Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

SISTEMA PER LA GESTIONE DI NOLEGGIO DI VIDEOCASSETTE O DVD

0266643

Matteo Ciccaglione

Indice

1. Descrizione del Minimondo 2

2. Analisi dei Requisiti 4

3. Progettazione concettuale 8

4. Progettazione logica 16

5. Progettazione fisica 26

Appendice: Implementazione 70

1. Descrizione del Minimondo

Sistema per la gestione di noleggio di videocassette e DVD

Si vuole realizzare un sistema informativo per la gestione di una catena di centri di servizio per il noleggio delle videocassette e DVD, tenendo conto delle seguenti informazioni.

Ogni centro di servizio è identificato attraverso un codice numerico univoco; inoltre viene riportato l'indirizzo del centro, i contatti telefonici ed email, il nome di un responsabile.

Il sistema mantiene le informazioni relative a tutte le persone impiegate presso la catena. Per ciascun impiegato sono noti il codice fiscale, il nome, il titolo di studio ed un recapito. Gli impiegati possono essere spostati da un centro all'altro a seconda delle esigenze; si vuole pertanto tenere traccia di tutti gli intervalli di tempo in cui un impiegato ha prestato servizio presso un centro e della carica che ha rivestito in quel periodo (per esempio, cassiere o commesso). Le informazioni sul personale sono gestite dai manager, che sono in grado di inserire nuovo personale nel sistema e di visualizzare report mensili ed annuali sulle ore lavorate e sui luoghi di lavoro. I turni di lavoro sono inseriti, su base mensile, sempre dai manager

I film disponibili presso la catena sono identificati dal titolo e dal nome del regista; inoltre sono noti l'anno in cui il film è stato girato, l'elenco degli attori principali del film, il costo corrente di noleggio della videocassetta ed eventualmente i film disponibili presso la catena di cui il film in questione rappresenta la versione "remake".

Per ogni film è nota la collocazione all'interno di ciascun centro di servizio. In particolare, sono noti il settore, la posizione all'interno del settore ed il numero di copie in cui il film è disponibile. Ciascun settore è identificato attraverso un codice numerico univoco all'interno del centro di servizi e dal codice del centro di servizio stesso. I film sono differenziati in classici e nuove uscite, ciascuna associata ad un costo di noleggio giornaliero differente.

I clienti della catena, al momento del noleggio, ricevono una tessera cliente. Per ciascun cliente devono essere mantenute tutte le informazioni anagrafiche e viene associato anche un numero arbitrario di indirizzi e di recapiti (telefono, email, cellulare) a cui possono essere contattati. Quando un cliente effettua un noleggio, viene registrata la data entro cui il film dovrà essere restituito. Il personale della catena può gestire l'anagrafica dei clienti e gestire gli ordini. Inoltre, può visualizzare in ogni momento, per ciascun centro di servizio, quali titoli sono associati ad un noleggio scaduto e quali sono i clienti che hanno effettuato

tali noleggio.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
21	disponibile	noleggiabile	Il termine disponibile è già usato alla riga 13 per riferirsi alla disponibilità del titolo presso la catena. In questo caso si usa in riferimento al concetto di copia di film
29	film	Film effettivo	Qui il termine film si usa per riferirsi alla copia concreta del film che viene noleggiata e non il film in quanto tale.
31	titoli	film	Il termine titolo è utilizzato per riferirsi al film come concetto astratto (lo stesso definito nel paragrafo righe 15-18) e precedentemente è stato utilizzato per riferirsi al nome del film (riga 15)

Specifica disambiguata

Sistema per la gestione di noleggio di videocassette e DVD

Si vuole realizzare un sistema informativo per la gestione di una catena di centri di servizio per il noleggio delle videocassette e DVD, tenendo conto delle seguenti informazioni. Ogni centro di servizio è identificato attraverso un codice numerico univoco; inoltre viene riportato l'indirizzo del centro, i contatti telefonici ed email, il nome di un responsabile.
Il sistema mantiene le informazioni relative a tutte le persone impiegate presso la catena. Per ciascun impiegato sono noti il codice fiscale, il nome, il titolo di studio ed un recapito. Gli impiegati possono essere spostati da un centro all'altro a seconda delle esigenze; si vuole pertanto tenere traccia di tutti gli intervalli di tempo in cui un impiegato ha prestato servizio presso un centro e della carica che ha rivestito in quel periodo (per esempio, cassiere o commesso). Le informazioni sul personale sono gestite dai manager, che sono in grado di inserire nuovo personale nel sistema e di visualizzare report mensili ed annuali sulle ore lavorate e sui luoghi di lavoro. I turni di lavoro sono inseriti, su base mensile, sempre dai manager
I film disponibili presso la catena sono identificati dal titolo e dal nome del regista; inoltre sono noti l'anno in cui il film è stato girato, l'elenco degli attori principali del film, il costo corrente di

noleggio della videocassetta ed eventualmente i film disponibili presso la catena di cui il film in questione rappresenta la versione "remake".
Per ogni film è nota la collocazione all'interno di ciascun centro di servizio. In particolare, sono noti il settore, la posizione all'interno del settore ed il numero di copie in cui il film è noleggiabile.
Ciascun settore è identificato attraverso un codice numerico univoco all'interno del centro di servizi e dal codice del centro di servizio stesso. I film sono differenziati in classici e nuove uscite, ciascuna associata ad un costo di noleggio giornaliero differente.
I clienti della catena, al momento del noleggio, ricevono una tessera cliente. Per ciascun cliente devono essere mantenute tutte le informazioni anagrafiche e viene associato anche un numero arbitrario di indirizzi e di recapiti (telefono, email, cellulare) a cui possono essere contattati. Quando un cliente effettua un noleggio, viene registrata la data entro cui il film effettivo dovrà essere restituito. Il personale della catena può gestire l'anagrafica dei clienti e gestire gli ordini. Inoltre, può visualizzare in ogni momento, per ciascun centro di servizio, quali film sono associati ad un noleggio scaduto e quali sono i clienti che hanno effettuato tali noleggio.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Film	Titolo del film con annessi regista attori e anno di produzione. Può essere un film classico una nuova uscita o altro. Rappresenta una entità astratta ovvero il concetto di film e non una copia fisica noleggiabile	titolo	Settore,Copia di film
Copia di film	Copia fisica di un dato film per un dato settore. Può essere	videocassetta	Settore,Cliente,Film

	noleggiata.		
Cliente	Persona fisica in grado di poter noleggiare una copia di un film.		Copia di film.
Settore	Parte di un centro di servizio. Contiene le copie associate ad un dato titolo di film		Centro di servizio,Film,Copia di film
Centro di servizio	Negozio fisico della catena di centri di servizio. Contiene uno o più settori e può avere diversi impiegati. Ha un responsabile. Ha una unica sede.		Impiegato, Settore
Impiegato	Persona fisica che lavora presso un centro di servizio della catena. Non può essere un responsabile del centro. Può controllare I noleggi scaduti per un dato centro. Può essere spostato da un centro all'altro e cambiare il proprio ruolo nel centro.	Personale della catena	Centro

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a Film
I film disponibili presso la catena sono identificati dal titolo e dal nome del regista; inoltre sono noti l'anno in cui il film è stato girato, l'elenco degli attori principali del film, il costo corrente di noleggio della videocassetta ed eventualmente i film disponibili presso la catena di cui il film in questione rappresenta la versione "remake". Per ogni film è nota la collocazione all'interno di ciascun centro di servizio. I film sono differenziati in classici e nuove uscite, ciascuna associata ad un costo di noleggio giornaliero differente. Quali titoli sono associati ad un noleggio scaduto e quali sono i clienti che hanno effettuato tali noleggi.

