrebbero coincidere tra i vari corsi.

Possiamo però cambiare l'identificatore esterno dell'entità “**Corsi**” dal momento che basta il codice di riferimento al corso e non il nome del livello corrispondente.



Abbiamo invece due casi in cui l'identificatore è interno all'entità ma è composto da due attributi diversi. È il caso dell'identificatore di “**Insegnanti**” (Nome e Indirizzo) e di “**Allievi**” (Nome e Recapito). Dal

momento che data la presenza di 250 Allievi e di soli 20 professori è molto più probabile che esistano due Allievi con lo stesso nome è più congeniale mantenere questa come chiave composta da due attributi piuttosto che “**Insegnanti**”, per questo motivo la nuova chiave di **Insegnanti** sarà solo il **Nome**.



**Traduzione di entità e associazioni**

Corsi (Codice, data\_attivazione, libro, esame)

Insegnanti (Nome, Indirizzo, Nazione)

Livelli (Nome)

Allievi (Nome, Recapito, data\_assenze)

Lezione Corsi (Giorni, Orario, Corsi)

Lezione Privata (Giorni, Orario, Insegnanti, NomeAllievo, RecapitoAllievo)

Attività Culturali (Chiave\_ID, Tipologia, argomento, conferenziere, Regista, Nome Film)

Divisi (Corsi, Nome)

Iscritto (NomeAllievo, RecapitoAllievo, Corsi, data\_iscrizione, numero\_assenze)

Frequenza (NomeAllievo, RecapitoAllievo, GiornoLezCorsi, OraLezCorsi, Corsi)

Prenotazione (NomeAllievo, Recapito, Attività culturali)

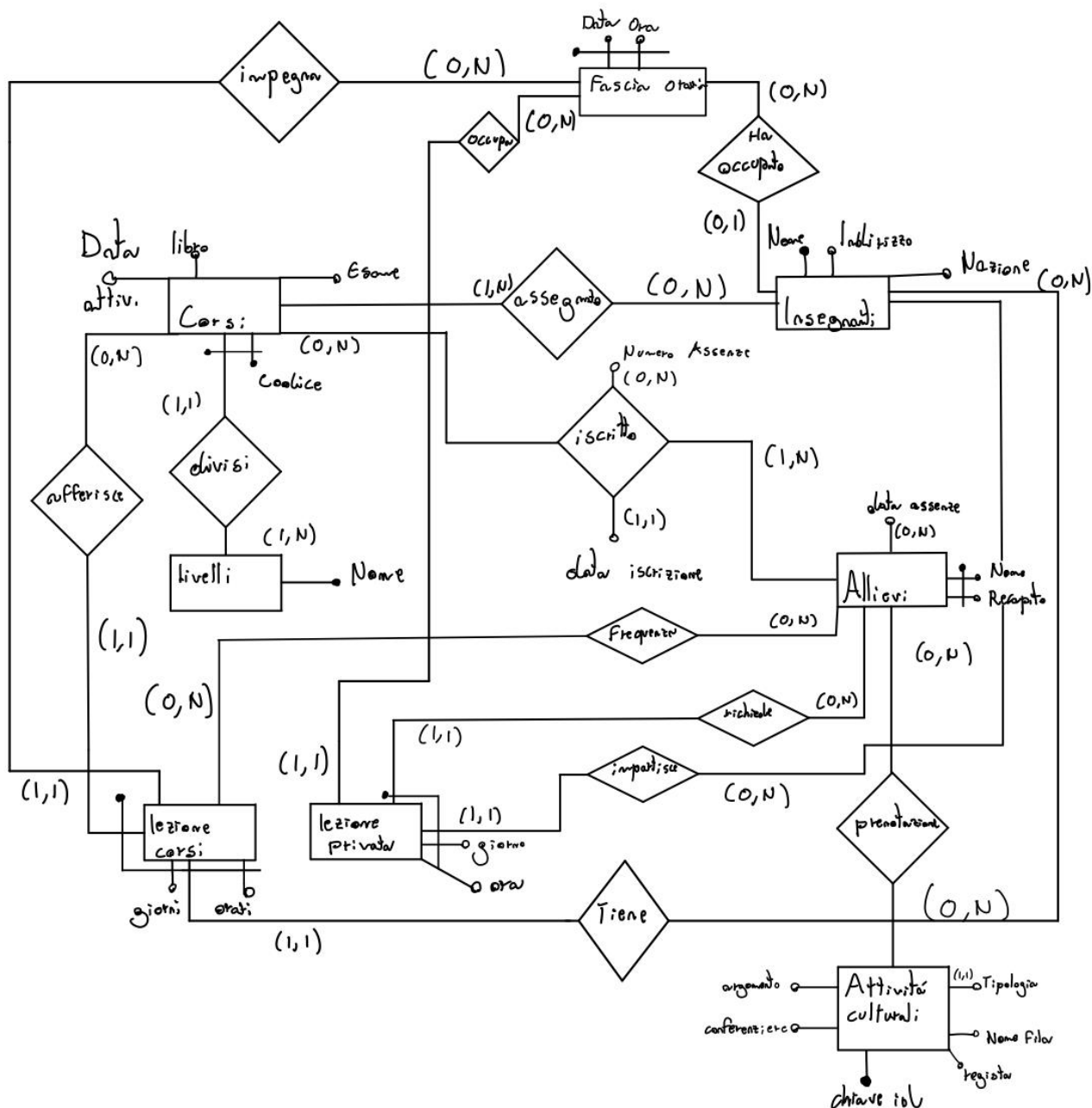
Assegnato (Insegnanti, Corsi, Orario, Data)

Fascia Oraria (Data, Ora)

Impegno (GiorniLezioniCorsi, OrarioLezioniCorsi, DataFasciaOraria, OraFasciaOraria)

Occupa (GiornoLezioniPrivate, OrarioLezioniPrivate, DataFasciaOraria, OraFasciaOraria)

Ha occupato (NomeInsegnanti, DataFasciaOraria, OraFasciaOraria)



## Normalizzazione del modello relazionale

La base di dati è in 3NF

## 5. Progettazione fisica

### Utenti e privilegi

#### Utenti:

- Personale amministrativo
- Personale di segreteria
- Insegnanti
- Allievi

#### Ruoli:

- Personale amministrativo
- Personale di segreteria
- Insegnanti
- Allievi

#### Privilegi:

##### Amministrazione:

- Inserimento informazioni legate ai corsi
- Inserimento informazioni legate ad Insegnanti
- Generare report mensili attività insegnanti
- Gestione assunzioni e contratti Insegnanti

##### Segreteria:

- Gestione iscrizioni ai corsi da parte degli allievi
- Gestione iscrizione allievi alla scuola e inserimento informazioni
- Organizzazione attività culturali settimanali
- Organizzazione esami finali
- Creazione calendario lezioni con orari e divisione fasce orarie Insegnanti
- Creare lezioni private
- Gestione prenotazioni attività culturali

##### Insegnanti:

- Generare Report riguardo la propria agenda su base settimanale

- Inserimento Allievi assenti ad una lezione

Allievi: gli allievi possono effettuare una forma di accesso di sola “lettura”, ovvero non hanno nessun tipo di privilegio, possono controllare il calendario dei corsi, delle lezioni private e tutte le info relative a corsi e ad insegnanti; possono inoltre informarsi sulle attività culturali e in caso prenotarsi.

- Prenotazione allievo ad un attività culturale

## Strutture di memorizzazione

Tabella <Corsi>		
Attributo	Tipo di dato	Attributi <sup>2</sup>
Codice	INT	PK NN
Livelli_Nome	VARCHAR(10)	PK NN
Data_attivazione	DATE	NN
Libro	VARCHAR(50)	NN
Esame	VARCHAR(16)	NN

Tabella <Insegnanti>		
Attributo	Tipo di dato	Attributi <sup>3</sup>
Nome	VARCHAR(20)	PK NN
User_Username	VARCHAR(45)	PK NN
Indirizzo	VARCHAR(45)	NN
Nazione	VARCHAR(20)	NN

---

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

<sup>3</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.



Tabella <Fascia Oraria >		
Attributo	Tipo di dato	Attributi <sup>4</sup>
<b>Data</b>	DATE	PK NN
<b>Ora</b>	TIME	PK NN

Tabella <Livelli>		
Attributo	Tipo di dato	Attributi <sup>5</sup>
<b>Nome</b>	VARCHAR(10)	PK NN

Tabella <Allievi>		
Attributo	Tipo di dato	Attributi <sup>6</sup>
<b>Nome</b>	VARCHAR(20)	PK NN
<b>Recapito</b>	VARCHAR(45)	PK NN
<b>User_Username</b>	VARCHAR(45)	PK NN
<b>Data_assenze</b>	DATE	

---

<sup>4</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

<sup>5</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

<sup>6</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

Tabella <Lezione Corsi>		
Attributo	Tipo di dato	Attributi <sup>7</sup>
<b>Giorni</b>	DATE	PK NN
<b>Orario</b>	TIME	PK NN
<b>Insegnanti_Nome</b>	VARCHAR(20)	PK NN
<b>Corsi_Codice</b>	INT	PK NN
<b>Corsi_Livelli_Nome</b>	VARCHAR(10)	PK NN
<b>FasciaOraria_Data</b>	DATE	NN
<b>FasciaOraria_Ora</b>	TIME	NN
<b>Lezione_Corsi_Insegnanti_underscorename</b>	VARCHAR(45)	PK NN

Tabella <Lezione Privata>		
Attributo	Tipo di dato	Attributi <sup>8</sup>
<b>Giorni</b>	DATE	PK NN
<b>Orario</b>	TIME	PK NN
<b>FasciaOraria_Data</b>	DATE	NN
<b>FasciaOraria_Ora</b>	TIME	NN
<b>Insegnanti</b>	VARCHAR(20)	PK NN
<b>NomeAllievo</b>	VARCHAR(20)	PK NN
<b>RecapitoAllievo</b>	VARCHAR(45)	PK NN

---

<sup>7</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

<sup>8</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

Tabella <Attività culturali>		
Attributo	Tipo di dato	Attributi <sup>9</sup>
<b>Chiave_ID</b>	INT	PK NN
<b>Tipologia</b>	VARCHAR(10)	NN
<b>Conferenziere</b>	VARCHAR(20)	
<b>Argomento</b>	VARCHAR(45)	
<b>Regista</b>	VARCHAR(20)	
<b>NomeFilm</b>	VARCHAR(20)	

Tabella <Iscritto>		
Attributo	Tipo di dato	Attributi <sup>10</sup>
<b>NomeAllievo</b>	VARCHAR(20)	PK NN
<b>RecapitoAllievo</b>	VARCHAR(45)	PK NN
<b>Corsi_Codice</b>	INT	PK NN
<b>Corsi_Livelli_Nome</b>	VARCHAR(10)	PK NN
<b>Data_iscrizione</b>	DATE	NN
<b>Numero_assenze</b>	INT	

---

<sup>9</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

<sup>10</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

Tabella <Prenotazione>		
Attributo	Tipo di dato	Attributi <sup>11</sup>
NomeAllievo	VARCHAR(20)	PK NN
RecapitoAllievo	VARCHAR(45)	PK NN
AttivitàCulturale	INT	PK NN

Tabella <Corsi_has_Insegnanti>		
Attributo	Tipo di dato	Attributi <sup>12</sup>
Insegnanti	VARCHAR(20)	PK NN
Corsi_Codice	INT	PK NN
Corsi_Livelli_Nome	VARCHAR(10)	PK NN

---

<sup>11</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

<sup>12</sup> PK = primary key, NN = not null, UQ = unique, UN = unsigned, AI = auto increment. È ovviamente possibile specificare più di un attributo per ciascuna colonna.

## Indici

Tabella <Corsi>	
Indice	Tipo <sup>13</sup> :
Codice	PRIMARY
Livelli_Nome	PRIMARY
Fk_Corsi_Livello_INDEX	INDEX

Tabella <Insegnanti>	
Indice	Tipo <sup>14</sup> :
Nome	PRIMARY
User_Username	PRIMARY
Fk_Insegnanti_User1_idx	INDEX

Tabella <Livelli>	
Indice	Tipo <sup>15</sup> :
Nome	PRIMARY

Tabella <Allievi>	
Indice	Tipo <sup>16</sup> :
Nome	PRIMARY
Recapito	PRIMARY
User_Username	PRIMARY
fk_Allievi_User1_idx	INDEX

---

<sup>13</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

<sup>14</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

<sup>15</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

<sup>16</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella <Corsi_has_Insegnanti>	
Indice	Tipo <sup>17</sup> :
Corsi_Codice	PRIMARY
Corsi_Livelli_Nome	PRIMARY
Insegnanti_Nome	PRIMARY
fk_Corsi_has_Insegnanti1_idx	INDEX
Fk_Corsi_has_Insegnanti_Corsi_idx	INDEX

Tabella <Iscritto>	
Indice	Tipo <sup>18</sup> :
Corsi_Codice	PRIMARY
Corsi_Livelli	PRIMARY
Allievi_Nome	PRIMARY
Allievi_Recapito	PRIMARY
fk_Corsi_has_Allievi_Allievi1_idx	INDEX
Fk_Corsi_has_Allievi_Corsi1_idx	INDEX

Tabella <Prenotazione>
------------------------

<sup>17</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

<sup>18</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

Indice	Tipo <sup>19</sup> :
Allievi_Nome	PRIMARY
Allievi_Recapito	PRIMARY
Attività Culturali_Chiave_ID	PRIMARY
fk_Allievi_has_ActivitàCulturali_ActivitàCulturali1_idx	INDEX
Fk_Allievi_has_ActivitàCulturali_Allievi1_idx	INDEX

Tabella <Fascia Oraria>	
Indice	Tipo <sup>20</sup> :
Data	PRIMARY
Ora	PRIMARY

Tabella <Lezione Corsi>	
Indice	Tipo <sup>21</sup> :
Giorni	PRIMARY
Orario	PRIMARY
Corsi_Codice	PRIMARY
Insegnanti_Nome	PRIMARY
Corsi_Livello	PRIMARY
Lezione_Corsi_Insegnanti_username	PRIMARY
fk_LezioneCorsi_Codice_idx	INDEX
fk_LezioneCorsi_FasciaOraria1_idx	INDEX
fk_LezioneCorsi_Insegnanti1_idx	INDEX

---

<sup>19</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

<sup>20</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

<sup>21</sup> IDX = index, UQ = unique, FT = full text, PR = primary.



Tabella <Attività Culturali>	
Indice	Tipo <sup>22</sup> :
Chiave_ID	PRIMARY

Tabella <Frequenza>	
Indice	Tipo <sup>23</sup> :
Allievi_Nome	PRIMARY
Allievi_Recapito	PRIMARY
LezioneCorsi_Giorni	PRIMARY
LezioneCorsi_Orario	PRIMARY
LezioneCorsi_Insegnanti_Nome	PRIMARY
LezioneCorsi_Cors_Codice	PRIMARY
LezioneCorsi_Corsi_Livelli_Nome	PRIMARY
fk_Allievi_has_LezionePrivata_Lezione_Corsi1_idx	INDEX
fk_Allievi_has_Lezione_Corsi_Allievi1_idx	INDEX

---

<sup>22</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

<sup>23</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella <Lezione Privata>	
Indice	Tipo <sup>24</sup> :
Giorni	PRIMARY
Orario	PRIMARY
Allievi_Nome	PRIMARY
Allievi_Recapito	PRIMARY
Insegnanti_Nome	PRIMARY
fk_LezionePrivata_Allievi1_idx	INDEX
fk_LezionePrivata_insegnanti1_idx	INDEX
fk_LezionePrivata_FasciaOraria1_idx	INDEX

---

<sup>24</sup> IDX = index, UQ = unique, FT = full text, PR = primary.

## Trigger

Non ci sono trigger specifici

## Stored Procedures e transazioni

### Assegnazione Insegnante a corso:

```
CREATE PROCEDURE `Assegnazione_insegnanti_a_corso` (In var_Corsi int, in var_livello  
VARCHAR(10), in var_Insegnanti varchar(20))
```

```
BEGIN
```

```
    declare exit handler for sqlexception
```

```
    BEGIN
```

```
        rollback;
```

```
    resignal;
```

```
END;
```

```
IF var_Insegnanti not in (Select Corsi_has_Insegnanti.Insegnanti_Nome
```

```
    FROM `Corsi_has_Insegnanti`)
```

```
    THEN
```

```
        Insert into `Corsi_has_Insegnanti` (`Corsi_Codice`,  
`Corsi_Livelli_Nome`, `Insegnanti_Nome`) values (var_Corsi, var_livello, var_Insegnanti);
```

```
    ELSE
```

```
        signal sqlstate '45002' set message_text = 'Insegnante assegnato già ad un corso';
```

```
    end if;
```

```
END
```

**Assegnazione Insegnante a Lezione Privata:**

declare exit handler for sqlexception

```
BEGIN

    rollback;

    resignal;

END;

IF var_insegnante not in ( Select HaOccupato.Insegnanti_Nome

FROM `HaOccupato`

WHERE HaOccupato.Insegnanti_Nome = var_insegnante and

HaOccupato.FasciaOraria_Data = var_giorno AND

HaOccupato.FasciaOraria_Ora = var_ora)

THEN

    insert into `Lezione_Privata` (`Giorni`, `Orario`, `FasciaOraria_Data`,

`FasciaOraria_Ora`, `Allievi_Nome`, `Allievi_Recapito`, `Insegnanti_Nome`) values (var_giorno,

var_ora, var_giorno, var_ora, var_allievoNome, var_allievoRecapito, var_Insegnante);

ELSE

    signal sqlstate '45002' set message_text = 'Insegnante non disponibile in questa fascia

oraria';

    end if;

END
```

**Controllo assenze studente:**

```
CREATE PROCEDURE `Controllo_assenze_studente` ( in var_nome VARCHAR(20), in
var_recapito VARCHAR(45), in var_giorno DATE, in var_corso INT, out var_allievo_assente
VARCHAR(20))

BEGIN

    declare exit handler for sqlexception

    BEGIN

        rollback;

    resignal;

END;

start transaction;

IF var_giorno in (Select Lezione_Corsi.Giorni

FROM `Lezione_Corsi`

WHERE

Lezione_Corsi.Giorni = var_giorno AND Lezione_Corsi.Corsi_Codice = var_corso)

THEN

    IF var_nome not in (Select Lezione_Corsi.Allievi_Nome

FROM `Frequenza`

WHERE

Lezione_Corsi.Allievi_Nome = var_nome AND
Lezione_Corsi.Allievi_Recapito = var_recapito and Frequenza.Lezione_Corsi_Giorni = var_giorno
and Frequenza.Lezione_Corsi_Corsi_Codice = var_corso)

THEN
```

```
        set var_allievo_assente = var_nome;

    ELSE

        signal sqlstate '45002' set message_text = 'Allievo presente alla lezione';

        end if;

    ELSE

        signal sqlstate '45002' set message_text = 'Non esiste una lezione di
questo corso nel giorno selezionato';

        end if;

        commit;

END
```