Frasi relative a Copia di film
la posizione all'interno del settore ed il numero di copie in cui il film è noleggiabile. la data entro cui il film dovrà essere restituito.

Frasi relative a Centro di servizio
Ogni centro di servizio è identificato attraverso un codice numerico univoco; inoltre viene riportato l'indirizzo del centro, i contatti telefonici ed email, il nome di un responsabile.

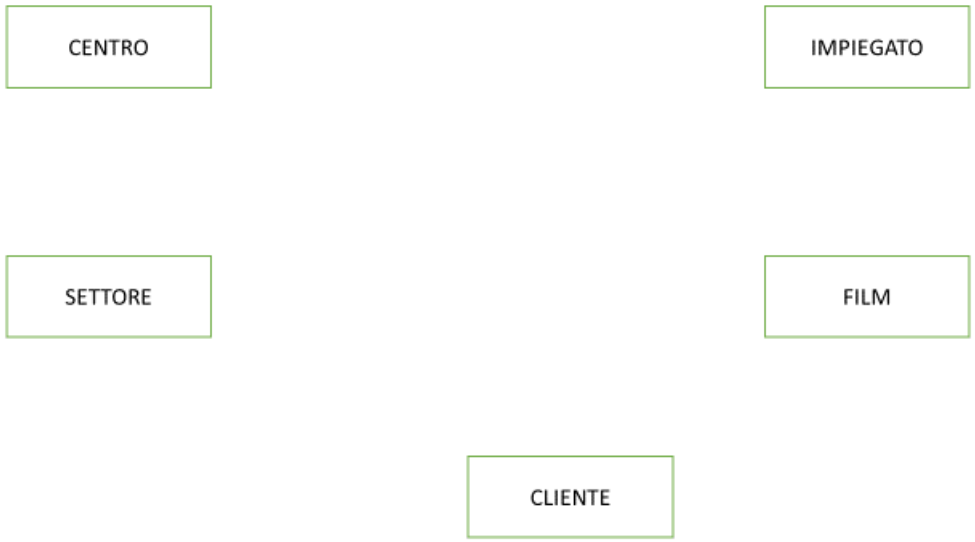
Frasi relative a Impiegato
Per ciascun impiegato sono noti il codice fiscale, il nome, il titolo di studio ed un recapito. Gli impiegati possono essere spostati da un centro all'altro a seconda delle esigenze; si vuole pertanto tenere traccia di tutti gli intervalli di tempo in cui un impiegato ha prestato servizio presso un centro e della carica che ha rivestito in quel periodo (per esempio, cassiere o commesso).

Frasi relative a Cliente
Per ciascun cliente devono essere mantenute tutte le informazioni anagrafiche e viene associato anche un numero arbitrario di indirizzi e di recapiti (telefono, email, cellulare) a cui possono essere contattati. Quando un cliente effettua un noleggio, viene registrata la data entro cui il film effettivo dovrà essere restituito.

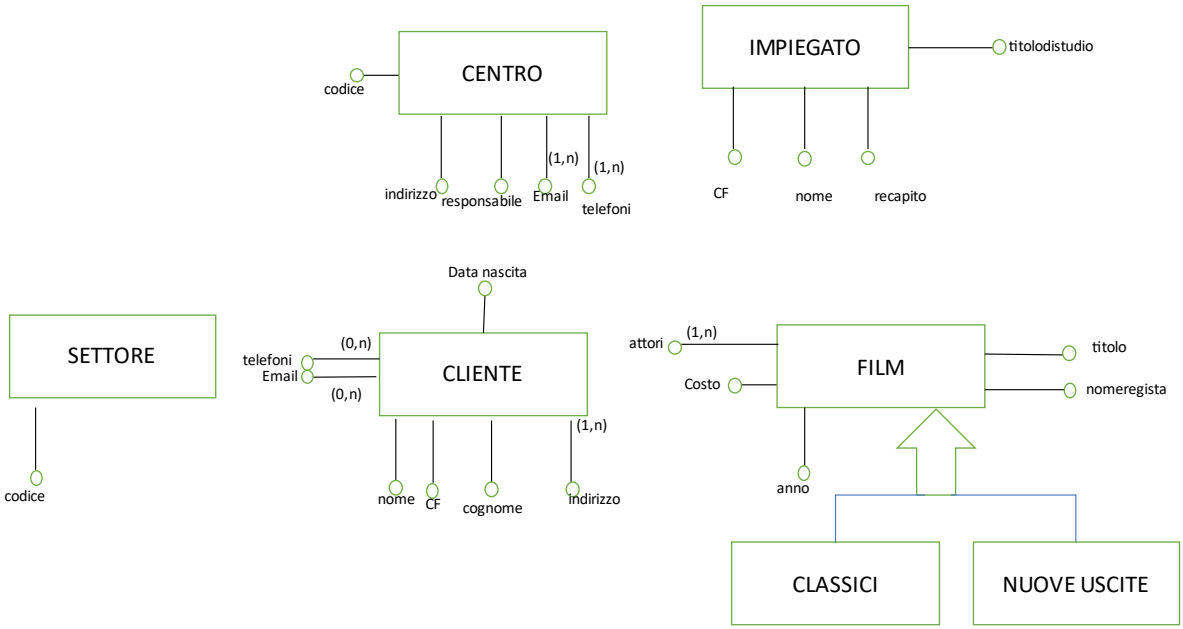
3. Progettazione concettuale

Costruzione dello schema E-R

Partendo dall'analisi dei requisiti rappresento come entità i concetti maggiormente rilevanti riportati anche nel glossario dei termini



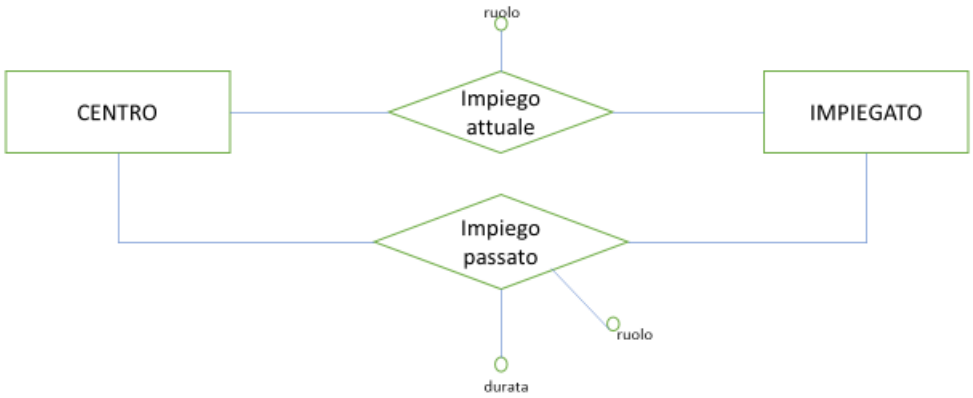
Completo la rappresentazione aggiungendo gli attributi per ciascun entità in base a quanto si evince dalle frasi relative a tale concetto nella fase di analisi. Inoltre aggiungo una generalizzazione all'entità film per riuscire a catturare la distinzione tra film classici e nuove uscite.



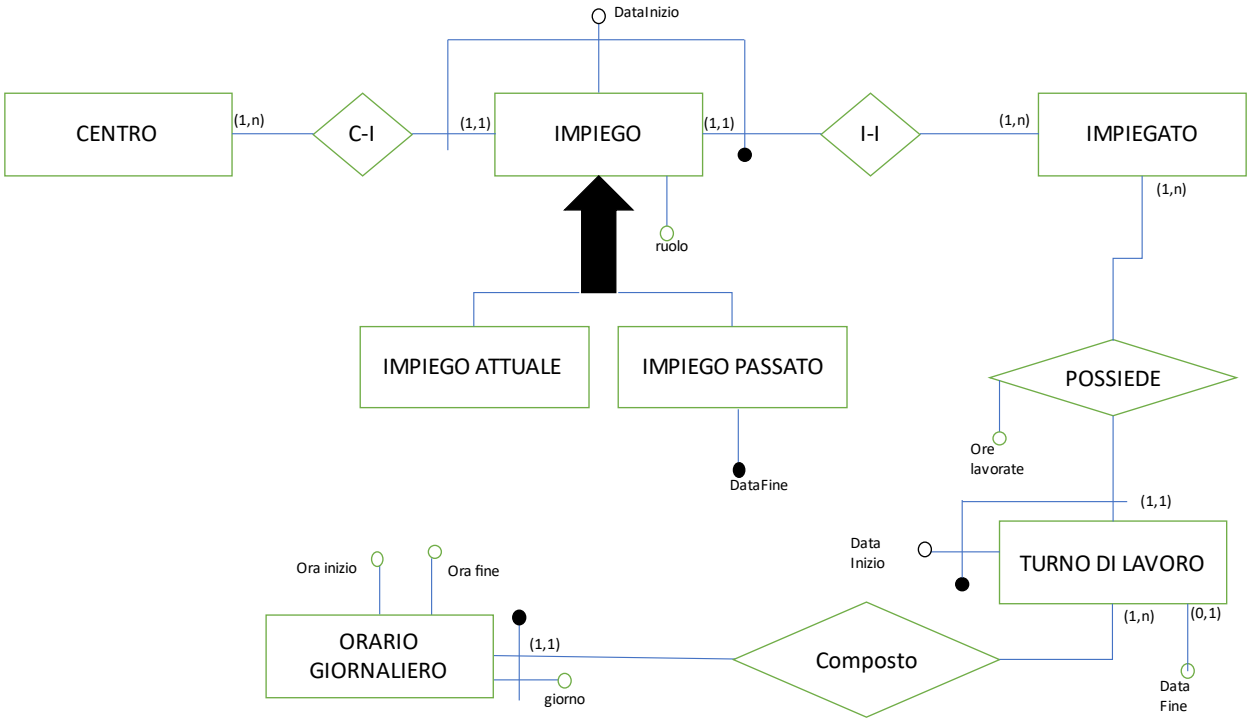
Per una questione di praticità da questo punto in poi si ometteranno gli attributi già individuati per ciascusa entità inserendo i soli attributi dedotti dal confronto tra i concetti.

Ora concentriamoci sulle singole relazioni tra i concetti che sono emerse leggendo la specifica.

Partiamo dal rapporto tra impiegato e centro

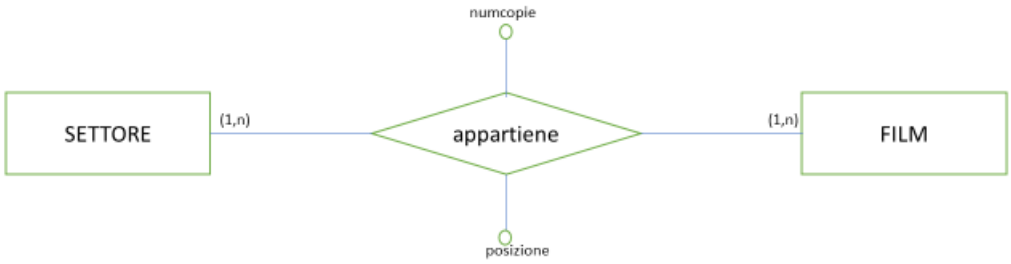


Una prima rappresentazione potrebbe essere questa, tuttavia si osserva che un impiegato può lavorare più volte con diversa durata e diverso ruolo presso lo stesso centro ma questo non è catturato completamente dalla soluzione per cui passiamo ad una reificazione della relazione.

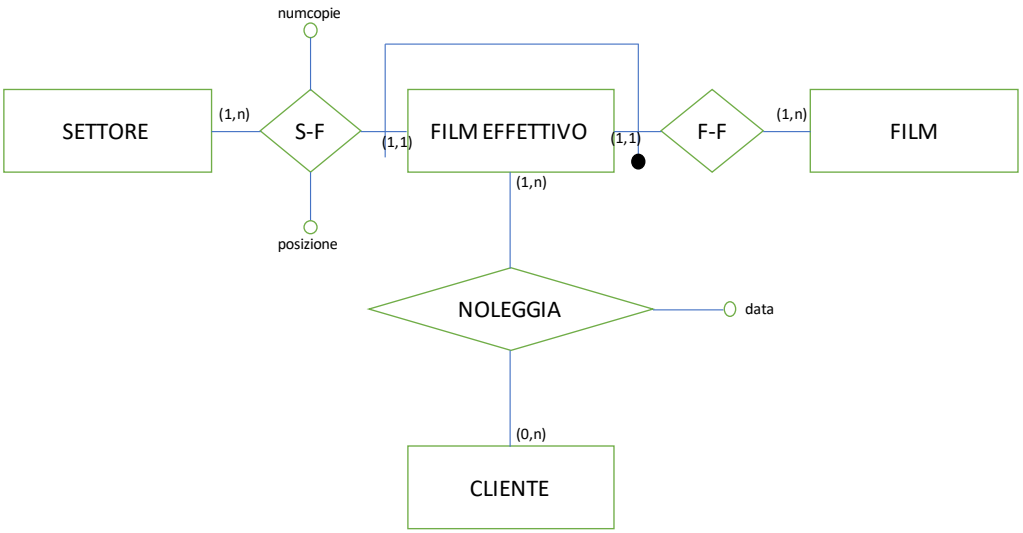


Inoltre si è aggiunta l'entità turno di lavoro per catturare i turni di lavoro associati ad un impiegato per risolvere il problema di stampare report sulle ore lavorate. Di fatti da questa necessità emerge il bisogno di mantenere informazioni relative ai turni per ciascun impiegato.

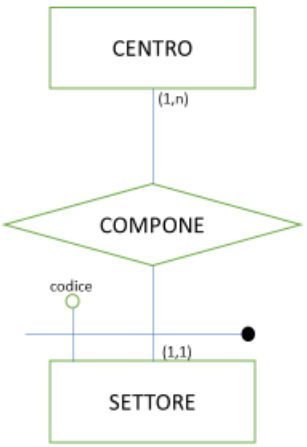
Ora spostiamo l'attenzione sul rapporto tra il concetto di film e quello di settore dove il film è collocato.