**Creazione Attività culturale:**

```
CREATE PROCEDURE `Creazione_attivita_culturale` (in var_chiave INT, in var_Tipologia
VARCHAR(10), in var_conferenzieri VARCHAR(20), in var_argomento VARCHAR(45), in
var_regista VARCHAR(20), in var_NomeFilm VARCHAR(20))
```

```
BEGIN
```

```
    declare exit handler for sqlexception
```

```
BEGIN
```

```
    rollback;
```

```
    resignal;
```

```
END;
```

```
    IF var_Tipologia = 'Conferenza' then
```

```
        insert into `Attività_Culturali` (`Chiave_ID`, `Tipologia`, `Conferenzieri`, `Argomento`) values
(var_chiave, var_Tipologia, var_conferenzieri, var_argomento);
```

```
    elseif
```

```
        var_Tipologia = 'Film'
```

```
then insert into `Attività_Culturali` (`Chiave_ID`, `Tipologia`,`Regista`, `NomeFilm`) values
(var_chiave, var_Tipologia, var_regista, var_NomeFilm);

else

signal sqlstate '45002' set message_text = 'Attività non supportata';

end if;

END
```

**Creazione corso:**

```
CREATE PROCEDURE `Creazione_corso` ( in var_corso INT, in var_data DATE, in var_libro
VARCHAR(50), in var_esame VARCHAR(16), in varlivello VARCHAR(45))

BEGIN

declare exit handler for sqlexception

BEGIN

rollback;

resignal;

END;

IF var_corso not in(Select Corsi.Codice

FROM `Corsi`

WHERE

Corsi.Codice = var_corso)

THEN

Insert into `Corsi` ( `Codice`, `Data_attivazione`, `Libro`, `Esame`, `Livelli_nome`) values
(var_corso, var_data, var_libro, var_esame, varlivello);

ELSE

signal sqlstate '45002' set message_text = 'Corso già presente';
```



```
end if;
```

```
END
```

### **Generare report Insegnante Mensile:**

```
CREATE PROCEDURE `Generare_report_Insegnante_Mensile` (in var_insegnante VARCHAR(20))
```

```
BEGIN
```

```
    drop temporary table if exists `Report Mensile Insegnante`;
```

```
    create temporary table `Report Mensile Insegnate`(`Insegnante` VARCHAR(20),`Nome livello  
corso`VARCHAR(20),`Lezione Corso Giorno` DATE,`Lezione Corso Ora` TIME, `Lezione Privata  
Giorno` DATE, `Lezione Privata Ora` TIME);
```

```
    insert into `Report Mensile Insegnante`
```

```
        SELECT    Insegnante.Nome,    Lezione_Corsi.Corsi_Livelli_Nome,    Lezione_Corsi.Giorni,  
        Lezione_Corsi.Orario, Lezione_Privata.Giorni, Lezione_Privata.Orario
```

```
        From `Insegnante`
```

```
        JOIN Lezione_Corsi on (Insegnante.Nome = Lezione_Corsi.Insegnanti_Nome) JOIN  
        Lezione_Privata on (Insegnante.Nome = Lezione_Privata.Insegnanti.Nome)
```

```
        WHERE Insegnante.Nome = var_insegnante and Lezione_Corsi.Giorni < var_giorno and  
        Lezione_Corsi.Giorni > var_giorno - 31 and Lezione_Privata.Giorni < var_giorno and  
        Lezione_Privata.Giorni > var_giorno - 31;
```

```
    drop temporary table `Report Mensile Insegnante`;
```

```
    commit;
```

```
END
```

### **Inserimento degli allievi presenti ad ogni Lezione:**

```
CREATE PROCEDURE `Inserimento_degli_allievi_presenti_ad_ogni_Lezione` (in var_nome  
VARCHAR(20), in var_Recapito VARCHAR(45), in var_giorno DATE, in var_ora TIME,in  
var_corso INT, in var_Insegnante VARCHAR(20), IN var_Livello varchar(10))
```

```
BEGIN
```

```
insert into Frequenza values (var_giorno, var_ora, var_nome, var_Recapito, var_corso,  
var_Insegnante, var_Livello);
```

```
END
```

### **Iscrizione Allievo a corso:**

```
CREATE PROCEDURE `iscrizione_Allievo_a_corso` (IN var_nome VARCHAR(20), IN  
var_recapito VARCHAR(45), in var_corso int, in var_livello VARCHAR(20), in var_data DATE)
```

```
BEGIN
```

```
declare exit handler for sqlexception
```

```
BEGIN
```

```
rollback;
```

```
resignal;
```

```
END;
```

```
IF var_nome not in(Select Iscritto.Allievi_Nome
```

```
FROM `Iscritto`
```

```
WHERE
```

```
Iscritto.Allievi_Nome = var_nome AND Iscritto.Allievi_Recapito =  
var_recapito)
```

```
THEN
```

```
INSERT into `Iscritto` ( `Allievi_Nome`, `Allievi_Recapito`, `Corsi_Codice`,  
`Corsi_Livelli_Nome`, `Data_Iscrizione` ) values (var_nome, var_recapito,var_corso, var_livello,  
var_data);
```

```
else signal sqlstate '45002' set message_text = 'allievo già presente in un altro corso';
```

```
end if;
```

```
END
```

**Login:**

```
CREATE PROCEDURE `Login` (in var_username varchar(45), in var_password varchar(45), out var_role INT)
```

```
BEGIN
```

```
    declare var_user_role ENUM('Amministrazione', 'Segreteria', 'Insegnante', 'Allievo');
```

```
    select `Ruolo`
```

```
        from `User`
```

```
        where `Username` = var_username
```

```
        and `Password` = md5(var_password)
```

```
        into var_user_role;
```

```
    if var_user_role = 'Amministrazione' then
```

```
        set var_role = 1;
```

```
    elseif var_user_role = 'Segreteria' then
```

```
        set var_role = 2;
```

```
    elseif var_user_role = 'Insegnante' then
```

```
        set var_role = 3;
```

```
    elseif var_user_role = 'Allievo' then
```

```
        set var_role = 4;
```

```
    else set var_role = 5;
```

```
    end if;
```

```
END
```

**Aggiunta Allievo:**

```
CREATE PROCEDURE `Aggiunta_Allievo` (in var_Nome VARCHAR(20), in var_Recapito
VARCHAR(45), in var_Username VARCHAR(45))

BEGIN

    declare exit handler for sqlexception

    BEGIN

        rollback;

    resignal;

    END;

    if var_Username not in (select `User_Username` from `Allievi` where var_Username =
`User_Username`) then

        INSERT into `Allievi` (`Nome`, `Recapito`,`User_Username`) values (var_Nome, var_Recapito,
var_Username);

    else

        signal sqlstate '45002' set message_text = 'Allievo presente o username usato per un
altro allievo';

    end if;

    commit;

END
```

**Aggiunta Insegnante:**

```
CREATE PROCEDURE `Aggiunta_Insegnante` (in var_Insegnante VARCHAR(20), in
var_username VARCHAR(20), in var_indirizzo VARCHAR(45), in var_nazione VARCHAR(20))

BEGIN

    INSERT into `Insegnanti` (`Nome`, `User_Username`,`Indirizzo`, `Nazione` ) values
(var_Insegnante, var_username, var_indirizzo, var_nazione);

END
```

**Inserisci Utente:**

```
CREATE PROCEDURE `Inserisci_Utente` (in var_username VARCHAR(20), in var_password
VARCHAR(20), in var_role VARCHAR(20))

BEGIN

    insert into User values(var_username, md5(var_password), var_role);

END
```

**Inserisci lezione:**

```
CREATE PROCEDURE `inserisci_lezione` (in var_giorno DATE, in var_ora TIME, in
var_insegnante VARCHAR(20), in var_corso INT, in var_livello VARCHAR(10), in var_username
VARCHAR(45))

BEGIN

    insert into FasciaOraria values (var_giorno, var_ora);

    #insert into HaOccupato values (var_insegnante, var_giorno, var_ora, var_username);

    insert into Lezione_Corsi values (var_giorno, var_ora, var_insegnante, var_corso, var_livello,
var_giorno, var_ora, var_username);

END
```

**Report agenda settimanale:**

```
CREATE PROCEDURE `Report_agenda_settimanale` (in var_Nome VARCHAR(20), in var_giorno DATE)
```

```
BEGIN
```

```
    drop temporary table if exists `Report Agenda`;
```

```
    create temporary table `Report Agenda`(`Insegnante` VARCHAR(20), `Nome livello corso` VARCHAR(20), `Lezione Corso Giorno` DATE, `Lezione Corso Ora` TIME, `Lezione Privata Giorno` DATE, `Lezione Privata Ora` TIME);
```

```
    insert into `Report Agenda`
```

```
        SELECT    Insegnante.Nome,    Lezione_Corsi.Corsi_Livelli_Nome,    Lezione_Corsi.Giorni,
        Lezione_Corsi.Orario, Lezione_Privata.Giorni, Lezione_Privata.Orario
```

```
        From `Insegnante`
```

```
        JOIN Lezione_Corsi on (Insegnante.Nome = Lezione_Corsi.Insegnanti_Nome) JOIN
        Lezione_Privata on (Insegnante.Nome = Lezione_Privata.Insegnanti.Nome)
```

```
        WHERE Insegnante.Nome = var_insegnante and Lezione_Corsi.Giorni = var_giorno + 7 and
        Lezione_Privata.Giorni = var_giorno + 7;
```

```
    drop temporary table `Report Agenda`;
```

```
END
```

**Prenotazione allievo ad un attivita culturale:**

```
CREATE    PROCEDURE    `prenotazione_allievo_ad_un_attivita_culturale`    (in    var_nome VARCHAR(20), in var_recapito VARCHAR(45), in var_attivita INT)
```

```
BEGIN
```

```
    #select `Allievi.Nome`, `Allievi.Recapito`, `Attività Culturali.Chiave_ID` from `Allievi` join
    `Attività_Culturali` on
```

```
        INSERT into Prenotazione values ( var_nome, var_recapito, var_attivita);
```

```
END
```

**Elimina Insegnante:**

```
CREATE PROCEDURE `Elimina_Insegnante` (in var_Nome VARCHAR(20))  
  
BEGIN  
  
    DELETE FROM `Insegnanti` Where Insegnanti.Nome = var_Nome;  
  
END
```

**Elimina Allievo:**

```
CREATE PROCEDURE `Elimina_Allievo` (in var_Nome VARCHAR(20), in var_Recapito  
VARCHAR(45))  
  
BEGIN  
  
    DELETE FROM `Allievi` Where Allievi.Nome = var_Nome && `Allievi.Recapito` =  
var_Recapito;  
  
END
```

**Elimina Tutti Allievi:**

```
CREATE PROCEDURE `Elimina_Tutti_Allievi` ()  
  
BEGIN  
  
    DELETE FROM `Allievi`;  
  
END
```

**Elimina Corso:**

```
CREATE PROCEDURE `Elimina_Corso` (in var_Corso INT)  
  
BEGIN  
  
    DELETE FROM `Corsi` Where Corsi.Codice = var_Corso;  
  
END
```

**Visualizza corsi assegnati a insegnanti:**

```
CREATE PROCEDURE `visualizza_corsi_assegnati_a_insegnanti` (in var_Corso INT, out  
var_insegnante VARCHAR(45))  
  
BEGIN  
  
    set transaction read only;  
  
    set transaction isolation level read committed;  
  
    select Corsi_has_Insegnanti.Insegnanti_Nome from `Corsi_has_Insegnanti` where var_Corso =  
Corsi_has_Insegnanti.Corsi_Codice;  
  
    set var_insegnante = Corsi_has_Insegnanti.Insegnanti_Nome;  
  
    commit;  
  
END
```

**Visualizza Insegnanti assegnati a corsi:**

```
CREATE PROCEDURE `visualizza_Insegnanti_assegnati_a_corsi` (in var_Nome VARCHAR(45),  
out var_corso INT)  
  
BEGIN  
  
    set transaction read only;  
  
    set transaction isolation level read committed;  
  
    select Corsi_has_Insegnanti.Corsi_Codice from `Corsi_has_Insegnanti` where var_Nome =  
Corsi_has_Insegnanti.Insegnanti_Nome;  
  
    set var_corso = Corsi_has_Insegnanti.Corsi_Codice;  
  
    commit;  
  
END
```



## 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 GestioneCorsiDiLingue

-----

DROP SCHEMA IF EXISTS `GestioneCorsiDiLingue` ;

-----

-- Schema GestioneCorsiDiLingue

-----

CREATE SCHEMA IF NOT EXISTS `GestioneCorsiDiLingue` DEFAULT CHARACTER SET utf8  
;

USE `GestioneCorsiDiLingue` ;

-----  
-- Table `GestioneCorsiDiLingue`.`Livelli`  
-----

DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Livelli` ;

CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Livelli` (

`Nome` VARCHAR(10) NOT NULL,

PRIMARY KEY (`Nome`))

ENGINE = InnoDB;

-----  
-- Table `GestioneCorsiDiLingue`.`Corsi`  
-----

DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Corsi` ;

CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Corsi` (

`Codice` INT NOT NULL,

`Data\_attivazione` DATE NOT NULL,

`Libro` VARCHAR(50) NOT NULL,

`Esame` VARCHAR(16) NOT NULL,

```
`Livelli_Nome` VARCHAR(10) NOT NULL,  
  
PRIMARY KEY (`Codice`, `Livelli_Nome`),  
  
CONSTRAINT `fk_Corsi_Livello`  
  
FOREIGN KEY (`Livelli_Nome`)  
  
REFERENCES `GestioneCorsiDiLingue`.`Livelli` (`Nome`)  
  
ON DELETE NO ACTION  
  
ON UPDATE NO ACTION)  
  
ENGINE = InnoDB;  
  
  
  
CREATE INDEX `fk_Corsi_Livello_idx` ON `GestioneCorsiDiLingue`.`Corsi` (`Livelli_Nome`  
ASC);  
  
  
-----  
  
-- Table `GestioneCorsiDiLingue`.`User`  
  
-----  
  
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`User` ;  
  
  
  
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`User` (  
  
`Username` VARCHAR(45) NOT NULL,  
  
`Password` VARCHAR(45) NOT NULL,  
  
`Ruolo` ENUM('Allievo', 'Insegnante', 'Amministrazione', 'Segreteria') NOT NULL,
```

PRIMARY KEY (`Username`))

ENGINE = InnoDB;

-----  
-- Table `GestioneCorsiDiLingue`.`Insegnanti`  
-----

DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Insegnanti` ;

CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Insegnanti` (

`Nome` VARCHAR(20) NOT NULL,

`Indirizzo` VARCHAR(45) NOT NULL,

`Nazione` VARCHAR(20) NOT NULL,

`User\_Username` VARCHAR(45) NOT NULL,

PRIMARY KEY (`Nome`, `User\_Username`),

CONSTRAINT `fk\_Insegnanti\_User1`

FOREIGN KEY (`User\_Username`)

REFERENCES `GestioneCorsiDiLingue`.`User` (`Username`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

```
CREATE INDEX `fk_Insegnanti_User1_idx` ON `GestioneCorsiDiLingue`.`Insegnanti`  
(`User_Username` ASC);
```

```
-----
```

```
-- Table `GestioneCorsiDiLingue`.`Allievi`
```

```
-----
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Allievi` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Allievi` (
```

```
  `Nome` VARCHAR(20) NOT NULL,
```

```
  `Recapito` VARCHAR(45) NOT NULL,
```

```
  `Data_assenze` DATE NULL,
```

```
  `User_Username` VARCHAR(45) NOT NULL,
```

```
  PRIMARY KEY (`Nome`, `Recapito`, `User_Username`),
```

```
  CONSTRAINT `fk_Allievi_User1`
```

```
    FOREIGN KEY (`User_Username`)
```

```
      REFERENCES `GestioneCorsiDiLingue`.`User` (`Username`)
```

```
    ON DELETE NO ACTION
```

```
    ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;
```

```
CREATE INDEX `fk_Allievi_User1_idx` ON `GestioneCorsiDiLingue`.`Allievi` (`User_Username`  
ASC) ;
```

```
-- -----
```

```
-- Table `GestioneCorsiDiLingue`.`FasciaOraria`
```

```
-- -----
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`FasciaOraria` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`FasciaOraria` (
```

```
  `Data` DATE NOT NULL,
```

```
  `Ora` TIME NOT NULL,
```

```
  PRIMARY KEY (`Data`, `Ora`))
```

```
ENGINE = InnoDB;
```

```
-- -----
```

```
-- Table `GestioneCorsiDiLingue`.`Lezione_Corsi`
```

```
-- -----
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Lezione_Corsi` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Lezione_Corsi` (  
  
  `Giorni` DATE NOT NULL,  
  
  `Orario` TIME NOT NULL,  
  
  `Insegnanti_Nome` VARCHAR(20) NOT NULL,  
  
  `Corsi_Codice` INT NOT NULL,  
  
  `Corsi_Livelli_Nome` VARCHAR(10) NOT NULL,  
  
  `FasciaOraria_Data` DATE NOT NULL,  
  
  `FasciaOraria_Ora` TIME NOT NULL,  
  
  `Lezione_Corsi_Insegnanti_username` VARCHAR(45) NOT NULL,  
  
  PRIMARY KEY (`Giorni`, `Orario`, `Insegnanti_Nome`, `Corsi_Codice`, `Corsi_Livelli_Nome`,  
  `Lezione_Corsi_Insegnanti_username`),  
  
  CONSTRAINT `fk_Lezione_Corsi_Insegnanti1`  
  
    FOREIGN KEY (`Insegnanti_Nome`, `Lezione_Corsi_Insegnanti_username`)  
  
    REFERENCES `GestioneCorsiDiLingue`.`Insegnanti` (`Nome`, `User_Username`)  
  
    ON DELETE NO ACTION  
  
    ON UPDATE NO ACTION,  
  
  CONSTRAINT `fk_Lezione_Corsi_FasciaOraria1`  
  
    FOREIGN KEY (`FasciaOraria_Data`, `FasciaOraria_Ora`)  
  
    REFERENCES `GestioneCorsiDiLingue`.`FasciaOraria` (`Data`, `Ora`)  
  
    ON DELETE NO ACTION  
  
    ON UPDATE NO ACTION,  
  
  CONSTRAINT `fk_Lezione_Corsi_Codice_Corso`
```

```
FOREIGN KEY (`Corsi_Codice` , `Corsi_Livelli_Nome`)
```

```
REFERENCES `GestioneCorsiDiLingue`.`Corsi` (`Codice` , `Livelli_Nome`)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;
```

```
CREATE          INDEX          `fk_Lezione_Corsi_Insegnanti1_idx`          ON  
`GestioneCorsiDiLingue`.`Lezione_Corsi`          (`Insegnanti_Nome`          ASC,  
`Lezione_Corsi_Insegnanti_username` ASC) ;
```

```
CREATE          INDEX          `fk_Lezione_Corsi_FasciaOraria1_idx`          ON  
`GestioneCorsiDiLingue`.`Lezione_Corsi` (`FasciaOraria_Data` ASC, `FasciaOraria_Ora` ASC) ;
```

```
CREATE          INDEX          `fk_Lezione_Corsi_Codice_Corso_idx`          ON  
`GestioneCorsiDiLingue`.`Lezione_Corsi` (`Corsi_Codice` ASC, `Corsi_Livelli_Nome` ASC) ;
```

```
-----
```

```
-- Table `GestioneCorsiDiLingue`.`Lezione_Privata`
```

```
-----
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Lezione_Privata` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Lezione_Privata` (
```



```
`Giorni` DATE NOT NULL,  
  
`Orario` TIME NOT NULL,  
  
`FasciaOraria_Data` DATE NOT NULL,  
  
`FasciaOraria_Ora` TIME NOT NULL,  
  
`Allievi_Nome` VARCHAR(20) NOT NULL,  
  
`Allievi_Recapito` VARCHAR(45) NOT NULL,  
  
`Insegnanti_Nome` VARCHAR(20) NOT NULL,  
  
PRIMARY KEY (`Giorni`, `Orario`, `Allievi_Nome`, `Allievi_Recapito`, `Insegnanti_Nome`),  
  
CONSTRAINT `fk_Lezione_Privata_FasciaOraria1`  
  
FOREIGN KEY (`FasciaOraria_Data`, `FasciaOraria_Ora`)  
  
REFERENCES `GestioneCorsiDiLingue`.`FasciaOraria` (`Data`, `Ora`)  
  
ON DELETE NO ACTION  
  
ON UPDATE NO ACTION,  
  
CONSTRAINT `fk_Lezione_Privata_Allievi1`  
  
FOREIGN KEY (`Allievi_Nome`, `Allievi_Recapito`)  
  
REFERENCES `GestioneCorsiDiLingue`.`Allievi` (`Nome`, `Recapito`)  
  
ON DELETE NO ACTION  
  
ON UPDATE NO ACTION,  
  
CONSTRAINT `fk_Lezione_Privata_Insegnanti1`  
  
FOREIGN KEY (`Insegnanti_Nome`)  
  
REFERENCES `GestioneCorsiDiLingue`.`Insegnanti` (`Nome`)
```