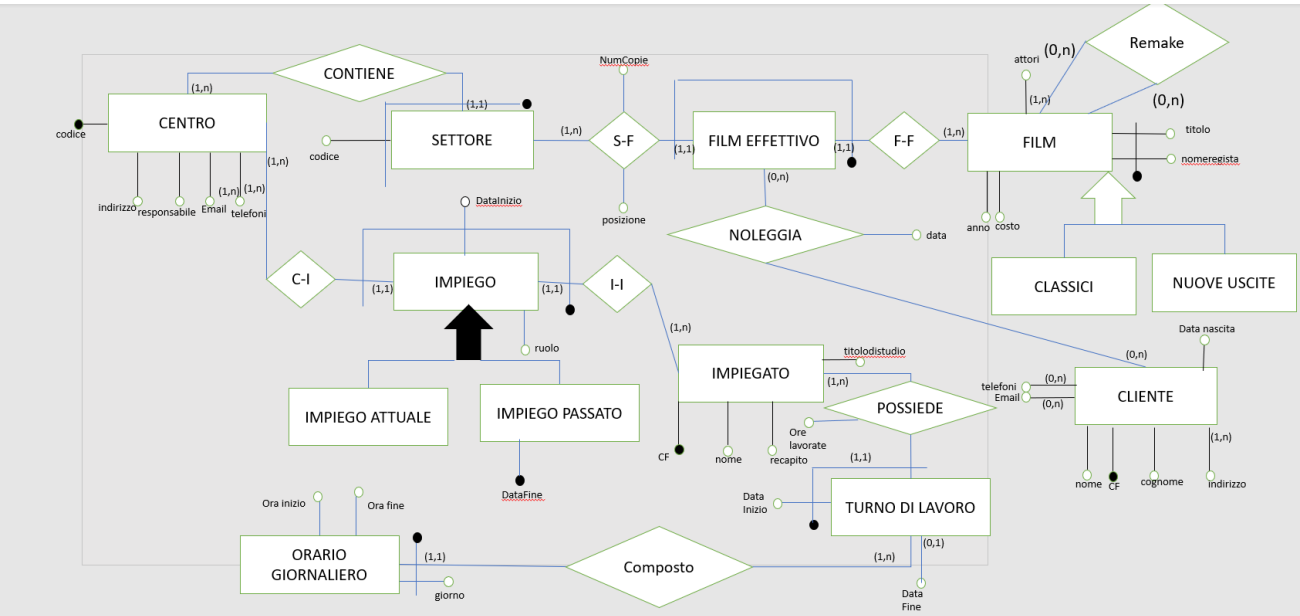
Questa relazione sembra catturare tutti gli aspetti relativi all'interazione tra un film e un settore, tuttavia se si introduce il concetto di cliente e la necessità per un cliente di noleggiare una specifica copia di un film presso un determinato settore allora si potrebbe pensare di passare ad una associazione ternaria ma questa soluzione non è totalmente corretta infatti il rapporto tra film e settore è di un concetto diverso(è una appartenenza) rispetto a quello esistente tra settore film e cliente che è un rapporto di noleggio. Si passa quindi anche qui ad una reificazione.



A questo punto non rimane altro di considerare il rapporto tra il concetto di settore e quello di centro.



Integrazione finale



Regole aziendali

Il valore dell'attributo Ora fine di Orario Giornaliero deve essere maggiore del valore dell'attributo Ora Inizio di Orario Giornaliero.

L'attributo numeroCopie deve essere maggiore o uguale a 0.

Ogni cliente deve avere almeno un telefono o una email

Dizionario dei dati

Per le entità deboli ho riportato come identificatori (oltre ad eventuali attributi dell'entità stessa) i nomi delle entità forti a cui sono associate invece di riportare i loro identificatori per una lettura più comprensibile.

Entità	Descrizione	Attributi	Identificatori
CENTRO	Rappresenta il concetto di centro di servizio della catena. Ogni centro ha uno o più dipendenti ed è composto da uno o più settori.	codice,indirizzo,responsabile,email,telefoni	codice
IMPIEGATO	Rappresenta il concetto di impiegato in quanto essere umano che lavora presso un centro di servizio della catena. Pertanto l'impiegato è l'individuo in sè e non la sua associazione ad un	CF,nome,recapito,titolodistudio	CF

	determinato centro.		
IMPIEGO	Rappresenta il concetto di impiego come il legame lavorativo esistente tra un determinato impiegato e un determinato centro della catena. Un impiego può essere un contratto di lavoro attualmente valido o uno già scaduto.	Ruolo	Centro,Impiegato
FILM	Rappresenta il film in quanto tale ovvero un determinato titolo con un determinato regista e una certa troupe. Un film può essere classico nuova uscita o nessuna delle due. Rappresenta solo l'idea del film e non la sua materializzazione in videocassette(o DVD)	Attori,anno,titolo, nomeregista,costo	Titolo,nomeregista
SETTORE	Un settore è una parte di un centro di servizio. Può contenere un certo numero di film ciascuno in copie differenti.	Codice	Codice,Centro
FILM EFFETTIVO	Un film effettivo è la reale materializzazione del film e la sua collocazione in un settore di un centro di servizio.		Settore,Film
CLIENTE	Un cliente è un individuo dotato di una tessera cliente che può effettuare noleggi su un determinato film effettivo	CF,nome,cognome,indirizzo,telefoni,email	CF
IMPIEGO ATTUALE	Rappresenta l'attuale rapporto lavorativo di un impiegato presso un centro. E' una sotto entità di Impiego		Impiegato,Centro
IMPIEGO PASSATO	Rappresenta i precedenti rapporti lavorativi di un impiegato presso un centro. E' una sotto entità di Impiego	DataFine	DataFine, Impiegato, Centro
CLASSICI	Rappresenta i film di tipologia "Classici". E' una sotto entità di Film		Titolo,nomeregista
NUOVE USCITE	Rappresenta i film di tipologia "Nuove uscite". E' una sotto entità di Film		Titolo,nomeregista
TURNO DI LAVORO	Rappresenta il turno di lavoro assegnato ad un impiegato composto da uno o più giorni lavorativi ciascuno con orario di inizio e fine.	DataInizio	DataInizio,Impiegato
ORARIO GIORNALIERO	Rappresenta per un turno di lavoro e un giorno della settimana l'orario di lavoro previsto per l'impiegato	Giorno	Giorno,Turno Di Lavoro

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo1	Volume atteso
CENTRO	E	100
IMPIEGATO	E	2000
IMPIEGO	E	20000
I-I	R	20000
C-I	R	20000
IMPIEGO ATTUALE	E	2000
IMPIEGO PASSATO	E	18000
TURNO DI LAVORO	E	24000
POSSIEDE	R	24000
FILM	E	500
CLASSICI	E	150
NUOVE USCITE	E	250
SETTORE	E	400
FILM EFFETTIVO	E	100000
F-F	R	100000
S-F	R	100000
CLIENTE	E	30000
NOLEGGIA	R	120000

Tavola delle operazioni

1 Indicare con E le entità, con R le relazioni

Cod.	Descrizione	Frequenza attesa
1	Spostare un impiegato da un centro ad un altro.	1/mese
2	Visualizzare il report mensile sulle ore lavorate e I luoghi di lavoro di un impiegato	200/mese
3	Per ogni impiegato stampare report annuali sulle ore lavorate e i luoghi di lavoro.	1/anno
4	Registrare un nuovo impiegato	2/mese
5	Inserire turno di lavoro.	200/mese
6	Registrare un nuovo cliente	20/giorno
7	Registrare un noleggio.	100/giorno
8	Stampare per ogni centro di servizio i titoli associati ad un noleggio scaduto e i clienti che hanno effettuato tali noleggi.	20/mese
9	Registrare la restituzione di un noleggio	30/giorno

Costo delle operazioni

Codice 1: L'operazione comprende 2 fasi, una prima fase di modifica dell'impiego attuale che diviene impiego passato e una fase di scrittura di un nuovo impiego attuale. Quindi un accesso in lettura all'entità impiegato un accesso in lettura alla relazione I-I e un accesso in scrittura all'entità impiego attuale per rimuovere il record relativo all'attuale impiego e un accesso in scrittura all'entità impiego passato per registrare il record rimosso e poi un accesso in scrittura alla relazione I-I e all'entità impiego attuale per registrare il nuovo impiego e quindi un totale di 2 accessi in lettura e 4 in scrittura da eseguire 1 volta al mese quindi 2 + 4*2 = 10 accessi al mese

Codice 2: L'operazione richiede un accesso in lettura all'entità impiegato poi in media 1 acceso in lettura all'entità turno di lavoro e alla relazione possiede per individuare le ore lavorate e in media 1 accesso alla relazione I-I e all'entità impiego e alla relazione C-I e all'entità Centro per determinare I luoghi di lavoro quindi in media 9 accessi al mese e considerando il volume previsto degli impiegati pari a 2000 e supponendo che tale operazione venga eseguita per ogni impiegato abbiamo un totale di 18000 accessi al mese

Codice 3: Richiede di stampare report annuali quindi report per ogni mese, dunque un totale di accessi pari a 216000/anno.

Codice 4: Si richiede un accesso in scrittura ad Impiegato, un accesso in scrittura a I-I poi 1 accesso in scrittura a Impiego e un accesso in scrittura a C-I e un accesso in scrittura a Impiego Attuale e poi un accesso in scrittura alla relazione Possiede e uno all'entità turno di lavoro.Totale 14 accessi e considerando la frequenza stimata si hanno 14*2/mese= 28 accessi/mese

Codice 5: Si richiede 1 accesso in lettura all'entità Impiegato, 1 accesso in lettura alla relazione Possiede e un accesso in lettura e uno in scrittura all'entità turno di lavoro per modificare il turno di lavoro attuale. Poi un accesso in scrittura all'entità turno di lavoro e alla relazione Possiede per inserire il nuovo turno di lavoro. Dunque 9 accessi che considerate la frequenza prevista sono 1800/mese

Codice 6: richiede un solo accesso in scrittura a Cliente. Considerata la frequenza si hanno un totale di 40 accessi/giorno

Codice 7: Si richiede un accesso in lettura all'entità cliente per individuare il cliente che ha effettuato il noleggio, un accesso all'entità noleggio in scrittura, un accesso in lettura e uno in scrittura all'entità film effettivo per aggiornare il numero di copie del film noleggiato . Totale 6 accessi e considerata la frequenza si hanno 6*100=600 accessi/giorno

Codice 8: Data la frequenza stimata si suppone che la suddetta operazione venga eseguita 2 volte al mese per ciascun centro di servizio che dato il volume di dati attesi porta ad una frequenza di 200/mese. Considerando che un Centro ha mediamente 4 settori (fare riferimento sempre al volume atteso stimato) e che ciascun settore ha mediamente 250 film e che un film venga noleggiato mediamente 4 volte abbiamo per un singolo settore 1 accesso all'entità settore, 250 accessi all'entità Film Effettivo e alla relazione S-I e per ciascun film 4 accessi alla relazione noleggia e alla entità cliente per individuare I noleggi scaduti e I client che li hanno effettuati. Dunque avremo 1+250+250+8*250 = 901 accessi per un singolo settore, considerata la media di 4 settori abbiamo 3604 accessi e vista la frequenza prevista abbiamo 72080 accessi al mese. Si noti che per effettuare la stima si è supposto che tutti I film presso il settore siano stati noleggiati da 4 clienti distinti e che tutti questi 4 clienti non abbiano rispettato I termini di consegna, si noti infatti che se l'accesso alla relazione noleggia determina che la data di scadenza non è antecedente alla data dell'esecuzione dell'operazione allora l'accesso all'entità cliente non si ritiene più necessario. Pertanto questa stima rappresenta un upperbound del costo effettivo dell'operazione.

Codice 9: Si richiede un accesso in scrittura alla relazione Noleggia per rimuovere il film noleggiato e 1 accesso in lettura all'entità Cliente per individuare il cliente che ha noleggiato il film e 1 accesso in lettura all'entità film effettivo per individuare il film e un accesso in scrittura all'entità film effettivo per aggiornare il numero di copie. Totale 6*30 = 180 accessi/giorno

Ristrutturazione dello schema E-R

Al fine di risolvere le generalizzazioni presenti per l'associazione Impiego che si generalizza in Impiego attuale e Impiego passato si procede come segue: dal momento che le operazioni che coinvolgono l'entità impiego tipicamente accedono sia all'entità Impiego attuale che all'entità impiego passato allora si è deciso di introdurre un attributo aggiuntivo che permette di discriminare se un impiego è attualmente in corso o meno ovvero l'attributo data_fine e assumendo che se tale attributo è assente allora l'impiego è attualmente in corso (si noti la cardinalità 0,1). Per tale risoluzione vedere lo schema parziale in figura 1. Per eliminare la generalizzazione di Film in Classici e Nuove Uscite si procede anche qui all'accorpamento delle entità figlie nell'entità padre introducendo un attributo tipo il cui valore è uno tra {classico,nuovo,normale}. Nessuna delle operazioni previste si interessa di discriminare tra le due tipologie di film pertanto gli accessi a Film sono congiunti sia a Classici che Nuove uscite e questo giustifica la scelta presa (vedere figura 3 per uno schema di tale scelta). Infine risolviamo l'attributo composto Orario in Turno di Lavoro come illustrato in figura 2.