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

```
CREATE          INDEX          `fk_Lezione_Privata_FasciaOraria1_idx`          ON
`GestioneCorsiDiLingue`.`Lezione_Privata` (`FasciaOraria_Data` ASC, `FasciaOraria_Ora` ASC);
```

```
CREATE          INDEX          `fk_Lezione_Privata_Allievi1_idx`          ON
`GestioneCorsiDiLingue`.`Lezione_Privata` (`Allievi_Nome` ASC, `Allievi_Recapito` ASC);
```

```
CREATE          INDEX          `fk_Lezione_Privata_Insegnanti1_idx`          ON
`GestioneCorsiDiLingue`.`Lezione_Privata` (`Insegnanti_Nome` ASC);
```

```
-- -----
-- Table `GestioneCorsiDiLingue`.`Attività_Culturali`
-- -----
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Attività_Culturali` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Attività_Culturali` (
  `Chiave_ID` INT NOT NULL,
  `Tipologia` ENUM('Conferenza', 'Film') NOT NULL,
  `Conferenziere` VARCHAR(20) NULL,
```

```
`Argomento` VARCHAR(45) NULL,  
  
`Regista` VARCHAR(20) NULL,  
  
`NomeFilm` VARCHAR(20) NULL,  
  
PRIMARY KEY (`Chiave_ID`))  
  
ENGINE = InnoDB;
```

```
-----  
-- Table `GestioneCorsiDiLingue`.`Iscritto`  
-----
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Iscritto` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Iscritto` (  
  
  `Corsi_Codice` INT NOT NULL,  
  
  `Corsi_Livelli_Nome` VARCHAR(10) NOT NULL,  
  
  `Allievi_Nome` VARCHAR(20) NOT NULL,  
  
  `Allievi_Recapito` VARCHAR(45) NOT NULL,  
  
  `Data_Iscrizione` DATE NOT NULL,  
  
  `Numero_assenze` INT NULL,  
  
  PRIMARY KEY (`Corsi_Codice`, `Corsi_Livelli_Nome`, `Allievi_Nome`, `Allievi_Recapito`),  
  
  CONSTRAINT `fk_Corsi_has_Allievi_Corsi1`
```

```
FOREIGN KEY (`Corsi_Codice`, `Corsi_Livelli_Nome`)
```

```
REFERENCES `GestioneCorsiDiLingue`.`Corsi` (`Codice`, `Livelli_Nome`)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION,
```

```
CONSTRAINT `fk_Corsi_has_Allievi_Allievi1`
```

```
FOREIGN KEY (`Allievi_Nome`, `Allievi_Recapito`)
```

```
REFERENCES `GestioneCorsiDiLingue`.`Allievi` (`Nome`, `Recapito`)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;
```

```
CREATE INDEX `fk_Corsi_has_Allievi_Allievi1_idx` ON `GestioneCorsiDiLingue`.`Iscritto`  
(`Allievi_Nome` ASC, `Allievi_Recapito` ASC);
```

```
CREATE INDEX `fk_Corsi_has_Allievi_Corsi1_idx` ON `GestioneCorsiDiLingue`.`Iscritto`  
(`Corsi_Codice` ASC, `Corsi_Livelli_Nome` ASC);
```

```
-- Table `GestioneCorsiDiLingue`.`Prenotazione`
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Prenotazione` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Prenotazione` (  
  
  `Allievi_Nome` VARCHAR(20) NOT NULL,  
  
  `Allievi_Recapito` VARCHAR(45) NOT NULL,  
  
  `Attività_Culturali_Chiave_ID` INT NOT NULL,  
  
  PRIMARY KEY (`Allievi_Nome`, `Allievi_Recapito`, `Attività_Culturali_Chiave_ID`),  
  
  CONSTRAINT `fk_Allievi_has_Actività Culturali_Allievi1`  
  
    FOREIGN KEY (`Allievi_Nome`, `Allievi_Recapito`)  
  
    REFERENCES `GestioneCorsiDiLingue`.`Allievi` (`Nome`, `Recapito`)  
  
    ON DELETE NO ACTION  
  
    ON UPDATE NO ACTION,  
  
  CONSTRAINT `fk_Allievi_has_Actività Culturali_Actività Culturali1`  
  
    FOREIGN KEY (`Attività_Culturali_Chiave_ID`)  
  
    REFERENCES `GestioneCorsiDiLingue`.`Attività_Culturali` (`Chiave_ID`)  
  
    ON DELETE NO ACTION  
  
    ON UPDATE NO ACTION)  
  
ENGINE = InnoDB;  
  
CREATE INDEX `fk_Allievi_has_Actività Culturali_Actività Culturali1_idx` ON  
`GestioneCorsiDiLingue`.`Prenotazione` (`Attività_Culturali_Chiave_ID` ASC);
```

```
CREATE INDEX `fk_Allievi_has_Attività_Culturali_Allievi1_idx` ON  
`GestioneCorsiDiLingue`.`Prenotazione` (`Allievi_Nome` ASC, `Allievi_Recapito` ASC);
```

```
-----  
-- Table `GestioneCorsiDiLingue`.`HaOccupato`  
-----
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`HaOccupato` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`HaOccupato` (  
  
  `Insegnanti_Nome` VARCHAR(20) NOT NULL,  
  
  `FasciaOraria_Data` DATE NOT NULL,  
  
  `FasciaOraria_Ora` TIME NOT NULL,  
  
  `Insegnanti_username` VARCHAR(45) NOT NULL,  
  
  PRIMARY KEY (`Insegnanti_Nome`, `FasciaOraria_Data`, `FasciaOraria_Ora`,  
  `Insegnanti_username`),  
  
  CONSTRAINT `fk_Insegnanti_has_FasciaOraria_Insegnanti1`  
  
    FOREIGN KEY (`Insegnanti_Nome`, `Insegnanti_username`)  
  
    REFERENCES `GestioneCorsiDiLingue`.`Insegnanti` (`Nome`, `User_Username`)  
  
  ON DELETE NO ACTION  
  
  ON UPDATE NO ACTION,  
  
  CONSTRAINT `fk_Insegnanti_has_FasciaOraria_FasciaOraria1`
```

```
FOREIGN KEY (`FasciaOraria_Data`, `FasciaOraria_Ora`)
```

```
REFERENCES `GestioneCorsiDiLingue`.`FasciaOraria` (`Data`, `Ora`)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION)
```

```
ENGINE = InnoDB;
```

```
CREATE      INDEX      `fk_Insegnanti_has_FasciaOraria_FasciaOraria1_idx`      ON  
`GestioneCorsiDiLingue`.`HaOccupato` (`FasciaOraria_Data` ASC, `FasciaOraria_Ora` ASC);
```

```
CREATE      INDEX      `fk_Insegnanti_has_FasciaOraria_Insegnanti1_idx`      ON  
`GestioneCorsiDiLingue`.`HaOccupato` (`Insegnanti_Nome` ASC, `Insegnanti_username` ASC);
```

```
-----  
-- Table `GestioneCorsiDiLingue`.`Frequenza`  
-----
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Frequenza` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Frequenza` (
```

```
`Allievi_Nome` VARCHAR(20) NOT NULL,
```

```
`Allievi_Recapito` VARCHAR(45) NOT NULL,
```

```
`Lezione_Corsi_Giorni` DATE NOT NULL,
```

```

`Lezione_Corsi_Orario` TIME NOT NULL,

`Lezione_Corsi_Insegnanti_Nome` VARCHAR(20) NOT NULL,

`Lezione_Corsi_Corsi_Codice` INT NOT NULL,

`Lezione_Corsi_Corsi_Livelli_Nome` VARCHAR(10) NOT NULL,

PRIMARY KEY (`Allievi_Nome`, `Allievi_Recapito`, `Lezione_Corsi_Giorni`,
`Lezione_Corsi_Orario`, `Lezione_Corsi_Insegnanti_Nome`, `Lezione_Corsi_Corsi_Codice`,
`Lezione_Corsi_Corsi_Livelli_Nome`),

CONSTRAINT `fk_Allievi_has_Lezione_Corsi_Allievi1`

FOREIGN KEY (`Allievi_Nome`, `Allievi_Recapito`)

REFERENCES `GestioneCorsiDiLingue`.`Allievi` (`Nome`, `Recapito`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk_Allievi_has_Lezione_Corsi_Lezione_Corsi1`

FOREIGN KEY (`Lezione_Corsi_Giorni`, `Lezione_Corsi_Orario`,
`Lezione_Corsi_Insegnanti_Nome`, `Lezione_Corsi_Corsi_Codice`,
`Lezione_Corsi_Corsi_Livelli_Nome`)

REFERENCES `GestioneCorsiDiLingue`.`Lezione_Corsi` (`Giorni`, `Orario`, `Insegnanti_Nome`,
`Corsi_Codice`, `Corsi_Livelli_Nome`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk_Allievi_has_Lezione_Corsi_Lezione_Corsi1_idx` ON
`GestioneCorsiDiLingue`.`Frequenza` (`Lezione_Corsi_Giorni` ASC, `Lezione_Corsi_Orario` ASC,

```



```
`Lezione_Corsi_Insegnanti_Nome`      ASC,      `Lezione_Corsi_Corsi_Codice`      ASC,  
`Lezione_Corsi_Corsi_Livelli_Nome` ASC) ;
```

```
CREATE      INDEX      `fk_Allievi_has_Lezione_Corsi_Allievi1_idx`      ON  
`GestioneCorsiDiLingue`.`Frequenza` (`Allievi_Nome` ASC, `Allievi_Recapito` ASC) ;
```

```
-- -----
```

```
-- Table `GestioneCorsiDiLingue`.`Corsi_has_Insegnanti`
```

```
-- -----
```

```
DROP TABLE IF EXISTS `GestioneCorsiDiLingue`.`Corsi_has_Insegnanti` ;
```

```
CREATE TABLE IF NOT EXISTS `GestioneCorsiDiLingue`.`Corsi_has_Insegnanti` (
```

```
`Corsi_Codice` INT NOT NULL,
```

```
`Corsi_Livelli_Nome` VARCHAR(10) NOT NULL,
```

```
`Insegnanti_Nome` VARCHAR(20) NOT NULL,
```

```
PRIMARY KEY (`Corsi_Codice`, `Corsi_Livelli_Nome`, `Insegnanti_Nome`),
```

```
CONSTRAINT `fk_Corsi_has_Insegnanti_Corsi1`
```

```
FOREIGN KEY (`Corsi_Codice`, `Corsi_Livelli_Nome`)
```

```
REFERENCES `GestioneCorsiDiLingue`.`Corsi` (`Codice`, `Livelli_Nome`)
```

```
ON DELETE NO ACTION
```

```
ON UPDATE NO ACTION,
```

```
CONSTRAINT `fk_Corsi_has_Insegnanti_Insegnanti1`  
  
FOREIGN KEY (`Insegnanti_Nome`)  
  
REFERENCES `GestioneCorsiDiLingue`.`Insegnanti` (`Nome`)  
  
ON DELETE NO ACTION  
  
ON UPDATE NO ACTION)  
  
ENGINE = InnoDB;  
  
CREATE          INDEX          `fk_Corsi_has_Insegnanti_Insegnanti1`          ON  
`GestioneCorsiDiLingue`.`Corsi_has_Insegnanti` (`Insegnanti_Nome` ASC);  
  
CREATE          INDEX          `fk_Corsi_has_Insegnanti_Corsi1`          ON  
`GestioneCorsiDiLingue`.`Corsi_has_Insegnanti` (`Corsi_Codice` ASC, `Corsi_Livelli_Nome` ASC)  
;  
  
USE `GestioneCorsiDiLingue` ;  
  
-----  
  
-- procedure Assegnazione_insegnanti_a_corso  
  
-----  
  
USE `GestioneCorsiDiLingue`;  
  
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Assegnazione_insegnanti_a_corso`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Assegnazione_insegnanti_a_corso` (In var_Corsi int, in var_livello  
VARCHAR(10), in var_Insegnanti varchar(20))
```

```
BEGIN
```

```
    declare exit handler for sqlexception
```

```
    BEGIN
```

```
        rollback;
```

```
    resignal;
```

```
END;
```

```
IF var_Insegnanti not in (Select Corsi_has_Insegnanti.Insegnanti_Nome
```

```
    FROM `Corsi_has_Insegnanti`)
```

```
    THEN
```

```
        Insert into `Corsi_has_Insegnanti` (`Corsi_Codice`,  
`Corsi_Livelli_Nome`, `Insegnanti_Nome`) values (var_Corsi, var_livello, var_Insegnanti);
```

```
    ELSE
```

```
        signal sqlstate '45002' set message_text = 'Insegnante assegnato già ad un corso';
```

```
    end if;
```

```
END$$
```

```
DELIMITER ;
```

```
-----  
-- procedure iscrizione_Allievo_a_corso  
-----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`iscrizione_Allievo_a_corso`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `iscrizione_Allievo_a_corso` (IN var_nome VARCHAR(20), IN  
var_recapito VARCHAR(45), in var_corso int, in var_livello VARCHAR(20), in var_data DATE)
```

```
BEGIN
```

```
    declare exit handler for sqlexception
```

```
BEGIN
```

```
    rollback;
```

```
    resignal;
```

```
END;
```

```
    IF var_nome not in(Select Iscritto.Allievi_Nome
```

```
                                FROM `Iscritto`
```

```
                                WHERE
```

```
                                Iscritto.Allievi_Nome = var_nome AND Iscritto.Allievi_Recapito =  
var_recapito)
```

```
THEN

    INSERT into `Iscritto` ( `Allievi_Nome`, `Allievi_Recapito`, `Corsi_Codice`,
`Corsi_Livelli_Nome`, `Data_Iscrizione` ) values (var_nome, var_recapito,var_corso, var_livello,
var_data);

    else signal sqlstate '45002' set message_text = 'allievo già presente in un altro corso';

end if;

END$$

DELIMITER ;

-----

-- procedure assegnazione_insegnante_a_lezione_privata

-----

USE `GestioneCorsiDiLingue`;

DROP                procedure                IF                EXISTS
`GestioneCorsiDiLingue`.`assegnazione_insegnante_a_lezione_privata`;

DELIMITER $$

USE `GestioneCorsiDiLingue`$$

CREATE  PROCEDURE  `assegnazione_insegnante_a_lezione_privata` (in  var_insegnante
VARCHAR(20), in var_allievoNome VARCHAR(20), in var_allievoRecapito VARCHAR(45), in
var_giorno DATE, in var_ora TIME)
```

```
BEGIN
```

```
    declare exit handler for sqlexception
```

```
    BEGIN
```

```
        rollback;
```

```
    resignal;
```

```
END;
```

```
IF var_insegnante not in ( Select HaOccupato.Insegnanti_Nome
```

```
    FROM `HaOccupato`
```

```
        WHERE HaOccupato.Insegnanti_Nome = var_insegnante and
```

```
            HaOccupato.FasciaOraria_Data          =          var_giorno          AND
```

```
HaOccupato.FasciaOraria_Ora = var_oro)
```

```
    THEN
```

```
        insert    into    `Lezione_Privata`    (`Giorni`,    `Orario`,    `FasciaOraria_Data`,  
`FasciaOraria_Ora`, `Allievi_Nome`, `Allievi_Recapito`, `Insegnanti_Nome`) values (var_giorno,  
var_oro, var_giorno, var_oro, var_allievoNome, var_allievoRecapito, var_Insegnante);
```

```
    ELSE
```

```
        signal sqlstate '45002' set message_text = 'Insegnante non disponibile in questa fascia  
oraria';
```

```
    end if;
```

```
END$$
```

```
DELIMITER ;
```

```
-----  
-- procedure Inserimento_degli_allievi_presenti_ad_ogni_Lezione  
-----  
  
USE `GestioneCorsiDiLingue`;  
  
DROP                procedure                IF                EXISTS  
`GestioneCorsiDiLingue`.`Inserimento_degli_allievi_presenti_ad_ogni_Lezione`;  
  
DELIMITER $$  
  
USE `GestioneCorsiDiLingue`$$  
  
CREATE PROCEDURE `Inserimento_degli_allievi_presenti_ad_ogni_Lezione` (in var_nome  
VARCHAR(20), in var_Recapito VARCHAR(45), in var_giorno DATE, in var_ora TIME,in  
var_corso INT, in var_Insegnante VARCHAR(20), IN var_Livello varchar(10))  
  
BEGIN  
  
    insert into Frequenza values (var_giorno, var_ora, var_nome, var_Recapito, var_corso,  
var_Insegnante, var_Livello);  
  
END$$  
  
DELIMITER ;  
  
-----  
  
-- procedure prenotazione_allievo_ad_un_attivita_culturale
```