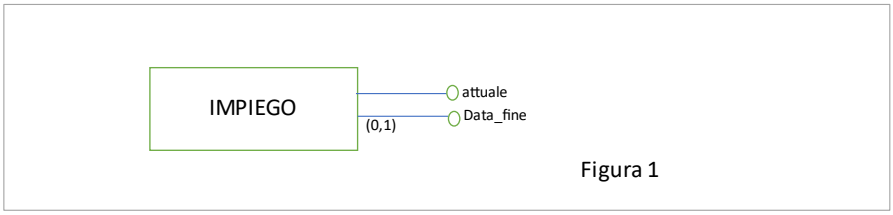


Figura 1

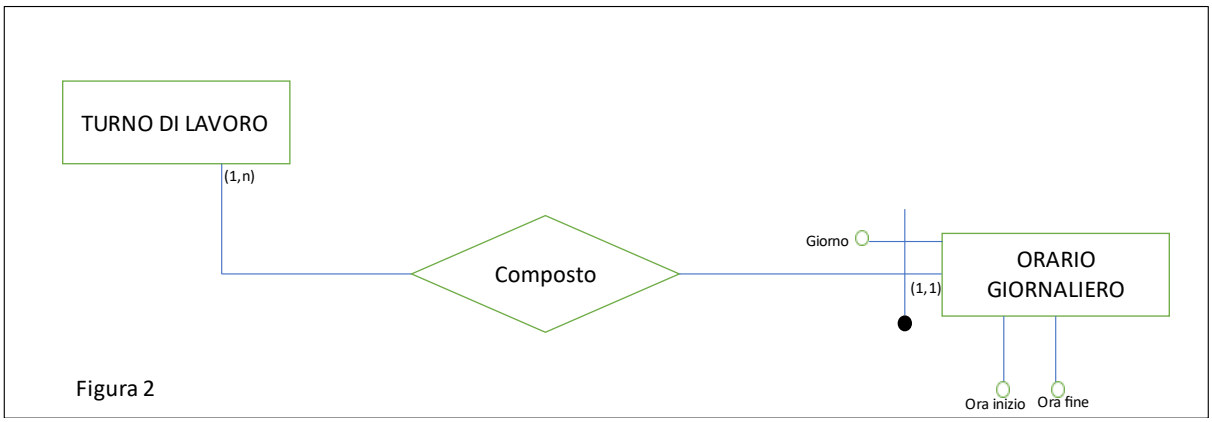


Figura 2

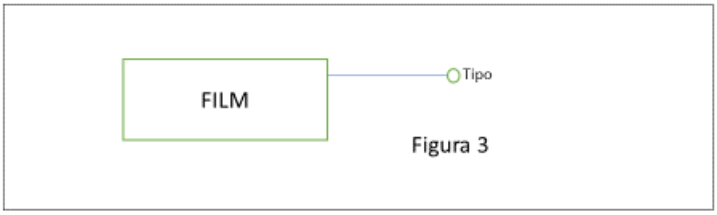
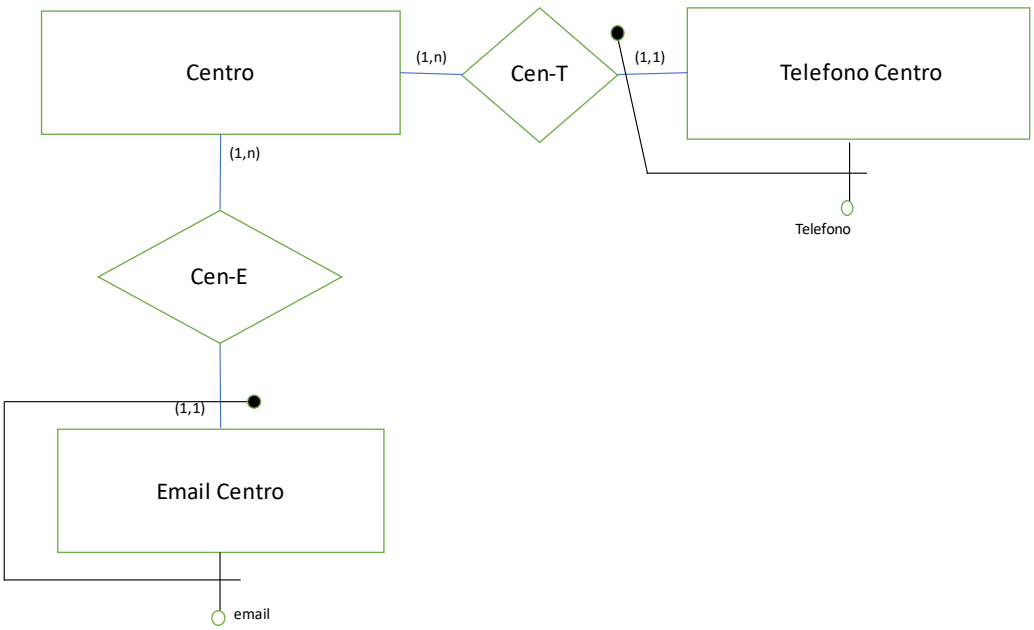
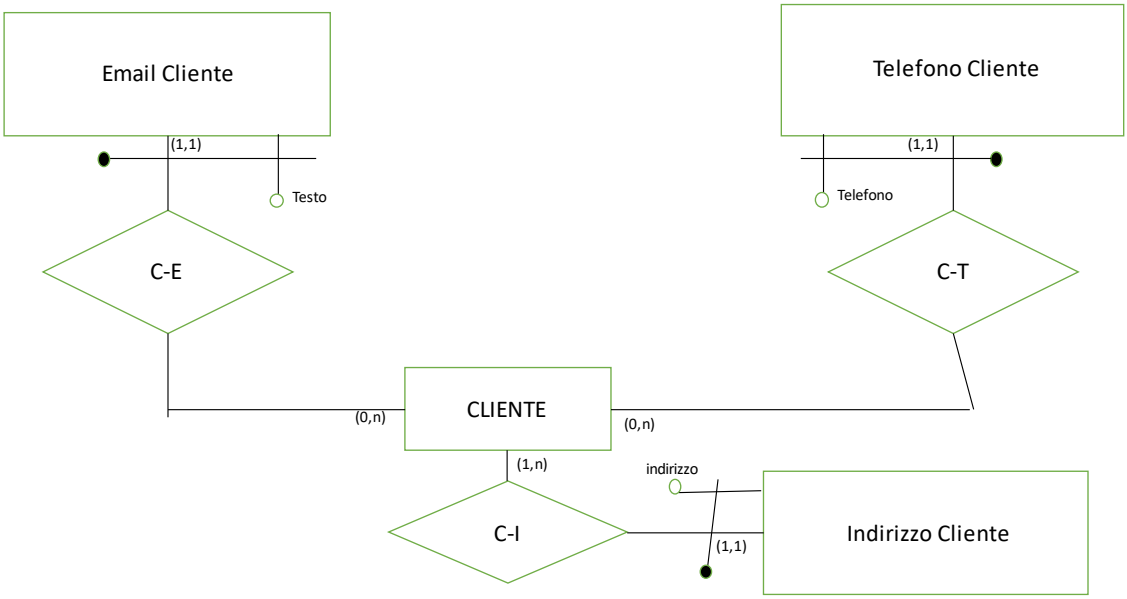
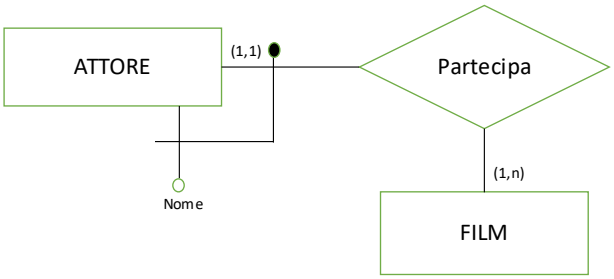


Figura 3

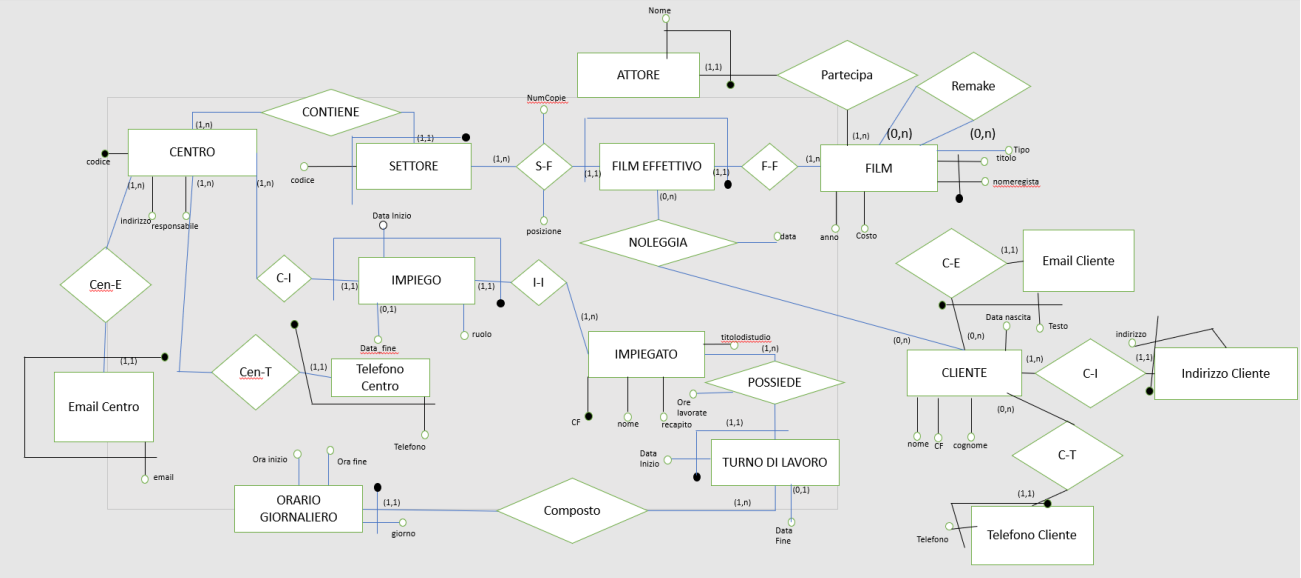
Si mostra adesso come sono stati risolti gli attributi multipli in entità deboli.