-----  
  
USE `GestioneCorsiDiLingue`;

DROP procedure IF EXISTS  
`GestioneCorsiDiLingue`.`prenotazione\_allievo\_ad\_un\_attivita\_culturale`;

DELIMITER \$\$

USE `GestioneCorsiDiLingue`\$\$

CREATE PROCEDURE `prenotazione\_allievo\_ad\_un\_attivita\_culturale` (in var\_nome  
VARCHAR(20), in var\_recapito VARCHAR(45), in var\_attivita INT)

BEGIN

    #select `Allievi.Nome`, `Allievi.Recapito`, `Attività Culturali.Chiave\_ID` from `Allievi` join  
    `Attività\_Culturali` on

    INSERT into Prenotazione values ( var\_nome, var\_recapito, var\_attivita);

END\$\$

DELIMITER ;

-----  
  
-- procedure Creazione\_corso  
  
-----

USE `GestioneCorsiDiLingue`;



```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Creazione_corso`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Creazione_corso` ( in var_corso INT, in var_data DATE, in var_libro  
VARCHAR(50), in var_esame VARCHAR(16), in varlivello VARCHAR(45))
```

```
BEGIN
```

```
    declare exit handler for sqlexception
```

```
BEGIN
```

```
        rollback;
```

```
        resignal;
```

```
END;
```

```
IF var_corso not in(Select Corsi.Codice
```

```
FROM `Corsi`
```

```
WHERE
```

```
    Corsi.Codice = var_corso)
```

```
THEN
```

```
    Insert into `Corsi` ( `Codice`, `Data_attivazione`, `Libro`, `Esame`, `Livelli_nome`) values  
(var_corso, var_data, var_libro, var_esame, varlivello);
```

```
ELSE
```

```
    signal sqlstate '45002' set message_text = 'Corso già presente';
```

```
end if;
```

```
END$$
```

```
DELIMITER ;
```

```
-----
```

```
-- procedure Controllo_assenze_studente
```

```
-----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Controllo_assenze_studente`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Controllo_assenze_studente` ( in var_nome VARCHAR(20), in  
var_recapito VARCHAR(45), in var_giorno DATE, in var_corso INT, out var_allievo_assente  
VARCHAR(20))
```

```
BEGIN
```

```
    declare exit handler for sqlexception
```

```
    BEGIN
```

```
        rollback;
```

```
    resignal;
```

```
END;
```

start transaction;

IF var\_giorno in (Select Lezione\_Corsi.Giorni

FROM `Lezione\_Corsi`

WHERE

Lezione\_Corsi.Giorni = var\_giorno AND Lezione\_Corsi.Corsi\_Codice = var\_corso)

THEN

IF var\_nome not in (Select Lezione\_Corsi.Allievi\_Nome

FROM `Frequenza`

WHERE

Lezione\_Corsi.Allievi\_Nome = var\_nome AND  
Lezione\_Corsi.Allievi\_Recapito = var\_recapito and Frequenza.Lezione\_Corsi\_Giorni = var\_giorno  
and Frequenza.Lezione\_Corsi\_Corsi\_Codice = var\_corso)

THEN

set var\_allievo\_assente = var\_nome;

ELSE

signal sqlstate '45002' set message\_text = 'Allievo presente alla lezione';

end if;

ELSE

signal sqlstate '45002' set message\_text = 'Non esiste una lezione di  
questo corso nel giorno selezionato';

end if;

```
        commit;

END$$

DELIMITER ;

-----

-- procedure Generare_report_Insegnante_Mensile
-----

USE `GestioneCorsiDiLingue`;

DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Generare_report_Insegnante_Mensile`;

DELIMITER $$

USE `GestioneCorsiDiLingue`$$

CREATE PROCEDURE `Generare_report_Insegnante_Mensile` (in var_insegnante VARCHAR(20))

BEGIN

    drop temporary table if exists `Report Mensile Insegnante`;

    create temporary table `Report Mensile Insegnate`(`Insegnante` VARCHAR(20),`Nome livello
corso`VARCHAR(20),`Lezione Corso Giorno` DATE,`Lezione Corso Ora` TIME, `Lezione Privata
Giorno` DATE, `Lezione Privata Ora` TIME);

    insert into `Report Mensile Insegnante`

        SELECT  Insegnante.Nome,  Lezione_Corsi.Corsi_Livelli_Nome,  Lezione_Corsi.Giorni,
Lezione_Corsi.Orario, Lezione_Privata.Giorni, Lezione_Privata.Orario
```

```
From `Insegnante`
```

```
JOIN Lezione_Corsi on (Insegnante.Nome = Lezione_Corsi.Insegnanti_Nome) JOIN  
Lezione_Privata on (Insegnante.Nome = Lezione_Privata.Insegnanti.Nome)
```

```
WHERE Insegnante.Nome = var_insegnante and Lezione_Corsi.Giorni < var_giorno and  
Lezione_Corsi.Giorni > var_giorno - 31 and Lezione_Privata.Giorni < var_giorno and  
Lezione_Privata.Giorni > var_giorno - 31;
```

```
drop temporary table `Report Mensile Insegnante`;
```

```
commit;
```

```
END$$
```

```
DELIMITER ;
```

```
-- -----
```

```
-- procedure Creazione_attivita_culturale
```

```
-- -----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Creazione_attivita_culturale`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Creazione_attivita_culturale` (in var_chiave INT, in var_Tipologia  
VARCHAR(10), in var_conferenziere VARCHAR(20), in var_argomento VARCHAR(45), in  
var_regista VARCHAR(20), in var_NomeFilm VARCHAR(20))
```

```
BEGIN
```

```
    declare exit handler for sqlexception
```

```
BEGIN
```

```
    rollback;
```

```
    resignal;
```

```
END;
```

```
    IF var_Tipologia = 'Conferenza' then
```

```
        insert into `Attività_Culturali` (`Chiave_ID`, `Tipologia`, `Conferenziere`, `Argomento`) values  
(var_chiave, var_Tipologia, var_conferenziere, var_argomento);
```

```
    elseif
```

```
        var_Tipologia = 'Film'
```

```
        then insert into `Attività_Culturali` (`Chiave_ID`, `Tipologia`, `Regista`, `NomeFilm`) values  
(var_chiave, var_Tipologia, var_regista, var_NomeFilm);
```

```
    else
```

```
        signal sqlstate '45002' set message_text = 'Attività non supportata';
```

```
    end if;
```

```
END$$
```

```
DELIMITER ;
```

```
-----  
-- procedure Login  
-----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Login`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Login` (in var_username varchar(45), in var_password varchar(45), out  
var_role INT)
```

```
BEGIN
```

```
    declare var_user_role ENUM('Amministrazione', 'Segreteria', 'Insegnante', 'Allievo');
```

```
    select `Ruolo`
```

```
        from `User`
```

```
        where `Username` = var_username
```

```
        and `Password` = md5(var_password)
```

```
        into var_user_role;
```

```
    if var_user_role = 'Amministrazione' then
```

```
        set var_role = 1;
```

```
    elseif var_user_role = 'Segreteria' then
```

```
        set var_role = 2;
```

```
elseif var_user_role = 'Insegnante' then

    set var_role = 3;

elseif var_user_role = 'Allievo' then

    set var_role = 4;

else set var_role = 5;

end if;

END$$

DELIMITER ;

-----

-- procedure Aggiunta_Allievo

-----

USE `GestioneCorsiDiLingue`;

DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Aggiunta_Allievo`;

DELIMITER $$

USE `GestioneCorsiDiLingue`$$

CREATE PROCEDURE `Aggiunta_Allievo` (in var_Nome VARCHAR(20), in var_Recapito
VARCHAR(45), in var_Username VARCHAR(45))

BEGIN
```



```
declare exit handler for sqlexception

BEGIN

    rollback;

    resignal;

END;

if var_Username not in (select `User_Username` from `Allievi` where var_Username =
`User_Username`) then

    INSERT into `Allievi` (`Nome`, `Recapito`,`User_Username`) values (var_Nome, var_Recapito,
var_Username);

else

    signal sqlstate '45002' set message_text = 'Allievo presente o username usato per un
altro allievo';

end if;

commit;

END$$
```

```
DELIMITER ;
```

```
-----
```

```
-- procedure Aggiunta_Insegnante
```

```
-----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Aggiunta_Insegnante`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Aggiunta_Insegnante` (in var_Insegnante VARCHAR(20), in  
var_username VARCHAR(20), in var_indirizzo VARCHAR(45), in var_nazione VARCHAR(20))
```

```
BEGIN
```

```
    INSERT into `Insegnanti` (`Nome`, `User_Username`,`Indirizzo`, `Nazione` ) values  
(var_Insegnante, var_username, var_indirizzo, var_nazione);
```

```
END$$
```

```
DELIMITER ;
```

```
-- -----
```

```
-- procedure Elimina_Corso
```

```
-- -----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Elimina_Corso`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Elimina_Corso` (in var_Corso INT)
```

```
BEGIN
```

```
    DELETE FROM `Corsi` Where Corsi.Codice = var_Corso;
```

```
END$$
```

```
DELIMITER ;
```

```
-- -----
```

```
-- procedure Elimina_Allievo
```

```
-- -----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Elimina_Allievo`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Elimina_Allievo` (in var_Nome VARCHAR(20), in var_Recapito  
VARCHAR(45))
```

```
BEGIN
```

```
    DELETE FROM `Allievi` Where Allievi.Nome = var_Nome && `Allievi.Recapito` =  
var_Recapito;
```

```
END$$
```

DELIMITER ;

-- -----

-- procedure Elimina\_Insegnante

-- -----

USE `GestioneCorsiDiLingue`;

DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Elimina\_Insegnante`;

DELIMITER \$\$

USE `GestioneCorsiDiLingue`\$\$

CREATE PROCEDURE `Elimina\_Insegnante` (in var\_Nome VARCHAR(20))

BEGIN

DELETE FROM `Insegnanti` Where Insegnanti.Nome = var\_Nome;

END\$\$

DELIMITER ;

-- -----

-- procedure Elimina\_Tutti\_Allievi

-----

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Elimina_Tutti_Allievi`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Elimina_Tutti_Allievi` ()
```

```
BEGIN
```

```
    DELETE FROM `Allievi`;
```

```
END$$
```

```
DELIMITER ;
```

-----

```
-- procedure Report_agenda_settimanale
```

-----

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Report_agenda_settimanale`;
```

DELIMITER \$\$

USE `GestioneCorsiDiLingue`\$\$

CREATE PROCEDURE `Report\_agenda\_settimanale` (in var\_Nome VARCHAR(20), in var\_giorno DATE)

BEGIN

drop temporary table if exists `Report Agenda`;

create temporary table `Report Agenda`(`Insegnante` VARCHAR(20), `Nome livello corso` VARCHAR(20), `Lezione Corso Giorno` DATE, `Lezione Corso Ora` TIME, `Lezione Privata Giorno` DATE, `Lezione Privata Ora` TIME);

insert into `Report Agenda`

SELECT Insegnante.Nome, Lezione\_Corsi.Corsi\_Livelli\_Nome, Lezione\_Corsi.Giorni, Lezione\_Corsi.Orario, Lezione\_Privata.Giorni, Lezione\_Privata.Orario

From `Insegnante`

JOIN Lezione\_Corsi on (Insegnante.Nome = Lezione\_Corsi.Insegnanti\_Nome) JOIN Lezione\_Privata on (Insegnante.Nome = Lezione\_Privata.Insegnanti.Nome)

WHERE Insegnante.Nome = var\_insegnante and Lezione\_Corsi.Giorni = var\_giorno + 7 and Lezione\_Privata.Giorni = var\_giorno + 7;

drop temporary table `Report Agenda`;

END\$\$

DELIMITER ;

-----

-- procedure Inserisci\_Utente

```
-----  
  
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`Inserisci_Utente`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `Inserisci_Utente` (in var_username VARCHAR(20), in var_password  
VARCHAR(20), in var_role VARCHAR(20))
```

```
BEGIN
```

```
    insert into User values(var_username, md5(var_password), var_role);
```

```
END$$
```

```
DELIMITER ;
```

```
-----  
  
-- procedure visualizza_Insegnanti_assegnati_a_corsi  
  
-----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`visualizza_Insegnanti_assegnati_a_corsi`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `visualizza_Insegnanti_assegnati_a_corsi` (in var_Nome VARCHAR(45),  
out var_corso INT)
```

```
BEGIN
```

```
    set transaction read only;
```

```
    set transaction isolation level read committed;
```

```
    select Corsi_has_Insegnanti.Corsi_Codice from `Corsi_has_Insegnanti` where var_Nome =  
Corsi_has_Insegnanti.Insegnanti_Nome;
```

```
    set var_corso = Corsi_has_Insegnanti.Corsi_Codice;
```

```
    commit;
```

```
END$$
```

```
DELIMITER ;
```

```
-- -----
```

```
-- procedure visualizza_corsi_assegnati_a_insegnanti
```

```
-- -----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`visualizza_corsi_assegnati_a_insegnanti`;
```



```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `visualizza_corsi_assegnati_a_insegnanti` (in var_Corso INT, out  
var_insegnante VARCHAR(45))
```

```
BEGIN
```

```
    set transaction read only;
```

```
    set transaction isolation level read committed;
```

```
    select Corsi_has_Insegnanti.Insegnanti_Nome from `Corsi_has_Insegnanti` where var_Corso =  
Corsi_has_Insegnanti.Corsi_Codice;
```

```
    set var_insegnante = Corsi_has_Insegnanti.Insegnanti_Nome;
```

```
    commit;
```

```
END$$
```

```
DELIMITER ;
```

```
-- -----
```

```
-- procedure inserisci_lezione
```

```
-- -----
```

```
USE `GestioneCorsiDiLingue`;
```

```
DROP procedure IF EXISTS `GestioneCorsiDiLingue`.`inserisci_lezione`;
```

```
DELIMITER $$
```

```
USE `GestioneCorsiDiLingue`$$
```

```
CREATE PROCEDURE `inserisci_lezione` (in var_giorno DATE, in var_ora TIME, in  
var_insegnante VARCHAR(20), in var_corso INT, in varlivello VARCHAR(10), in var_username  
VARCHAR(45))
```

```
BEGIN
```

```
    insert into FasciaOraria values (var_giorno, var_ora);
```

```
    #insert into HaOccupato values (var_insegnante, var_giorno, var_ora, var_username);
```

```
    insert into Lezione_Corsi values (var_giorno, var_ora, var_insegnante, var_corso, varlivello,  
var_giorno, var_ora, var_username);
```

```
END$$
```

```
DELIMITER ;
```

```
SET SQL_MODE = '';
```

```
DROP USER 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';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Generare_report_Insegnante_Mensile`  
TO 'Amministrazione';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Elimina_Corso` TO  
'Amministrazione';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Aggiunta_Insegnante` TO  
'Amministrazione';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Elimina_Insegnante` TO  
'Amministrazione';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Creazione_corso` TO  
'Amministrazione';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Assegnazione_insegnanti_a_corso`  
TO 'Amministrazione';
```

```
SET SQL_MODE = "";
```

```
DROP USER 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';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`iscrizione_Allievo_a_corso` TO  
'Segreteria';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Elimina_Tutti_Allievi` TO  
'Segreteria';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Elimina_Allievo` TO 'Segreteria';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Aggiunta_Allievo` TO 'Segreteria';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Creazione_attivita_culturale` TO  
'Segreteria';
```

```
GRANT EXECUTE ON procedure  
`GestioneCorsiDiLingue`.`assegnazione_insegnante_a_lezione_privata` TO 'Segreteria';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Inserisci_Utente` TO 'Segreteria';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Controllo_assenze_studente` TO 'Segreteria';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`visualizza_Insegnanti_assegnati_a_corsi` TO 'Segreteria';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`visualizza_corsi_assegnati_a_insegnanti` TO 'Segreteria';
```

```
SET SQL_MODE = '';
```

```
DROP USER 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';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Inserimento_degli_allievi_presenti_ad_ogni_Lezione` TO 'Insegnante';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Report_agenda_settimanale` TO 'Insegnante';
```

```
GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`inserisci_lezione` TO 'Insegnante';
```

```
SET SQL_MODE = '';
```

```
DROP USER Allievo;
```

```
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 'Allievo' IDENTIFIED BY 'allievo';
```

```
GRANT EXECUTE ON procedure
`GestioneCorsiDiLingue`.`prenotazione_allievo_ad_un_attivita_culturale` TO 'Allievo';

SET SQL_MODE = "";

DROP USER 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';

GRANT EXECUTE ON procedure `GestioneCorsiDiLingue`.`Login` TO 'Login';

SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

-- -----
-- Data for table `GestioneCorsiDiLingue`.`Livelli`
-- -----

START TRANSACTION;

USE `GestioneCorsiDiLingue`;

INSERT INTO `GestioneCorsiDiLingue`.`Livelli` (`Nome`) VALUES ('A1');

INSERT INTO `GestioneCorsiDiLingue`.`Livelli` (`Nome`) VALUES ('A2');
```

```
INSERT INTO `GestioneCorsiDiLingue`.`Livelli` (`Nome`) VALUES ('B1');
```

```
INSERT INTO `GestioneCorsiDiLingue`.`Livelli` (`Nome`) VALUES ('B2');
```

```
INSERT INTO `GestioneCorsiDiLingue`.`Livelli` (`Nome`) VALUES ('C1');
```

```
INSERT INTO `GestioneCorsiDiLingue`.`Livelli` (`Nome`) VALUES ('C2');
```

```
COMMIT;
```

```
-- -----  
-- Data for table `GestioneCorsiDiLingue`.`Corsi`  
-- -----
```

```
START TRANSACTION;
```

```
USE `GestioneCorsiDiLingue`;
```

```
INSERT INTO `GestioneCorsiDiLingue`.`Corsi` (`Codice`, `Data_attivazione`, `Libro`, `Esame`,  
`Livelli_Nome`) VALUES (01, '2021-02-07', 'Libro inglese 1', 'Scritto', 'A1');
```

```
COMMIT;
```

```
-- -----  
-- Data for table `GestioneCorsiDiLingue`.`User`  
-- -----
```

```
START TRANSACTION;
```

```
USE `GestioneCorsiDiLingue`;
```

```
INSERT INTO `GestioneCorsiDiLingue`.`User` (`Username`, `Password`, `Ruolo`) VALUES ('aledifi', '81dc9bdb52d04dc20036dbd8313ed055', 'Amministrazione');
```

```
INSERT INTO `GestioneCorsiDiLingue`.`User` (`Username`, `Password`, `Ruolo`) VALUES ('alessandro', '81dc9bdb52d04dc20036dbd8313ed055', 'Segreteria');
```

```
INSERT INTO `GestioneCorsiDiLingue`.`User` (`Username`, `Password`, `Ruolo`) VALUES ('sandro', '81dc9bdb52d04dc20036dbd8313ed055', 'Insegnante');
```

```
INSERT INTO `GestioneCorsiDiLingue`.`User` (`Username`, `Password`, `Ruolo`) VALUES ('ale', '81dc9bdb52d04dc20036dbd8313ed055', 'Allievo');
```

```
INSERT INTO `GestioneCorsiDiLingue`.`User` (`Username`, `Password`, `Ruolo`) VALUES ('di  
filippo', '81dc9bdb52d04dc20036dbd8313ed055', 'Allievo');
```

```
COMMIT;
```

```
-- -----
```

```
-- Data for table `GestioneCorsiDiLingue`.`Insegnanti`
```

```
-- -----
```

```
START TRANSACTION;
```

```
USE `GestioneCorsiDiLingue`;
```

```
INSERT INTO `GestioneCorsiDiLingue`.`Insegnanti` (`Nome`, `Indirizzo`, `Nazione`,  
`User_Username`) VALUES ('sandro', 'Via Udine', 'Italia', 'sandro');
```