Email Cliente
(1,1)
Testo
C-E
Telefono Cliente
(1,1)
Telefono
C-T
CLIENTE
(0,n)
(0,n)
(1,n)
indirizzo
C-I
(1,1)
Indirizzo Cliente


ATTORE
(1,1)
Partecipa
Nome
(1,n)
FILM

Si propone di seguito lo schema E-R ristrutturato con i nuovi identificatori per ciascuna entità.



Trasformazione di attributi e identificatori

Traduzione di entità e associazioni

IMPIEGATO(CF,Nome,Recapito,TitoloDiStudio)

IMPIEGO(Impiegato,DataInizio,Centro,DataFine*,ruolo)

CENTRO(Codice,Indirizzo,Responsabile)

EMAILCENTRO(Centro,Email)

TELEFONOCENTRO(Centro,Telefono)

TURNO DI LAVORO(Impiegato,DataInizio,DataFine*)

ORARIO GIORNALIERO(Impiegato,TurnoDataInizio,Giorno,OraInizio,OraFine)

SETTORE(Codice,Centro)

FILM(Titolo,Regista,Anno,Tipo,Costo)

ATTORE(FilmTitolo,FilmRegista,Nome)

FILME FFETTIVO(FilmTitolo,FilmRegista,SettoreCodice,SettoreCentro,numCopie,posizione)

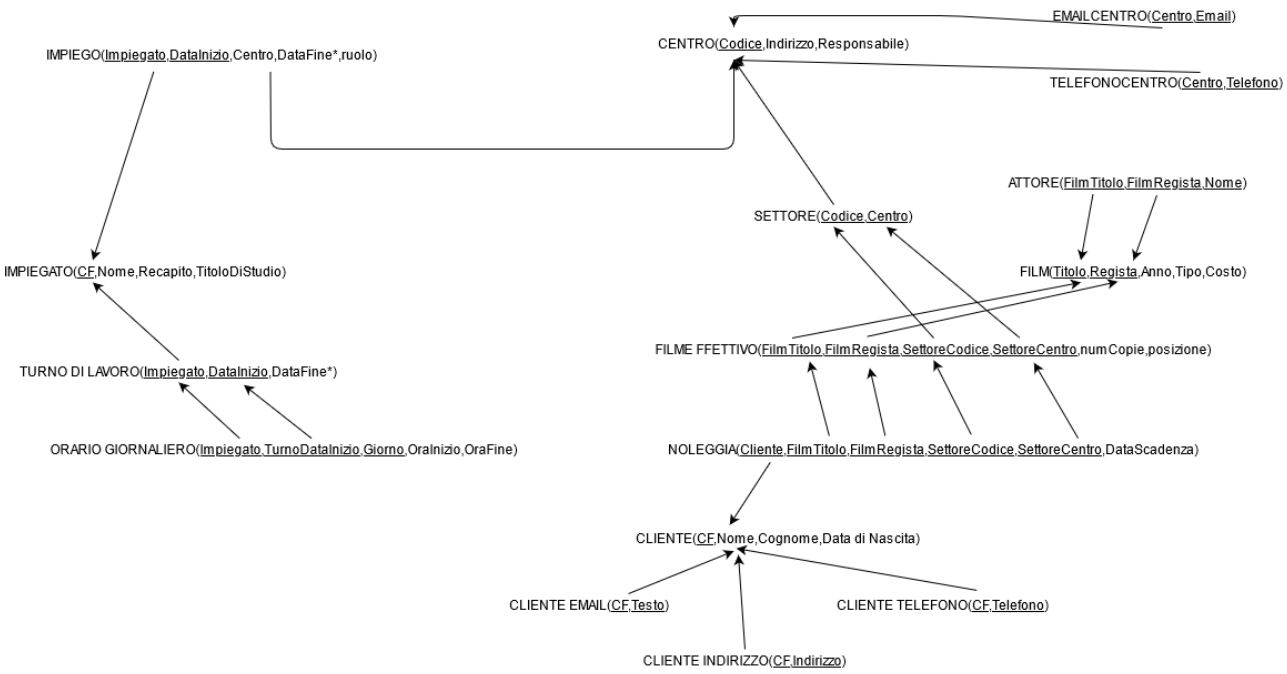
NOLEGGIA(Cliente,FilmTitolo,FilmRegista,SettoreCodice,SettoreCentro,DataScadenza)

CLIENTE(CF,Nome,Cognome,Data di Nascita)

CLIENTE EMAIL(CF,Testo)

CLIENTE TELEFONO(CF,Telefono)

CLIENTE INDIRIZZO(CF,Indirizzo)



Normalizzazione del modello relazionale

Il modello relazionale presentato è in forma normale di Boyce e Codd (in seguito BCNF). Per mostrarlo è sufficiente analizzare la struttura delle varie relazioni andando ad analizzare se siano presenti o meno dipendenze funzionali non banali. Per dipendenza non banale si intende una dipendenza funzionale che non coinvolge la chiave primaria.

Partiamo dalla relazione di Impiegato. L'attributo scelto come chiave primaria è il codice fiscale (CF) e non è difficile convincersi che non ci sono dipendenze funzionali che non coinvolgano la chiave primaria. Tutti gli attributi presenti si riferiscono infatti all'anagrafica dell'impiegato che per definizione stessa del codice fiscale dipendono solo da quest'ultimo.

Passiamo alla relazione Impiego dove la chiave primaria scelta è la chiave composta (Impiegato,DataInizio). In questo caso ci sono le seguenti dipendenze funzionali: Impiegato,DataInizio -> Centro,Ruolo,DataFine. Difatti per tutti gli attributi non chiave di questa relazione non è possibile individuare una dipendenza funzionale non banale, basti ad esempio pensare al significato assegnato all'attributo Centro in relazione al minimondo di riferimento che rappresenta un centro della catena di servizio presso il quale sono tipicamente impiegati più di un impiegato. In generale dunque non vi sono dipendenze funzionali al di fuori di quella "banale" strettamente legata al ruolo stesso di chiave primaria. Infatti il Ruolo non è un attributo strettamente legato ad un singolo impiegato (è possibile che in centri diversi o nello stesso centro vi siano più impiegati che prestano servizio per lo stesso ruolo). Per la DataFine valgono ragionamenti analoghi. Si noti che la scelta di una chiave primaria più piccola (ad esempio solo l'attributo Impiegato) avrebbe portato ad una anomalia di inserimento (si può registrare un solo impiego per ciascun impiegato). E' interessante osservare come la coppia (Impiegato,DataInizio) sia sufficiente a consentire l'inserimento di più impieghi per uno stesso impiegato (anche riferiti allo stesso centro) dal momento che non è mai possibile che un impiegato inizi due impieghi presso un centro nella stessa data.

Passiamo alla relazione Centro. La chiave primaria è l'attributo Codice che rappresenta il codice univoco identificativo del centro. Non ci sono dipendenze funzionali al di fuori di quella banale.

Passiamo alle relazioni EMAILCENTRO e TELEFONOCENTRO. In entrambe le relazioni sono presenti esclusivamente 2 attributi che contribuiscono a definire la chiave primaria di conseguenza queste relazioni sono chiaramente in BCNF. Stesso ragionamento vale per le relazioni Settore, Cliente Telefono, Cliente Email e Cliente Indirizzo.