COMMIT;

-- -----

-- Data for table `GestioneCorsiDiLingue`.`Allievi`

-- -----

START TRANSACTION;

USE `GestioneCorsiDiLingue`;

INSERT INTO `GestioneCorsiDiLingue`.`Allievi` (`Nome`, `Recapito`, `Data\_assenze`,  
`User\_Username`) VALUES ('di filippo', 'via milano', '2021-02-07', 'di filippo');

COMMIT;

-- -----

-- Data for table `GestioneCorsiDiLingue`.`FasciaOraria`

-- -----

START TRANSACTION;

USE `GestioneCorsiDiLingue`;

INSERT INTO `GestioneCorsiDiLingue`.`FasciaOraria` (`Data`, `Ora`) VALUES ('2021-02-08',  
'09:30:00');

INSERT INTO `GestioneCorsiDiLingue`.`FasciaOraria` (`Data`, `Ora`) VALUES ('2021-02-07',  
'11:30:00');



```
INSERT INTO `GestioneCorsiDiLingue`.`FasciaOraria` (`Data`, `Ora`) VALUES ('2021-02-08', '11:30:00');
```

```
INSERT INTO `GestioneCorsiDiLingue`.`FasciaOraria` (`Data`, `Ora`) VALUES ('2021-02-07', '15:30:00');
```

```
COMMIT;
```

```
-- -----  
-- Data for table `GestioneCorsiDiLingue`.`Attività_Culturali`  
-- -----
```

```
START TRANSACTION;
```

```
USE `GestioneCorsiDiLingue`;
```

```
INSERT INTO `GestioneCorsiDiLingue`.`Attività_Culturali` (`Chiave_ID`, `Tipologia`,  
`Conferenziere`, `Argomento`, `Regista`, `NomeFilm`) VALUES (1, 'Conferenza', 'Tizio', 'Inglese',  
NULL, NULL);
```

```
COMMIT;
```

```
-- -----  
-- Data for table `GestioneCorsiDiLingue`.`Corsi_has_Insegnanti`  
-- -----
```

```
START TRANSACTION;
```

```
USE `GestioneCorsiDiLingue`;
```

```
INSERT INTO `GestioneCorsiDiLingue`.`Corsi_has_Insegnanti` (`Corsi_Codice`,  
`Corsi_Livelli_Nome`, `Insegnanti_Nome`) VALUES (1, 'A1', 'sandro');
```

```
COMMIT; DROP TABLE IF EXISTS `mydb`.`Prenotazione` ;
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`Prenotazione` (  
  `Allievi_Nome` VARCHAR(20) NOT NULL,  
  `Allievi_Recapito` VARCHAR(45) NOT NULL,  
  `Attività_Culturali_Chiave_ID` INT NOT NULL,  
  PRIMARY KEY (`Allievi_Nome`, `Allievi_Recapito`, `Attività_Culturali_Chiave_ID`),  
  INDEX `fk_Allievi_has_Actività Culturali_Actività Culturali1_idx` (`Attività_Culturali_Chiave_ID`  
ASC),  
  INDEX `fk_Allievi_has_Actività Culturali_Allievi1_idx` (`Allievi_Nome` ASC, `Allievi_Recapito`  
ASC))  
ENGINE = InnoDB;
```

```
-----  
-- Table `mydb`.`timestamps`  
-----
```

```
DROP TABLE IF EXISTS `mydb`.`timestamps` ;
```

```
CREATE TABLE IF NOT EXISTS `mydb`.`timestamps` (
```

```
`create_time` TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP,  
`update_time` TIMESTAMP NULL);
```

```
USE `mydb` ;
```

**Codice del Front-End:****Main.c:**

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

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

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

```
#include <mysql.h>
```

```
#include "defines.h"
```

```
typedef enum {
```

```
    AMMINISTRAZIONE = 1,
```

```
    SEGRETERIA = 2,
```

```
    INSEGNANTE = 3,
```

```
    ALLIEVO= 4,
```

```
    FAILED_LOGIN
```

```
} role_t;
```

```
struct configuration conf;
```

```
static MYSQL *conn;
```

```
static role_t attempt_login(MYSQL *conn, char *username, char *password) {
```

```
    MYSQL_STMT *login_procedure;
```

```
    MYSQL_BIND param[3];
```

```
int role = 0;
```

```
if(!setup_prepared_stmt(&login_procedure, "call login(?, ?, ?)", conn)) {  
    print_stmt_error(login_procedure, "Unable to initialize login statement\n");  
    goto err2;  
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;  
param[0].buffer = username;  
param[0].buffer_length = strlen(username);
```

```
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;  
param[1].buffer = password;  
param[1].buffer_length = strlen(password);
```

```
param[2].buffer_type = MYSQL_TYPE_LONG;  
param[2].buffer = &role;  
param[2].buffer_length = sizeof(role);
```

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

```
}
```

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

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

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

```
// Retrieve output parameter
```

```
if(mysql_stmt_fetch(login_procedure)) {  
    print_stmt_error(login_procedure, "Could not buffer results");  
    goto err;  
}
```

```
mysql_stmt_close(login_procedure);

return role;

err:

mysql_stmt_close(login_procedure);

err2:

return FAILED_LOGIN;
}

int main(void) {

    role_t role;

    if(!parse_config("users/Login.json", &conf)) {

        fprintf(stderr, "Unable to load login configuration\n");

        exit(EXIT_FAILURE);

    }

    conn = mysql_init (NULL);

    if (conn == NULL) {

        fprintf (stderr, "mysql_init() failed (probably out of memory)\n");

        exit(EXIT_FAILURE);

    }
}
```

```

    if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf
.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {

        fprintf(stderr, "mysql_real_connect() failed\n");

        mysql_close (conn);

        exit(EXIT_FAILURE);

    }

    printf("+++++++\n| Benvenuto
o nel Sistema di Gestione dell'Istituto  |\n++++++
++++++\n");

    printf("Username: ");

    getInput(128, conf.username, false);

    printf("Password: ");

    getInput(128, conf.password, true);

    role = attempt_login(conn, conf.username, conf.password);

    switch(role) {

        case AMMINISTRAZIONE:

            run_as_Ammministrazione(conn);

            break;

        case SEGRETERIA:

            run_as_Segreteria(conn);

            break;

```



*case* INSEGNANTE:

*run\_as\_Insegnante*(conn);

*break*;

*case* ALLIEVO:

*run\_as\_Allievo*(conn);

*break*;

*case* FAILED\_LOGIN:

*fprintf*(stderr, "Invalid credentials\n");

*exit*(EXIT\_FAILURE);

*break*;

*default*:

*fprintf*(stderr, "Invalid condition at %s:%d\n", \_\_FILE\_\_, \_\_LINE\_\_);

*abort*();

}

*printf*("+++++\n| Arrivederci!! |\n+++++\n");

*mysql\_close* (conn);

*return* 0;

}

**Allievo.c:**

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

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

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

```
#include "defines.h"
```

```
static void prenotazione_attivita_culturale (MYSQL *conn)
```

```
{
```

```
    MYSQL_STMT *prepared_stmt;
```

```
    MYSQL_BIND param[3];
```

```
    char name[21];
```

```
    char recapito[46];
```

```
    char attivita[46];
```

```
    int attivita_int;
```

```
    printf("\nNome Studente: ");
```

```
    getInput(21, name, false);
```

```
    printf("Recapito Studente: ");
```

```
    getInput(46, recapito, false);
```

```
    printf("Codice attivita : ");
```

```
    getInput(46, attivita, false);
```

```
attivit _int = atoi(attivit );
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call prenotazione_allievo_ad_un_attivit _culturale(?, ?, ?)", conn)) {
```

```
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n", false);
```

```
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = name;
```

```
param[0].buffer_length = strlen(name);
```

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

```
param[1].buffer = recapito;
```

```
param[1].buffer_length = strlen(recapito);
```

```
param[2].buffer_type = MYSQL_TYPE_LONG;
```

```
param[2].buffer = &attivit _int;
```

```
param[2].buffer_length = sizeof(attivit _int);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student insertion\n", true);  
}
```

```
if (mysql_stmt_execute(prepared_stmt) != 0) {  
    print_stmt_error(prepared_stmt, "An error occurred while adding the student.");  
}  
  
else {  
    printf("Attivita' Culturale correctly booked\n");  
}
```

```
mysql_stmt_close(prepared_stmt);  
}
```

```
void run_as_Allievo(MYSQL *conn)  
{  
    char options[2] = {'a','b'};  
    char op;  
    printf("Switching to Allievo role...\n");  
  
    if(!parse_config("users/Allievo.json", &conf)) {  
        fprintf(stderr, "Unable to load Allievo configuration\n");  
        exit(EXIT_FAILURE);  
    }
```

```
if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {  
    fprintf(stderr, "mysql_change_user() failed\n");  
    exit(EXIT_FAILURE);  
}
```

```
while(true){  
    printf("+++Choose One***\n");  
    printf("a) Prenotati per un attività culturale\n");  
    printf("b) Quit\n");  
  
    op = multiChoice("Select an option", options, 2);
```

```
switch(op) {  
    case 'a':  
        prenotazione_attivita_culturale(conn);  
        break;  
    case 'b':  
        return;  
  
    default:  
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);  
        abort();
```

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

**Amministrazione.c:**

```
#include <stdio.h>  
  
#include <stdlib.h>  
  
#include <string.h>  
  
#include "defines.h"  
  
static void aggiungi_corso(MYSQL *conn)  
{  
    MYSQL_STMT *prepared_stmt;  
    MYSQL_BIND param[5];  
  
    MYSQL_TIME dataAttivazione;  
  
    char corso[17];  
    int corso_int;  
    char giorno[5];  
    char mese[5];
```

```
char anno[5];
```

```
char libro[51];
```

```
char esame[17];
```

```
char livello[46];
```

```
printf("\nInserisci il Codice del corso: ");
```

```
getInput(17, corso, false);
```

```
printf("Inserisci il Giorno di attivazione: ");
```

```
getInput(5, giorno, false);
```

```
printf("Mese: ");
```

```
getInput(5, mese, false);
```

```
printf("Anno: ");
```

```
getInput(5, anno, false);
```

```
printf("Inserisci il libro del corso: ");
```

```
getInput(51, libro, false);
```

```
printf("Inserisci il tipo di esame finale: ");
```

```
getInput(17, esame, false);
```

```
printf("Inserisci il livello: ");
```

```
getInput(46, livello, false);
```

```
corso_int = atoi(corso);
```

```
dataAttivazione.year = atoi(anno);
```

```
dataAttivazione.month = atoi(mese);
```

```
dataAttivazione.day = atoi(giorno);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call Creazione_corso(?, ?, ?, ?, ?)", conn)) {  
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize exam insertion statement\n", false);  
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_LONG;
```

```
param[0].buffer = &corso_int;
```

```
param[0].buffer_length = sizeof(corso_int);
```

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

```
param[1].buffer = &dataAttivazione;
```

```
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[2].buffer = libro;
```

```
param[2].buffer_length = strlen(libro);
```

```
param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[3].buffer = esame;
```



```
param[3].buffer_length = strlen(esame);
```

```
param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[4].buffer = livello;
```

```
param[4].buffer_length = strlen(livello);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for exam insertion\n",  
true);
```

```
}
```

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

```
    print_stmt_error (prepared_stmt, "An error occurred while adding the exam.");
```

```
} else {
```

```
    printf("Corso aggiunto correttamente...\n");
```

```
}
```

```
mysql_stmt_close(prepared_stmt);
```

```
}
```

```
static void elimina_corso(MYSQL *conn)
```

```
{
```

```
    MYSQL_STMT *prepared_stmt;
```

```
    MYSQL_BIND param[1];
```

```
char chiave[21];

int chiave_int;

printf("\nChiave corso: ");

getInput(21, chiave, false);

chiave_int = atoi(chiave);

if(!setup_prepared_stmt(&prepared_stmt, "call Elimina_Corso(?)", conn)) {

    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n"
, false);

}

memset(param, 0, sizeof(param));

param[0].buffer_type = MYSQL_TYPE_LONG;

param[0].buffer = &chiave_int;

param[0].buffer_length = sizeof(chiave_int);

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

    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student insertion\n"
, true);
```

```
}
```

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

```
    print_stmt_error(prepared_stmt, "An error occurred while adding the student.");
```

```
}
```

```
else {
```

```
    printf("Corso Eliminato\n");
```

```
}
```

```
mysql_stmt_close(prepared_stmt);
```

```
}
```

```
/*static void generare_report_insegnante_mensile(MYSQL *conn)
```

```
{
```

```
}
```

```
*/
```

```
static void aggiunta_insegnante(MYSQL *conn)
```

```
{
```

```
    MYSQL_STMT *prepared_stmt;
```

```
    MYSQL_BIND param[4];
```

```
char insegnante[21];
```

```
char username[21];
```

```
char indirizzo[46];
```

```
char nazione[21];
```

```
printf("\nNome Insegnante: ");
```

```
getInput(21, insegnante, false);
```

```
printf("Username: ");
```

```
getInput(21, username, false);
```

```
printf("Inserisci indirizzo: ");
```

```
getInput(46, indirizzo, false);
```

```
printf("Inserisci nazione: ");
```

```
getInput(21, nazione, false);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call Aggiunta_Insegnante(?, ?, ?, ?)", conn)) {
```

```
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize teacher insertion statement\n", false);
```

```
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = insegnante;
```

```
param[0].buffer_length = strlen(insegnante);
```

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

```
param[1].buffer = username;
```

```
param[1].buffer_length = strlen(username);
```

```
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[2].buffer = indirizzo;
```

```
param[2].buffer_length = strlen(indirizzo);
```

```
param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[3].buffer = nazione;
```

```
param[3].buffer_length = strlen(nazione);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for teacher insertion\n", true);
```

```
}
```

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

```
    print_stmt_error(prepared_stmt, "An error occurred while adding the teacher.");
```

```
}
```

```
else {
```

```
    printf("Insegnante correttamente aggiunto\n");
```

```
}
```

```
mysql_stmt_close(prepared_stmt);
```

```
}
```

```
static void elimina_insegnante(MYSQL *conn)
```

```
{
```

```
    MYSQL_STMT *prepared_stmt;
```

```
    MYSQL_BIND param[1];
```

```
    char insegnante[21];
```

```
    printf("\nInserire nome insegnante: ");
```

```
    getInput(21, insegnante, false);
```

```
    if(!setup_prepared_stmt(&prepared_stmt, "call Elimina_Insegnante(?)", conn)) {
```

```
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n", false);
```

```
    }
```

```
    memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;

param[0].buffer = insegnante;

param[0].buffer_length = strlen(insegnante);


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

    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for teacher insertion\n", true);

}


if (mysql_stmt_execute(prepared_stmt) != 0) {

    print_stmt_error(prepared_stmt, "An error deleting the teacher.");

}

else {

    printf("Insegnante eliminato correttamente\n");

}


mysql_stmt_close(prepared_stmt);

}


static void assegna_insegnante(MYSQL *conn)

{

    MYSQL_STMT *prepared_stmt;

    MYSQL_BIND param[3];
```

```
char chiave[5];
```

```
int chiave_int;
```

```
char insegnante[21];
```

```
char livello[21];
```

```
printf("\nChiave corso: ");
```

```
getInput(5, chiave, false);
```

```
printf("Inserire nome Insegnante: ");
```

```
getInput(21, insegnante, false);
```

```
printf("Livello corso: ");
```

```
getInput(11, livello, false);
```

```
chiave_int = atoi(chiave);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call Assegnazione_insegnanti_a_corso(?, ?, ?)", conn))  
{  
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize teacher insertion statement\n"  
, false);  
}
```

```
memset(param, 0, sizeof(param));
```



```
param[0].buffer_type = MYSQL_TYPE_LONG;
```

```
param[0].buffer = &chiave_int;
```

```
param[0].buffer_length = sizeof(chiave_int);
```

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

```
param[1].buffer = livello;
```

```
param[1].buffer_length = strlen(livello);
```

```
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[2].buffer = insegnante;
```

```
param[2].buffer_length = strlen(insegnante);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for teacher insertion\n", true);
```

```
}
```

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

```
    print_stmt_error(prepared_stmt, "An error occurred while adding the teacher.");
```

```
}
```

```
else {
```

```
    printf("Insegnante assegnato a corso\n");
```

```
}
```

```
mysql_stmt_close(prepared_stmt);
}

void run_as_Amministrazione(MYSQL *conn)
{
    char options[7] = {'a','b','c','d','e','f','g'};

    char op;

    printf("Switching to Amministrazione role...\n");

    if(!parse_config("users/Amministrazione.json", &conf)) {
        fprintf(stderr, "Unable to load Amministrazione configuration\n");
        exit(EXIT_FAILURE);
    }

    if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
        fprintf(stderr, "mysql_change_user() failed\n");
        exit(EXIT_FAILURE);
    }

    while(true){
        printf("+++Choose One***\n");

        printf("a) Aggiungi Corso\n");

        printf("b) Elimina corso\n");
```

```
printf("c) Generare report Insegnante Mensile\n");
```

```
printf("d) Aggiunta Insegnante\n");
```

```
printf("e) Elimina Insegnante\n");
```

```
printf("f) Assegna Insegnante a corso\n");
```

```
printf("g) Quit\n");
```

```
op = multiChoice("Select an option", options, 7);
```

```
switch(op) {
```

```
    case 'a':
```

```
        aggiungi_corso(conn);
```

```
    break;
```

```
    case 'b':
```

```
        elimina_corso(conn);
```

```
    break;
```

```
    /*case 'c':
```

```
        generare_report_insegnante_mensile(conn);
```

```
    break;*/
```

```
    case 'd':
```

```
        aggiunta_insegnante(conn);
```

```
    break;
```

```
    case 'e':
```

```
        elimina_insegnante(conn);
```

```
    break;
```

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

**Inout.c:**

```
#include <unistd.h>
```

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

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

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

```
#include <ctype.h>
```

```
#include <termios.h>
```

```
#include <sys/ioctl.h>
```

```
#include <pthread.h>
```

```
#include <signal.h>
```

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

```
#include "defines.h"
```

```
// Per la gestione dei segnali
```

```
static volatile sig_atomic_t signo;
```

```
typedef struct sigaction sigaction_t;
```

```
static void handler(int s);
```

```
char *getInput(unsigned int lung, char *stringa, bool hide)
```

```
{
```

```
    char c;
```

```
    unsigned int i;
```

```
// Dichiarare le variabili necessarie ad un possibile mascheramento dell'input
sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;

sigaction_t savetstp, savettin, savettou;

struct termios term, oterm;

if(hide) {

    // Svuota il buffer

    (void) fflush(stdout);

    // Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente se
    nza output sulla shell

    sigemptyset(&sa.sa_mask);

    sa.sa_flags = SA_INTERRUPT; // Per non resettare le system call

    sa.sa_handler = handler;

    (void) sigaction(SIGALRM, &sa, &savealrm);

    (void) sigaction(SIGINT, &sa, &saveint);

    (void) sigaction(SIGHUP, &sa, &savehup);

    (void) sigaction(SIGQUIT, &sa, &savequit);

    (void) sigaction(SIGTERM, &sa, &saveterm);

    (void) sigaction(SIGTSTP, &sa, &savetstp);

    (void) sigaction(SIGTTIN, &sa, &savettin);

    (void) sigaction(SIGTTOU, &sa, &savettou);

    // Disattiva l'output su schermo

    if (tcgetattr(fileno(stdin), &oterm) == 0) {
```

```

(void) memcpy(&term, &oterm, sizeof(struct termios));

term.c_lflag &= ~(ECHO|ECHONL);

(void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);

} else {


(void) memset(&term, 0, sizeof(struct termios));

(void) memset(&oterm, 0, sizeof(struct termios));

}

}

```

*// Acquisisce da tastiera al pi  lung - l caratteri*

```

for(i = 0; i < lung; i++) {

(void) fread(&c, sizeof(char), 1, stdin);

if(c == '\n') {

stringa[i] = '\0';

break;

} else

stringa[i] = c;

```

*// Gestisce gli asterischi*

```

if(hide) {

if(c == '\b') // Backspace

(void) write(fileno(stdout), &c, sizeof(char));

else

(void) write(fileno(stdout), "*", sizeof(char));

```

```
    }  
}  
  
// Controlla che il terminatore di stringa sia stato inserito  
  
if(i == lung - 1)  
    stringa[i] = '\0';  
  
// Se sono stati digitati più caratteri, svuota il buffer della tastiera  
  
if(strlen(stringa) >= lung) {  
    // Svuota il buffer della tastiera  
    do {  
        c = getchar();  
    } while (c != '\n');  
}  
  
if(hide) {  
    //L'a capo dopo l'input  
    (void) write(fileno(stdout), "\n", 1);  
  
    // Ripristina le impostazioni precedenti dello schermo  
    (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);  
  
    // Ripristina la gestione dei segnali  
    (void) sigaction(SIGALRM, &savealarm, NULL);
```



```
(void) sigaction(SIGINT, &saveint, NULL);  
(void) sigaction(SIGHUP, &savehup, NULL);  
(void) sigaction(SIGQUIT, &savequit, NULL);  
(void) sigaction(SIGTERM, &saveterm, NULL);  
(void) sigaction(SIGTSTP, &savetstp, NULL);  
(void) sigaction(SIGTTIN, &savettin, NULL);  
(void) sigaction(SIGTTOU, &savettou, NULL);
```

```
// Se era stato ricevuto un segnale viene rilanciato al processo stesso
```

```
if(signo)  
  
    (void) raise(signo);  
  
}
```

```
return stringa;
```

```
}
```

```
// Per la gestione dei segnali
```

```
static void handler(int s) {  
  
    signo = s;  
  
}
```

```
bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive)  
{
```

```
// I caratteri 'yes' e 'no' devono essere minuscoli
```

```
yes = tolower(yes);
```

```
no = tolower(no);
```

```
// Decide quale delle due lettere mostrare come predefinite
```

```
char s, n;
```

```
if(predef) {
```

```
    s = toupper(yes);
```

```
    n = no;
```

```
} else {
```

```
    s = yes;
```

```
    n = toupper(no);
```

```
}
```

```
// Richiesta della risposta
```


```
while(true) {
```


```
    // Mostra la domanda
```

```
    printf("%s [%c/%c]: ", domanda, s, n);
```

```
    char c;
```

```
    getInput(1, &c, false);
```

```
// Controlla quale risposta  stata data
```

```
if(c == '\0') { // getInput() non pu  restituire '\n'!
```

```

    return predef;

} else if(c == yes) {

    return true;

} else if(c == no) {

    return false;

} else if(c == toupper(yes)) {

    if(predef || insensitive) return true;

} else if(c == toupper(yes)) {

    if(!predef || insensitive) return false;

}

}

}

```

```

char multiChoice(char *domanda, char choices[], int num)

```

```

{

    // Genera la stringa delle possibilit 
    char *possib = malloc(2 * num * sizeof(char));

    int i, j = 0;

    for(i = 0; i < num; i++) {

        possib[j++] = choices[i];

        possib[j++] = '/';

    }

    possib[j-1] = '\0'; // Per eliminare l'ultima '/'

```

```
// Chiede la risposta


while(true) {

    // Mostra la domanda

    printf("%s [%s]: ", domanda, possib);

    char c;

    getInput(1, &c, false);

    // Controlla se  un carattere valido

    for(i = 0; i < num; i++) {

        if(c == choices[i])

            return c;

    }

}

}
```

**Insegnante.c:**

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

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

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

```
#include "defines.h"
```

```
static void inserimento_allievi_presenti(MYSQL *conn)
```

```
{
```

```
    MYSQL_STMT *prepared_stmt;
```

```
    MYSQL_BIND param[7];
```

```
    MYSQL_TIME dataPresenza;
```

```
    MYSQL_TIME oraLezione;
```

```
    char nome[21];
```

```
    char recapito[46];
```

```
    char giorno[5];
```

```
    char mese[5];
```

```
    char anno[5];
```

```
    char ora[5];
```

```
    char min[5];
```

```
    char corso[17];
```

```
    int corso_int;
```

```
char insegnante[21];
```

```
char livello[11];
```

```
printf("\nInserisci il Nome dello studente: ");
```

```
getInput(21, nome, false);
```

```
printf("Inserisci il Recapito: ");
```

```
getInput(46, recapito, false);
```

```
printf("Inserisci il giorno della Lezione: ");
```

```
getInput(5, giorno, false);
```

```
printf("Mese: ");
```

```
getInput(5, mese, false);
```

```
printf("Anno: ");
```

```
getInput(5, anno, false);
```

```
printf("Inserisci l'ora: ");
```

```
getInput(5, ora, false);
```

```
printf("minuti: ");
```

```
getInput(5, min, false);
```

```
printf("Inserisci il codice corso: ");
```

```
getInput(17, corso, false);
```

```
printf("Inserisci nome Insegnante: ");
```

```
getInput(21, insegnante, false);
```

```
printf("Inserisci Livello corso: ");
```

```
getInput(11, livello, false);
```

```
corso_int = atoi(corso);
```

```
dataPresenza.year = atoi(anno);
```

```
dataPresenza.month = atoi(mese);
```

```
dataPresenza.day = atoi(giorno);
```

```
oraLezione.hour = atoi(ora);
```

```
oraLezione.minute = atoi(min);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call Inserimento_degli_allievi_presenti_ad_ogni_Lezio  
ne(?, ?, ?, ?, ?, ?, ?)", conn)) {  
  
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insert statement\n", fa  
lse);  
  
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = nome;
```

```
param[0].buffer_length = strlen(nome);
```

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

```
param[1].buffer = recapito;
```

```
param[1].buffer_length = strlen(recapito);
```

```
param[2].buffer_type = MYSQL_TYPE_DATE;
```

```
param[2].buffer = &dataPresenza;
```

```
param[3].buffer_type = MYSQL_TYPE_TIME;
```

```
param[3].buffer = &oraLezione;
```

```
param[4].buffer_type = MYSQL_TYPE_LONG;
```

```
param[4].buffer = &corso_int;
```

```
param[4].buffer_length = sizeof(corso_int);
```

```
param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[5].buffer = insegnante;
```

```
param[5].buffer_length = strlen(insegnante);
```

```
param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[6].buffer = livello;
```

```
param[6].buffer_length = strlen(livello);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student insertion\n", true);
```

```
}
```



```
if (mysql_stmt_execute(prepared_stmt) != 0) {  
    print_stmt_error (prepared_stmt, "An error occurred while adding the student");  
} else {  
    printf("Studente segnato Presente correttamente...\n");  
}
```

```
mysql_stmt_close(prepared_stmt);  
}
```

```
/*static void report_agenda_settimanale(MYSQL *conn)  
{  
  
}  
*/
```

```
static void inserisci_lezione(MYSQL *conn)  
{
```

```
    MYSQL_STMT *prepared_stmt;
```

```
    MYSQL_BIND param[6];
```

```
    MYSQL_TIME dataLezione;
```

```
MYSQL_TIME oraLezione;
```

```
char giorno[5];
```

```
char mese[5];
```

```
char anno[5];
```

```
char ora[5];
```

```
char min[5];
```

```
char insegnante[21];
```

```
char corso[5];
```

```
int corso_int;
```

```
char livello[11];
```

```
char username[46];
```

```
printf("Inserisci il giorno della Lezione: ");
```

```
getInput(5, giorno, false);
```

```
printf("Mese: ");
```

```
getInput(5, mese, false);
```

```
printf("Anno: ");
```

```
getInput(5, anno, false);
```

```
printf("Inserisci l'ora: ");
```

```
getInput(5, ora, false);
```

```
printf("minuti: ");
```

```
getInput(5, min, false);
```

```
printf("Inserisci il codice corso: ");
```

```
    getInput(5, corso, false);

    printf("Inserisci nome Insegnante: ");

    getInput(21, insegnante, false);

    printf("Inserisci username professore: ");

    getInput(46, username, false);

    printf("Inserisci Livello corso: ");

    getInput(11, livello, false);


    corso_int = atoi(corso);


    dataLezione.year = atoi(anno);

    dataLezione.month = atoi(mese);

    dataLezione.day = atoi(giorno);


    oraLezione.hour = atoi(ora);

    oraLezione.minute = atoi(min);

    oraLezione.second = 0;


    if(!setup_prepared_stmt(&prepared_stmt, "call inserisci_lezione(?, ?, ?, ?, ?, ?)", conn)) {

        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize lesson insert statement\n", false);

    }


    memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_DATE;
```

```
param[0].buffer = &dataLezione;
```

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

```
param[1].buffer = &oraLezione;
```

```
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[2].buffer = insegnante;
```

```
param[2].buffer_length = strlen(insegnante);
```

```
param[3].buffer_type = MYSQL_TYPE_LONG;
```

```
param[3].buffer = &corso_int;
```

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

```
param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[4].buffer = livello;
```

```
param[4].buffer_length = strlen(livello);
```

```
param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[5].buffer = username;
```

```
param[5].buffer_length = strlen(username);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for lesson insertion\n", true);  
}
```

```
if (mysql_stmt_execute(prepared_stmt) != 0) {  
    print_stmt_error (prepared_stmt, "An error occurred while adding the lesson");  
} else {  
    printf("Lezione inserita correttamente...\n");  
}
```

```
mysql_stmt_close(prepared_stmt);
```

```
}
```

```
void run_as_Insegnante(MYSQL *conn)
```

```
{
```

```
    char options[4] = {'a','b','c','d'};
```

```
    char op;
```

```
    printf("Switching to Insegnante role...\n");
```

```
    if(!parse_config("users/Insegnante.json", &conf)) {
```

```
        fprintf(stderr, "Unable to load Insegnante configuration\n");
```

```
        exit(EXIT_FAILURE);
```

```
}
```

```
if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {  
    fprintf(stderr, "mysql_change_user() failed\n");  
    exit(EXIT_FAILURE);  
}
```

```
while(true){  
    printf("+++Choose One***\n");  
    printf("a) Inserimento Allievi Presenti a Lezione\n");  
    printf("b) Report Agenda Settimanale\n");  
    printf("c) Inserisci lezione\n");  
    printf("d) Quit\n");
```

```
op = multiChoice("Select an option", options, 4);
```

```
switch(op) {  
    case 'a':  
        inserimento_allievi_presenti(conn);  
        break;  
  
    /*case 'b':  
        report_agenda_settimanale(conn);  
        break;  
  
    */
```

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

**parse.c:**

```
#include <stddef.h>
```

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

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

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

```
#include "defines.h"
```

```
#define BUFF_SIZE 4096
```

```
// The final config struct will point into this
```

```
static char config[BUFF_SIZE];
```

```
/**
```

```
 * JSON type identifier. Basic types are:
```

```
 * o Object
```

```
 * o Array
```

```
 * o String
```

```
 * o Other primitive: number, boolean (true/false) or null
```

```
 */
```

```
typedef enum {
```

```
    JSMN_UNDEFINED = 0,
```

```
    JSMN_OBJECT = 1,
```

```
    JSMN_ARRAY = 2,
```



```
JSMN_STRING = 3,

JSMN_PRIMITIVE = 4

} jsmntype_t;


enum jsmnerr {

    /* Not enough tokens were provided */

    JSMN_ERROR_NOMEM = -1,

    /* Invalid character inside JSON string */

    JSMN_ERROR_INVAL = -2,

    /* The string is not a full JSON packet, more bytes expected */

    JSMN_ERROR_PART = -3

};


/**

 * JSON token description.

 * type    type (object, array, string etc.)

 * start    start position in JSON data string

 * end      end position in JSON data string

 */

typedef struct {

    jsmntype_t type;

    int start;

    int end;

    int size;
```

```
#ifdef JSMN_PARENT_LINKS
```

```
    int parent;
```

```
#endif
```

```
} jsmntok_t;
```

```
/**
```

```
 * JSON parser. Contains an array of token blocks available. Also stores
```

```
 * the string being parsed now and current position in that string
```

```
 */
```

```
typedef struct {
```

```
    unsigned int pos; /* offset in the JSON string */
```

```
    unsigned int toknext; /* next token to allocate */
```

```
    int toksuper; /* superior token node, e.g parent object or array */
```

```
} jsmn_parser;
```

```
/**
```

```
 * Allocates a fresh unused token from the token pool.
```

```
 */
```

```
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens, size_t num_tokens) {
```

```
    jsmntok_t *tok;
```

```
    if (parser->toknext >= num_tokens) {
```

```
        return NULL;
```

```
    }
```

```
    tok = &tokens[parser->toknext++];
```

```
    tok->start = tok->end = -1;

    tok->size = 0;

#ifdef JSMN_PARENT_LINKS

    tok->parent = -1;

#endif

    return tok;
}

/**
 * Fills token type and boundaries.
 */

static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
                           int start, int end) {

    token->type = type;

    token->start = start;

    token->end = end;

    token->size = 0;

}

/**
 * Fills next available token with JSON primitive.
 */

static int jsmn_parse_primitive(jsmn_parser *parser, const char *js,
                               size_t len, jsmntok_t *tokens, size_t num_tokens) {
```

```
jsmntok_t *token;

int start;

start = parser->pos;

for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {

    switch (js[parser->pos]) {

#ifdef JSMN_STRICT

        /* In strict mode primitive must be followed by "," or "\"" or "]" */

        case ':':

#endif

        case '\t' : case '\r' : case '\n' : case ' ' :

        case ',' : case ']' : case '}' :

            goto found;

    }

    if (js[parser->pos] < 32 || js[parser->pos] >= 127) {

        parser->pos = start;

        return JSMN_ERROR_INVALID;

    }

}

#ifdef JSMN_STRICT

    /* In strict mode primitive must be followed by a comma/object/array */

    parser->pos = start;

    return JSMN_ERROR_PART;
```

```
#endif
```

```
found:
```

```
    if (tokens == NULL) {
```

```
        parser->pos--;
```

```
        return 0;
```

```
    }
```

```
    token = jsmn_alloc_token(parser, tokens, num_tokens);
```

```
    if (token == NULL) {
```

```
        parser->pos = start;
```

```
        return JSMN_ERROR_NOMEM;
```

```
    }
```

```
    jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
```

```
#ifdef JSMN_PARENT_LINKS
```

```
    token->parent = parser->toksuper;
```

```
#endif
```

```
    parser->pos--;
```

```
    return 0;
```

```
}
```

```
/**
```

```
 * Fills next token with JSON string.
```

```
*/
```

```
static int jsmn_parse_string(jsmn_parser *parser, const char *js,
```

```
size_t len, jsmntok_t *tokens, size_t num_tokens) {  
    jsmntok_t *token;  
  
    int start = parser->pos;  
  
    parser->pos++;  
  
    /* Skip starting quote */  
    for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {  
        char c = js[parser->pos];  
  
        /* Quote: end of string */  
        if (c == "\"") {  
            if (tokens == NULL) {  
                return 0;  
            }  
            token = jsmn_alloc_token(parser, tokens, num_tokens);  
            if (token == NULL) {  
                parser->pos = start;  
                return JSMN_ERROR_NOMEM;  
            }  
            jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);  
#ifdef JSMN_PARENT_LINKS  
            token->parent = parser->toksuper;
```

```
#endif

    return 0;

}

/* Backslash: Quoted symbol expected */

if (c == '\\' && parser->pos + 1 < len) {

    int i;

    parser->pos++;

    switch (js[parser->pos]) {

        /* Allowed escaped symbols */

        case '\"': case '/' : case '\\': case 'b' :

        case 'f' : case 'r' : case 'n' : case 't' :

            break;

        /* Allows escaped symbol \uXXXX */

        case 'u':

            parser->pos++;

            for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0'; i++) {

                /* If it isn't a hex character we have an error */

                if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) || /* 0-9 */

                    (js[parser->pos] >= 65 && js[parser->pos] <= 70) || /* A-F */

                    (js[parser->pos] >= 97 && js[parser->pos] <= 102)))) { /* a-f */

                    parser->pos = start;

                    return JSMN_ERROR_INVALID;

                }

            }

        }

    }
```

```
        parser->pos++;

    }

    parser->pos--;

    break;

    /* Unexpected symbol */

    default:

        parser->pos = start;

        return JSMN_ERROR_INVALID;

    }

}

}

}

parser->pos = start;

return JSMN_ERROR_PART;

}

/**

 * Parse JSON string and fill tokens.

 */

static int jsmn_parse(jsmn_parser *parser, const char *js, size_t len, jsmntok_t *tokens, unsigned int
num_tokens) {

    int r;

    int i;

    jsmntok_t *token;

    int count = parser->toknext;
```



```
for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {  
  
    char c;  
  
    jsmntype_t type;  
  
    c = js[parser->pos];  
  
    switch (c) {  
  
        case '{': case '[':  
  
            count++;  
  
            if (tokens == NULL) {  
  
                break;  
  
            }  
  
            token = jsmn_alloc_token(parser, tokens, num_tokens);  
  
            if (token == NULL)  
  
                return JSMN_ERROR_NOMEM;  
  
            if (parser->toksuper != -1) {  
  
                tokens[parser->toksuper].size++;  
  
#ifdef JSMN_PARENT_LINKS  
  
                token->parent = parser->toksuper;  
  
#endif  
  
            }  
  
            token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);  
  
            token->start = parser->pos;  
  
            parser->toksuper = parser->toknext - 1;  
  
            break;  
  

```

```
case '}': case ']':

    if (tokens == NULL)

        break;

    type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);

#ifdef JSMN_PARENT_LINKS

    if (parser->toknext < 1) {

        return JSMN_ERROR_INVALID;

    }

    token = &tokens[parser->toknext - 1];

    for (;;) {

        if (token->start != -1 && token->end == -1) {

            if (token->type != type) {

                return JSMN_ERROR_INVALID;

            }

            token->end = parser->pos + 1;

            parser->toksuper = token->parent;

            break;

        }

        if (token->parent == -1) {

            if (token->type != type || parser->toksuper == -1) {

                return JSMN_ERROR_INVALID;

            }

            break;

        }

    }

}
```

```
        token = &tokens[token->parent];
    }

    #else

    for (i = parser->toknext - 1; i >= 0; i--) {

        token = &tokens[i];

        if (token->start != -1 && token->end == -1) {

            if (token->type != type) {

                return JSMN_ERROR_INVALID;

            }

            parser->toksuper = -1;

            token->end = parser->pos + 1;

            break;

        }

    }

    /* Error if unmatched closing bracket */

    if (i == -1) return JSMN_ERROR_INVALID;

    for (; i >= 0; i--) {

        token = &tokens[i];

        if (token->start != -1 && token->end == -1) {

            parser->toksuper = i;

            break;

        }

    }

}

#endif
```

```
        break;

    case '\':

        r = jsmn_parse_string(parser, js, len, tokens, num_tokens);

        if (r < 0) return r;

        count++;

        if (parser->toksuper != -1 && tokens != NULL)

            tokens[parser->toksuper].size++;

        break;

    case '\t' : case '\r' : case '\n' : case ' ':

        break;

    case ':':

        parser->toksuper = parser->toknext - 1;

        break;

    case ',':

        if (tokens != NULL && parser->toksuper != -1 &&

            tokens[parser->toksuper].type != JSMN_ARRAY &&

            tokens[parser->toksuper].type != JSMN_OBJECT) {

#ifdef JSMN_PARENT_LINKS

            parser->toksuper = tokens[parser->toksuper].parent;

#endif

        }

    #else

        for (i = parser->toknext - 1; i >= 0; i--) {

            if (tokens[i].type == JSMN_ARRAY || tokens[i].type == JSMN_OBJECT) {

                if (tokens[i].start != -1 && tokens[i].end == -1) {

                    parser->toksuper = i;

                }

            }

        }

    #endif
```

```
                break;
            }
        }
    }

#endif

    }

    break;

#ifdef JSMN_STRICT

    /* In strict mode primitives are: numbers and booleans */

    case '-': case '0': case '1': case '2': case '3': case '4':

    case '5': case '6': case '7': case '8': case '9':

    case 't': case 'f': case 'n' :

        /* And they must not be keys of the object */

        if (tokens != NULL && parser->toksuper != -1) {

            jsmntok_t *t = &tokens[parser->toksuper];

            if (t->type == JSMN_OBJECT ||

                (t->type == JSMN_STRING && t->size != 0)) {

                return JSMN_ERROR_INVAL;

            }

        }

    }

#else

    /* In non-strict mode every unquoted value is a primitive */

    default:

#endif

#endif
```

```
r = jsmn_parse_primitive(parser, js, len, tokens, num_tokens);  
  
if (r < 0) return r;  
  
count++;  
  
if (parser->toksuper != -1 && tokens != NULL)  
    tokens[parser->toksuper].size++;  
  
break;
```

```
#ifdef JSMN_STRICT
```

```
    /* Unexpected char in strict mode */
```

```
    default:
```

```
        return JSMN_ERROR_INVALID;
```

```
#endif
```

```
    }
```

```
}
```

```
if (tokens != NULL) {
```

```
    for (i = parser->toknext - 1; i >= 0; i--) {
```

```
        /* Unmatched opened object or array */
```

```
        if (tokens[i].start != -1 && tokens[i].end == -1) {
```

```
            return JSMN_ERROR_PART;
```

```
        }
```

```
    }
```

```
}
```

```
    return count;
}

/**
 * Creates a new parser based over a given buffer with an array of tokens
 * available.
 */
static void jsmn_init(jsmn_parser *parser) {
    parser->pos = 0;
    parser->toknext = 0;
    parser->toksuper = -1;
}

static int jsoneq(const char *json, jsmntok_t *tok, const char *s)
{
    if (tok->type == JSMN_STRING
        && (int) strlen(s) == tok->end - tok->start
        && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {
        return 0;
    }
    return -1;
}

static size_t load_file(char *filename)
```

```
{  
  
    FILE *f = fopen(filename, "rb");  
  
    if(f == NULL) {  
  
        fprintf(stderr, "Unable to open file %s\n", filename);  
  
        exit(1);  
  
    }  
  
  
    fseek(f, 0, SEEK_END);  
  
    size_t fsize = ftell(f);  
  
    fseek(f, 0, SEEK_SET); //same as rewind(f);  
  
  
    if(fsize >= BUFF_SIZE) {  
  
        fprintf(stderr, "Configuration file too large\n");  
  
        abort();  
  
    }  
  
  
    fread(config, fsize, 1, f);  
  
    fclose(f);  
  
  
    config[fsize] = 0;  
  
    return fsize;  
}  
  
  
int parse_config(char *path, struct configuration *conf)
```



```
{  
  
    int i;  
  
    int r;  
  
    jsmn_parser p;  
  
    jsmntok_t t[128]; /* We expect no more than 128 tokens */  
  
    load_file(path);  
  
    jsmn_init(&p);  
  
    r = jsmn_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));  
  
    if (r < 0) {  
        printf("Failed to parse JSON: %d\n", r);  
        return 0;  
    }  
  
    /* Assume the top-level element is an object */  
  
    if (r < 1 || t[0].type != JSMN_OBJECT) {  
        printf("Object expected\n");  
        return 0;  
    }  
  
    /* Loop over all keys of the root object */  
  
    for (i = 1; i < r; i++) {  
        if (jsoneq(config, &t[i], "host") == 0) {
```

```
/* We may use strdup() to fetch string value */

conf->host = strdup(config + t[i+1].start, t[i+1].end-t[i+1].start);

i++;

} else if (jsoneq(config, &t[i], "username") == 0) {

    conf->db_username = strdup(config + t[i+1].start, t[i+1].end-t[i+1].start);

    i++;

} else if (jsoneq(config, &t[i], "password") == 0) {

    conf->db_password = strdup(config + t[i+1].start, t[i+1].end-t[i+1].start);

    i++;

} else if (jsoneq(config, &t[i], "port") == 0) {

    conf->port = strtol(config + t[i+1].start, NULL, 10);

    i++;

} else if (jsoneq(config, &t[i], "database") == 0) {

    conf->database = strdup(config + t[i+1].start, t[i+1].end-t[i+1].start);

    i++;

} else {

    printf("Unexpected key: %.*s\n", t[i].end-t[i].start, config + t[i].start);

}

}

return 1;

}
```

**Segreteria.c:**

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

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

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

```
#include "defines.h"
```

```
static void aggiungi_allievo(MYSQL *conn)
```

```
{
```

```
    MYSQL_STMT *prepared_stmt;
```

```
    MYSQL_BIND param[3];
```

```
    char name[21];
```

```
    char recapito[46];
```

```
    char username[46];
```

```
    printf("\nNome Studente: ");
```

```
    getInput(21, name, false);
```

```
    printf("Recapito Studente: ");
```

```
    getInput(46, recapito, false);
```

```
    printf("Username Studente: ");
```

```
    getInput(46, username, false);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call Aggiunta_Allievo(?, ?, ?)", conn)) {  
  
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n"  
, false);  
  
}  
  
memset(param, 0, sizeof(param));  
  
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;  
param[0].buffer = name;  
param[0].buffer_length = strlen(name);  
  
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;  
param[1].buffer = recapito;  
param[1].buffer_length = strlen(recapito);  
  
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;  
param[2].buffer = username;  
param[2].buffer_length = strlen(username);  
  
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {  
  
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student insertion\n"  
, true);  
  
}
```

```
if (mysql_stmt_execute(prepared_stmt) != 0) {  
    print_stmt_error(prepared_stmt, "An error occurred while adding the student.");  
}  
  
else {  
    printf("Studente correctly added\n");  
}  
  
mysql_stmt_close(prepared_stmt);  
}  
  
static void elimina_allievo(MYSQL *conn)  
{  
    MYSQL_STMT *prepared_stmt;  
    MYSQL_BIND param[3];  
  
    char name[21];  
    char recapito[46];  
  
    printf("\nNome Studente: ");  
    getInput(21, name, false);  
    printf("Recapito Studente: ");
```

```
getInput(46, recapito, false);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call Elimina_Allievo(?, ?)", conn)) {
```

```
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n", false);  
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = name;
```

```
param[0].buffer_length = strlen(name);
```

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

```
param[1].buffer = recapito;
```

```
param[1].buffer_length = strlen(recapito);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student insertion\n", true);  
}
```

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

```
    print_stmt_error(prepared_stmt, "An error occurred while adding the student.");
}

else {
    printf("Studente correttamente cancellato\n");
}

mysql_stmt_close(prepared_stmt);
}

static void elimina_tutti_allievi(MYSQL *conn)
{
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[0];

    if(!setup_prepared_stmt(&prepared_stmt, "call Elimina_Tutti_Allievi()", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n", false);
    }

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student insertion\n", true);
    }
}
```

```
if (mysql_stmt_execute(prepared_stmt) != 0) {  
    print_stmt_error(prepared_stmt, "An error occurred while adding the student.");  
}  
else {  
    printf("Studenti correttamente cancellati\n");  
}  
  
mysql_stmt_close(prepared_stmt);  
  
}  
  
static void crea_attivita_culturale(MYSQL *conn)  
{  
    MYSQL_STMT *prepared_stmt;  
    MYSQL_BIND param[6];  
  
    char chiave[21];  
    int chiave_int;  
    char tipo[11];  
    char conferenziere[21];  
    char argomento[46];  
    char regista[21];
```



```
char nomeFilm[21];
```

```
printf("\nChiave attivita': ");
```

```
getInput(21, chiave, false);
```

```
printf("Tipologia Attivita' [Conferenza/Film]: ");
```

```
getInput(11, tipo, false);
```

```
printf("\nConferenziera (Null se si e' selezionato Film): ");
```

```
getInput(21, conferenziera, false);
```

```
printf("\nArgomento (Null se si e' selezionato Film): ");
```

```
getInput(46, argomento, false);
```

```
printf("\nRegista (Null se si e' selezionato Conferenza): ");
```

```
getInput(21, regista, false);
```

```
printf("\nNome Film (Null se si e' selezionato Conferenza): ");
```

```
getInput(21, nomeFilm, false);
```

```
chiave_int = atoi(chiave);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call Creazione_attivita_culturale(?, ?, ?, ?, ?, ?)", conn)
) {
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n"
, false);
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_LONG;
```

```
param[0].buffer = &chiave_int;
```

```
param[0].buffer_length = sizeof(chiave_int);
```

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

```
param[1].buffer = tipo;
```

```
param[1].buffer_length = strlen(tipo);
```

```
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[2].buffer = conferenziere;
```

```
param[2].buffer_length = strlen(conferenziere);
```

```
param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[3].buffer = argomento;
```

```
param[3].buffer_length = strlen(argomento);
```

```
param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[4].buffer = regista;
```

```
param[4].buffer_length = strlen(regista);
```

```
param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[5].buffer = nomeFilm;
```

```
param[5].buffer_length = strlen(nomeFilm);

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

    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student insertion\n", true);

}

if (mysql_stmt_execute(prepared_stmt) != 0) {

    print_stmt_error(prepared_stmt, "An error occurred while adding the student.");

}

else {

    printf("Attivita' correttamente aggiunta\n");

}

mysql_stmt_close(prepared_stmt);

}

static void iscrivi_allievo_corso(MYSQL *conn)

{

    MYSQL_STMT *prepared_stmt;

    MYSQL_BIND param[5];

    MYSQL_TIME dataIscrizione;
```

```
char name[21];

char recapito[46];

char corso[21];

int corso_int;

char livello[21];

char giorno[5];

char mese[5];

char anno[5];


printf("\nNome Studente: ");

getInput(21, name, false);

printf("Recapito Studente: ");

getInput(46, recapito, false);

printf("Codice corso: ");

getInput(21, corso, false);

printf("Livello corso: ");

getInput(21, livello, false);

printf("Giorno iscrizione: ");

getInput(5, giorno, false);

printf("Mese: ");

getInput(5, mese, false);

printf("Anno: ");

getInput(5, anno, false);
```

```
corso_int = atoi(corso);
```

```
dataIscrizione.day = atoi(giorno);
```

```
dataIscrizione.month = atoi(mese);
```

```
dataIscrizione.year = atoi(anno);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call iscrizione_Allievo_a_corso(?, ?, ?, ?, ?)", conn)) {  
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n"  
, false);  
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = name;
```

```
param[0].buffer_length = strlen(name);
```

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

```
param[1].buffer = recapito;
```

```
param[1].buffer_length = strlen(recapito);
```

```
param[2].buffer_type = MYSQL_TYPE_LONG;
```

```
param[2].buffer = &corso_int;

param[2].buffer_length = sizeof(corso_int);


param[3].buffer_type = MYSQL_TYPE_VAR_STRING;

param[3].buffer = livello;

param[3].buffer_length = strlen(livello);


param[4].buffer_type = MYSQL_TYPE_DATE;

param[4].buffer = &dataIscrizione;


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

    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student insertion\n", true);

}


if (mysql_stmt_execute(prepared_stmt) != 0) {

    print_stmt_error(prepared_stmt, "An error occurred while adding the student.");

}

else {

    printf("Studente correttamente iscritto al corso\n");

}


mysql_stmt_close(prepared_stmt);

}
```

```
static void assegna_insegnante_lezione_privata(MYSQL *conn)
{
    MYSQL_STMT *prepared_stmt;

    MYSQL_BIND param[5];

    MYSQL_TIME dataLezionePrivata;

    MYSQL_TIME oraLezionePrivata;

    char nome[21];

    char recapito[46];

    char giorno[5];

    char mese[5];

    char anno[5];

    char ora[5];

    char min[5];

    char insegnante[21];

    printf("\nInserisci il Nome dello studente: ");

    getInput(17, nome, false);

    printf("Inserisci il Recapito: ");

    getInput(46, recapito, false);

    printf("Inserisci il giorno della Lezione: ");
```

```
getInput(5, giorno, false);
```

```
printf("Mese: ");
```

```
getInput(5, mese, false);
```

```
printf("Anno: ");
```

```
getInput(5, anno, false);
```

```
printf("Inserisci l'ora: ");
```

```
getInput(5, ora, false);
```

```
printf("minuti: ");
```

```
getInput(5, min, false);
```

```
printf("Inserisci l'insegnante: ");
```

```
getInput(21, insegnante, false);
```

```
dataLezionePrivata.year = atoi(anno);
```

```
dataLezionePrivata.month = atoi(mese);
```

```
dataLezionePrivata.day = atoi(giorno);
```

```
oraLezionePrivata.hour = atoi(ora);
```

```
oraLezionePrivata.minute = atoi(min);
```

```
oraLezionePrivata.second = 0;
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call assegnazione_insegnante_a_lezione_privata(?, ?, ?, ?, ?)", conn)) {
```

```
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize exam insertion statement\n", false);
```



```
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = insegnante;
```

```
param[0].buffer_length = strlen(nome);
```

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

```
param[1].buffer = nome;
```

```
param[1].buffer_length = strlen(recapito);
```

```
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[2].buffer = recapito;
```

```
param[2].buffer_length = strlen(recapito);
```

```
param[3].buffer_type = MYSQL_TYPE_DATE;
```

```
param[3].buffer = &dataLezionePrivata;
```

```
param[4].buffer_type = MYSQL_TYPE_TIME;
```

```
param[4].buffer = &oraLezionePrivata;
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for lesson insertion\n", true);  
}
```

```
if (mysql_stmt_execute(prepared_stmt) != 0) {  
    print_stmt_error (prepared_stmt, "An error occurred while adding the lesson.");  
} else {  
    printf("Insegnante assegnato a Lezione Privata correttamente!...\n");  
}
```

```
mysql_stmt_close(prepared_stmt);  
}
```

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

```
    char insegnante[46];
```

```
    int corso;
```

```
    printf("\nNome Insegnante: ");
```

```
    getInput(46, insegnante, false);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_Insegnanti_assegnati_a_corsi(?, ?)", conn)) {  
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n", false);  
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = insegnante;
```

```
param[0].buffer_length = strlen(insegnante);
```

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

```
param[1].buffer = &corso;
```

```
param[1].buffer_length = sizeof(corso);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for teacher insertion\n", true);  
}
```

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

```
print_stmt_error(prepared_stmt, "An error occurred while adding the teacher.");

goto out;

}


memset(param, 0, sizeof(param));

param[0].buffer_type = MYSQL_TYPE_LONG;

param[0].buffer = &corso;

param[0].buffer_length = sizeof(corso);


if(mysql_stmt_bind_result(prepared_stmt, param)) {

    finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);

}


if(mysql_stmt_fetch(prepared_stmt)) {

    finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);

}


printf("L'insegnante è assegnato al corso %d", corso);

out:

mysql_stmt_close(prepared_stmt);

}
```

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

    char insegnante[46];
    char corso[5];
    int corso_int;

    printf("\nChiave corso: ");
    getInput(5, corso, false);

    corso_int = atoi(corso);

    if(!setup_prepared_stmt(&prepared_stmt, "call visualizza_corsi_assegnati_a_insegnanti(?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n", false);
    }

    memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_LONG;
```

```
param[0].buffer = &corso_int;
```

```
param[0].buffer_length = sizeof(corso_int);
```

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

```
param[1].buffer = insegnante;
```

```
param[1].buffer_length = strlen(insegnante);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for teacher insertion\n", true);
```

```
}
```

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

```
    print_stmt_error(prepared_stmt, "An error occurred while adding the teacher.");
```

```
    goto out;
```

```
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = insegnante;
```

```
param[0].buffer_length = strlen(insegnante);
```

```
if(mysql_stmt_bind_result(prepared_stmt, param)) {  
    finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);  
}
```

```
if(mysql_stmt_fetch(prepared_stmt)) {  
    finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);  
}
```

```
printf("al corso selezionato e' assegnato l'insegnante : %s", insegnante);
```

```
out:
```

```
mysql_stmt_close(prepared_stmt);  
}
```

```
static void controllo_assenze(MYSQL *conn)
```

```
{  
    MYSQL_STMT *prepared_stmt;  
    MYSQL_BIND param[5];  
  
    MYSQL_TIME dataIscrizione;
```

```
char name[21];
```

```
char recapito[46];

char corso[21];

int corso_int;

char giorno[5];

char mese[5];

char anno[5];

char allievo_assente[21];


printf("\nNome Studente: ");

getInput(21, name, false);

printf("Recapito Studente: ");

getInput(46, recapito, false);

printf("Codice corso: ");

getInput(21, corso, false);

printf("Giorno assenza: ");

getInput(5, giorno, false);

printf("Mese: ");

getInput(5, mese, false);

printf("Anno: ");

getInput(5, anno, false);


corso_int = atoi(corso);
```



```
dataIscrizione.day = atoi(giorno);
```

```
dataIscrizione.month = atoi(mese);
```

```
dataIscrizione.year = atoi(anno);
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call Controllo_assenze_studente(?, ?, ?, ?, ?)", conn)) {  
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize student insertion statement\n"  
, false);  
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = name;
```

```
param[0].buffer_length = strlen(name);
```

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

```
param[1].buffer = recapito;
```

```
param[1].buffer_length = strlen(recapito);
```

```
param[2].buffer_type = MYSQL_TYPE_DATE;
```

```
param[2].buffer = &dataIscrizione;
```

```
param[3].buffer_type = MYSQL_TYPE_LONG;
```

```
param[3].buffer = &corso_int;
```

```
param[3].buffer_length = sizeof(corso_int);
```

```
param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[4].buffer = allievo_assente;
```

```
param[4].buffer_length = strlen(allievo_assente);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for student insertion\n", true);
```

```
}
```

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

```
    print_stmt_error(prepared_stmt, "An error occurred while adding the student.");
```

```
    goto out;
```

```
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = allievo_assente;
```

```
param[0].buffer_length = strlen(allievo_assente);
```

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

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
```

```
}
```

```
// Retrieve output parameter
```

```
if(mysql_stmt_fetch(prepared_stmt)) {
```

```
    finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
```

```
}
```

```
printf("Allievo %s assente nel giorno selezionato", allievo_assente );
```

```
out:
```

```
mysql_stmt_close(prepared_stmt);
```

```
}
```

```
static void create_user(MYSQL *conn)
```

```
{
```

```
    MYSQL_STMT *prepared_stmt;
```

```
    MYSQL_BIND param[3];
```

```
    char options[5] = {'1','2','3','4'};
```

```
    char r;
```

```
    char username[21];
```

```
    char password[21];
```

```
    char ruolo[21];
```

```
printf("\nUsername: ");

getInput(46, username, false);

printf("password: ");

getInput(46, password, true);

printf("Assign a possible role:\n");

printf("\t1) Amministrazione\n");

printf("\t2) Segreteria\n");

printf("\t3) Insegnante\n");

printf("\t4) Allievo\n");

r = multiChoice("Select role", options, 4);

switch(r) {

    case '1':

        strcpy(ruolo, "Amministrazione");

        break;

    case '2':

        strcpy(ruolo, "Segreteria");

        break;

    case '3':

        strcpy(ruolo, "Insegnante");

        break;

    case '4':

        strcpy(ruolo, "Allievo");
```

```
break;
```

```
default:
```

```
fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
```

```
abort();
```

```
}
```

```
if(!setup_prepared_stmt(&prepared_stmt, "call Inserisci_Utente(?, ?, ?)", conn)) {
```

```
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize user insertion statement\n", false);
```

```
}
```

```
memset(param, 0, sizeof(param));
```

```
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[0].buffer = username;
```

```
param[0].buffer_length = strlen(username);
```

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

```
param[1].buffer = password;
```

```
param[1].buffer_length = strlen(password);
```

```
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
param[2].buffer = ruolo;
```

```
param[2].buffer_length = strlen(ruolo);
```

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

    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for user insertion\n", t
rue);

}

if (mysql_stmt_execute(prepared_stmt) != 0) {

    print_stmt_error (prepared_stmt, "An error occurred while adding the user.");

} else {

    printf("User correctly added...\n");

}

mysql_stmt_close(prepared_stmt);

}

void run_as_Segreteria(MYSQL *conn)

{

    char options[11] = {'a','b','c','d','e','f','g','h','i','j','k'};

    char op;

    printf("Switching to Segreteria role...\n");

    if(!parse_config("users/Segreteria.json", &conf)) {

        fprintf(stderr, "Unable to load Segreteria configuration\n");

        exit(EXIT_FAILURE);

    }

}
```

```
if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {  
    fprintf(stderr, "mysql_change_user() failed\n");  
    exit(EXIT_FAILURE);  
}
```

```
while(true){  
    printf("+++Choose One***\n");  
    printf("a) Aggiungi Allievo\n");  
    printf("b) Elimina Allievo\n");  
    printf("c) Elimina Tutti gli Allievi\n");  
    printf("d) Crea attivita' culturale\n");  
    printf("e) Iscriviti allievo a corso\n");  
    printf("f) Assegna insegnante a lezione privata\n");  
    printf("g) Inserisci utente\n");  
    printf("h) Controllo assenze studente\n");  
    printf("i) Visualizza Insegnanti assegnati a Corsi\n");  
    printf("j) Visualizza Corsi assegnati a Insegnanti\n");  
    printf("k) Quit\n");
```

```
op = multiChoice("Select an option", options, 11);
```

```
switch(op) {  
    case 'a':
```

```
aggiungi_allievo(conn);  
  
break;  
  
case 'b':  
  
    elimina_allievo(conn);  
  
break;  
  
case 'c':  
  
    elimina_tutti_allievi(conn);  
  
break;  
  
case 'd':  
  
    crea_attivita_culturale(conn);  
  
break;  
  
case 'e':  
  
    iscrivi_allievo_corso(conn);  
  
break;  
  
case 'f':  
  
    assegna_insegnante_lezione_privata(conn);  
  
break;  
  
case 'g':  
  
    create_user(conn);  
  
break;  
  
case 'h':  
  
    controllo_assenze(conn);  
  
break;  
  
case 'i':
```



```
        visualizza_insegnanti_assegnati_a_corsi(conn);

    break;

    case 'j':

        visualizza_corsi_assegnati_a_insegnanti(conn);

    break;

    case 'k':

        return;

    default:

        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);

        abort();

}

getchar();

}

}
```

**Utils.c:**

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

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

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

```
#include "defines.h"
```

```
void print_stmt_error (MYSQL_STMT *stmt, char *message)
```

```
{  
  
    fprintf(stderr, "%s\n", message);  
  
    if (stmt != NULL) {  
  
        fprintf(stderr, "Error %u (%s): %s\n",  
            mysql_stmt_errno (stmt),  
            mysql_stmt_sqlstate(stmt),  
            mysql_stmt_error (stmt));  
  
    }  
}
```

```
void print_error(MYSQL *conn, char *message)
```

```
{  
  
    fprintf(stderr, "%s\n", message);  
  
    if (conn != NULL) {  
  
        #if MYSQL_VERSION_ID >= 40101  
  
        fprintf(stderr, "Error %u (%s): %s\n",
```

```
mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));

#else

fprintf(stderr, "Error %u: %s\n",

mysql_errno (conn), mysql_error (conn));

#endif

}

}

bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)

{

my_bool update_length = true;

*stmt = mysql_stmt_init(conn);

if (*stmt == NULL)

{

print_error(conn, "Could not initialize statement handler");

return false;

}

if (mysql_stmt_prepare (*stmt, statement, strlen(statement)) != 0) {

print_stmt_error(*stmt, "Could not prepare statement");

return false;

}
```

```
mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);
```

```
return true;
```

```
}
```

```
void finish_with_error(MYSQL *conn, char *message)
```

```
{
```

```
    print_error(conn, message);
```

```
    mysql_close(conn);
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool close_stmt  
)
```

```
{
```

```
    print_stmt_error(stmt, message);
```

```
    if(close_stmt) mysql_stmt_close(stmt);
```

```
    mysql_close(conn);
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
static void print_dashes(MYSQL_RES *res_set)
```

```
{
```

```
    MYSQL_FIELD *field;
```

```
    unsigned int i, j;
```

```
mysql_field_seek(res_set, 0);

putchar('+');

for (i = 0; i < mysql_num_fields(res_set); i++) {

    field = mysql_fetch_field(res_set);

    for (j = 0; j < field->max_length + 2; j++)

        putchar('-');

    putchar('+');

}

putchar('\n');

}

static void dump_result_set_header(MYSQL_RES *res_set)

{

    MYSQL_FIELD *field;

    unsigned long col_len;

    unsigned int i;

    /* determine column display widths -- requires result set to be */

    /* generated with mysql_store_result(), not mysql_use_result() */

    mysql_field_seek (res_set, 0);

    for (i = 0; i < mysql_num_fields (res_set); i++) {
```

```
field = mysql_fetch_field (res_set);

col_len = strlen(field->name);


if (col_len < field->max_length)

    col_len = field->max_length;

if (col_len < 4 && !IS_NOT_NULL(field->flags))

    col_len = 4; /* 4 = length of the word "NULL" */

field->max_length = col_len; /* reset column info */

}


print_dashes(res_set);

putchar(' ');

mysql_field_seek (res_set, 0);

for (i = 0; i < mysql_num_fields(res_set); i++) {

    field = mysql_fetch_field(res_set);

    printf(" %-*s |", (int)field->max_length, field->name);

}

putchar("\n");


print_dashes(res_set);

}


void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)

{
```

```
int i;

int status;

int num_fields;    /* number of columns in result */

MYSQL_FIELD *fields; /* for result set metadata */

MYSQL_BIND *rs_bind; /* for output buffers */

MYSQL_RES *rs_metadata;

MYSQL_TIME *date;

size_t attr_size;


/* Prefetch the whole result set. This in conjunction with

 * STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`

 * updates the result set metadata which are fetched in this

 * function, to allow to compute the actual max length of

 * the columns.

 */

if (mysql_stmt_store_result(stmt)) {

    fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");

    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));

    exit(0);

}


/* the column count is > 0 if there is a result set */

/* 0 if the result is only the final status packet */

num_fields = mysql_stmt_field_count(stmt);
```

```
if (num_fields > 0) {  
  
    /* there is a result set to fetch */  
  
    printf("%s\n", title);  
  
  
    if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {  
  
        finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n", true);  
  
    }  
  
  
    dump_result_set_header(rs_metadata);  
  
  
    fields = mysql_fetch_fields(rs_metadata);  
  
  
    rs_bind = (MYSQL_BIND *)malloc(sizeof (MYSQL_BIND) * num_fields);  
  
    if (!rs_bind) {  
  
        finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);  
  
    }  
  
    memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);  
  
  
    /* set up and bind result set output buffers */  
  
    for (i = 0; i < num_fields; ++i) {  
  
  
        // Properly size the parameter buffer  
  
        switch(fields[i].type) {
```



```
case MYSQL_TYPE_DATE:

case MYSQL_TYPE_TIMESTAMP:

case MYSQL_TYPE_DATETIME:

case MYSQL_TYPE_TIME:

    attr_size = sizeof(MYSQL_TIME);

    break;

case MYSQL_TYPE_FLOAT:

    attr_size = sizeof(float);

    break;

case MYSQL_TYPE_DOUBLE:

    attr_size = sizeof(double);

    break;

case MYSQL_TYPE_TINY:

    attr_size = sizeof(signed char);

    break;

case MYSQL_TYPE_SHORT:

case MYSQL_TYPE_YEAR:

    attr_size = sizeof(short int);

    break;

case MYSQL_TYPE_LONG:

case MYSQL_TYPE_INT24:

    attr_size = sizeof(int);

    break;

case MYSQL_TYPE_LONGLONG:
```

```
        attr_size = sizeof(int);

        break;

    default:

        attr_size = fields[i].max_length;

        break;

    }

    // Setup the binding for the current parameter

    rs_bind[i].buffer_type = fields[i].type;

    rs_bind[i].buffer = malloc(attr_size + 1);

    rs_bind[i].buffer_length = attr_size + 1;

    if(rs_bind[i].buffer == NULL) {

        finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);

    }

}

if(mysql_stmt_bind_result(stmt, rs_bind)) {

    finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n", true);

}

/* fetch and display result set rows */

while (true) {

    status = mysql_stmt_fetch(stmt);
```

```
if (status == 1 || status == MYSQL_NO_DATA)

    break;

putchar('|');

for (i = 0; i < num_fields; i++) {

    if (rs_bind[i].is_null_value) {

        printf(" %-*s |", (int)fields[i].max_length, "NULL");

        continue;

    }

    switch (rs_bind[i].buffer_type) {

        case MYSQL_TYPE_VAR_STRING:

        case MYSQL_TYPE_DATETIME:

            printf(" %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);

            break;

        case MYSQL_TYPE_DATE:

        case MYSQL_TYPE_TIMESTAMP:

            date = (MYSQL_TIME *)rs_bind[i].buffer;

            printf(" %d-%02d-%02d |", date->year, date->month, date->day);
```

```
break;
```

```
case MYSQL_TYPE_STRING:
```

```
printf(" %-*s |", (int)fields[i].max_length, (char *)rs_bind[i].buffer);
```

```
break;
```

```
case MYSQL_TYPE_FLOAT:
```

```
case MYSQL_TYPE_DOUBLE:
```

```
printf(" %.02f |", *(float *)rs_bind[i].buffer);
```

```
break;
```

```
case MYSQL_TYPE_LONG:
```

```
case MYSQL_TYPE_SHORT:
```

```
case MYSQL_TYPE_TINY:
```

```
printf(" %-*d |", (int)fields[i].max_length, *(int *)rs_bind[i].buffer);
```

```
break;
```

```
case MYSQL_TYPE_NEWDECIMAL:
```

```
printf(" %-*.*02lf |", (int)fields[i].max_length, *(float*) rs_bind[i].buffer);
```

```
break;
```

```
default:
```

```
printf("ERROR: Unhandled type (%d)\n", rs_bind[i].buffer_type);
```

```
abort();
```

```
    }  
}  
  
    putchar('\n');  
  
    print_dashes(rs_metadata);  
}  
  
mysql_free_result(rs_metadata); /* free metadata */  
  
/* free output buffers */  
for (i = 0; i < num_fields; i++) {  
    free(rs_bind[i].buffer);  
}  
  
free(rs_bind);  
}  
}
```

**defines.c:**

```
#pragma once
```

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

```
#include <mysql.h>
```

```
struct configuration {
```

```
    char *host;
```

```
    char *db_username;
```

```
    char *db_password;
```

```
    unsigned int port;
```

```
    char *database;
```

```
    char username[128];
```

```
    char password[128];
```

```
};
```

```
extern struct configuration conf;
```

```
extern int parse_config(char *path, struct configuration *conf);
```

```
extern char *getInput(unsigned int lung, char *stringa, bool hide);
```

```
extern bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive);
```

```
extern char multiChoice(char *domanda, char choices[], int num);
```

```
extern void print_error (MYSQL *conn, char *message);
```

```
extern void print_stmt_error (MYSQL_STMT *stmt, char *message);

extern void finish_with_error(MYSQL *conn, char *message);

extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool
close_stmt);

extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);

extern void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);

extern void run_as_Insegnante(MYSQL *conn);

extern void run_as_Allievo(MYSQL *conn);

extern void run_as_Amministrazione(MYSQL *conn);

extern void run_as_Segreteria(MYSQL *conn);
```

**Allievo.json:**

```
{  
  "host": "localhost",  
  "username": "Allievo",  
  "password": "allievo",  
  "port": 3306,  
  "database": "GestioneCorsiDiLingue"  
}
```

**Amministrazione.json:**

```
{  
  "host": "localhost",  
  "username": "Amministrazione",  
  "password": "amministrazione",  
  "port": 3306,  
  "database": "GestioneCorsiDiLingue"  
}
```



**Insegnante.json:**

```
{  
  "host": "localhost",  
  "username": "Insegnante",  
  "password": "insegnante",  
  "port": 3306,  
  "database": "GestioneCorsiDiLingue"  
}
```

**Login.json:**

```
{  
  "host": "localhost",  
  "username": "Login",  
  "password": "login",  
  "port": 3306,  
  "database": "GestioneCorsiDiLingue"  
}
```

**Segreteria.json:**

```
{  
  "host": "localhost",  
  "username": "Segreteria",  
  "password": "segreteria",  
  "port": 3306,  
  "database": "GestioneCorsiDiLingue"  
}
```