Passiamo alla relazione TURNO DI LAVORO. La chiave primaria è la chiave composta (Impiegato,DataInizio). Anche qui non ci sono dipendenze funzionali al di fuori di quella banale. L'unico attributo che non compone la chiave primaria è DataFine che rappresenta la data di fine lavoro per un determinato impiegato e di conseguenza può essere comune a più impiegati. In generale vale la dipendenza (Impiegato)->DataFine che comunque dipendendo da parte della chiave primaria rientra tra le dipendenze banali. Questa dipendenza vale perché dato un impiegato allora

questo non può mai terminare due distinti turni di lavoro nella stessa data. Ci si potrebbe chiedere perché non si sia scelta una chiave primaria più piccola, per rispondere basti osservare che una chiave composta dal solo attributo Impiegato non sarebbe sufficiente ad inserire più turni di lavoro per lo stesso impiegato (anomalia di inserimento).

Per la relazione ORARIO GIORNALIERO la chiave primaria è la chiave composta (Impiegato,TurnoDataInizio,Giorno). La relazione è in BCNF infatti non è vero che OraInizio e/o OraFine dipendono solo da Giorno come potrebbe invece sembrare in quanto più impiegati possono avere uno stesso orario. Sembrerebbe dunque che (Impiegato,Giorno) -> (OraInizio,OraFine) ma non basta perché si vuole poter assegnare ad un impiegato turni di lavoro diversi e con una soluzione simile questo non sarebbe possibile. Di fatti la relazione nasce dall'esigenza di rappresentare orari di inizio e fine di lavoro per ogni giorno relativamente ad un turno di lavoro.

La relazione FILM ha come chiave primaria la chiave composta (Titolo,Regista). Anche qui non vi sono dipendenze non banali dal momento che gli altri attributi restanti fanno parte delle specifiche stesse del film che di conseguenza sono dipendenti dalla chiave di quest'ultimo. A causa di possibili omonimie nel titolo e data la possibilità che un regista giri più di un film diverso non è possibile utilizzare una chiave primaria più piccola.

La relazione Attore è in BCNF in quanto ha soli 3 attributi che compongono la chiave primaria.

La relazione FILM EFFETTIVO è in BCNF in quanto sia l'attributo numCopie che l'attributo posizione sono relative al film specifico registrato all'interno di un certo settore in un dato centro. Non è possibile individuare la posizione di un film o il numero di copie rimanenti per un certo settore senza conoscere tutti gli attributi della chiave primaria.

La relazione NOLEGGIA è in BCNF infatti c'è un unico attributo oltre alla chiave primaria. Ci si potrebbe chiedere se la chiave primaria implica la DataScadenza. Questo è banalmente vero poiché la chiave primaria rappresenta univocamente l'ordine combinando insieme il film effettivo e il cliente che lo ha noleggiato.

La relazione Cliente è in BCNF infatti la chiave primaria è il Codice Fiscale del Cliente e la restante parte degli attributi come per l'entità Impiegato rappresentano l'anagrafica del cliente quindi dipendono necessariamente unicamente dal codice fiscal quindi non vi sono dipendenze non banali.

Concludendo dal momento che tutte le entità che lo compongono sono in BCNF anche lo schema proposto è in BCNF. Si noti come la forma normale di Boyce e Codd implichi le altre forme normali (1NF,2NF,3NF)

5. Progettazione fisica

Utenti e privilegi

All'interno dell'applicazione sono stati previsti 4 tipi di utente. A ciascun utente è stato assegnato unicamente il privilegio di eseguire delle store procedure. Di seguito descrivo i vari utenti e riporto una lista di store procedure per le quali hanno privilegio di esecuzione.

Utente	**Descrizione**	**Privilegi**
login	Questo utente è stato realizzato al fine di avere una utenza base con la quale il client si collega al dbms, pertanto tale utente si limita ad eseguire la procedura di login.	login
amministratore	Si è deciso di introdurre questa utenza anche se non esplicitamente richiesta dalla specifica per individuare una figura a cui affidare la responsabilità di gestire la struttura della catena (quindi i centri di servizio e i settori che compongono tali centri) nonché la gestione del personale manageriale (i manager). La figura dell'amministratore permette di centralizzare queste responsabilità in pochi individui piuttosto che assegnarle a ciascun manager. Pertanto un amministratore è prima di tutto un manager e di fatti ne eredita tutti i privilegi.	creaUtente, inserisciCentro, inserisciSettore, inserisciFilmCatalogo, rimuoviManager, rimuoviFilm, inserisciTelefonoCentro, inserisciEmailCentro

impiegato	Questa utenza è utilizzata dagli impiegati registrati presso la catena di video noleggio. Ciascun impiegato può gestire le informazioni relative ai noleggi (ad esempio registrare un nuovo noleggio o la terminazione di uno). Può inoltre gestire l'anagrafica dei clienti e gestire il posizionamento dei film all'interno del centro presso cui afferisce. Infine l'impiegato può timbrare il cartellino per la registrazione delle ore di lavoro.	registraCliente, noleggiaFilm, timbra, inserisciFilmSettore, restituisciFilm, stampaFilmScaduti, visualizzaTurno, visualizzaFilmCentro, rimuoviFilmSettore, trovaCliente, visualizzaNoleggi, inserisciEmailCliente, inserisciTelefonoCliente, inserisciIndirizzoCliente, rimuoviCliente, aggiornaDisponibilita, visualizzaFilmMancanti
manager	Questa utenza è utilizzata dai manager della catena di noleggio. A questi ultimi è data la possibilità di gestire gli impiegati della catena (ad esempio eseguendo nuove assunzioni o cambiando il centro di afferenza per un impiegato).	registraImpiego, registraImpiegato, creaUtente, reportMensile, inserisciTurnoDiLavoro, visualizzaImpiegati, visualizzaCentri, reportAnnuale, rimuoviImpiegato

Strutture di memorizzazione

Tabella <attore>		
Attributo	**Tipo di dato**	**Attributi2**
FilmTitolo	VARCHAR(60)	PK, NN
FilmRegista	VARCHAR(45)	PK, NN
Nome	VARCHAR(45)	PK, NN

Tabella <centro>		
Attributo	**Tipo di dato**	**Attributi2**
Codice	INT	PK, NN, AI
Indirizzo	VARCHAR(60)	NN
Responsabile	VARCHAR(45)	NN

Tabella <cliente>		
Attributo	**Tipo di dato**	**Attributi2**
CF	VARCHAR(45)	PK, NN, UQ
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
Indirizzo	VARCHAR(60)	NN

Tabella <emailcentro>		
Attributo	**Tipo di dato**	**Attributi2**
Centro	INT	PK, NN
Testo	VARCHAR(45)	PK, NN

Tabella <film>		
Attributo	**Tipo di dato**	**Attributi2**

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

Titolo	VARCHAR(60)	PK, NN
Regista	VARCHAR(45)	PK, NN
Tipo	ENUM("other","nuovo", "classico")	NN
AnnoPubblicazione	YEAR	NN
Costo	DECIMAL	NN

Tabella <filmeffettivo>		
Attributo	**Tipo di dato**	**Attributi2**
FilmTitolo	VARCHAR(60)	PK, NN
FilmRegista	VARCHAR(45)	PK, NN
SettoreCodice	INT	PK, NN
SettoreCentro	INT	PK, NN
numCopie	INT	NN
posizione	VARCHAR(45)	NN

Tabella <impiegato>		
Attributo	**Tipo di dato**	**Attributi2**
CF	VARCHAR(45)	PK, NN
Nome	VARCHAR(45)	NN
Recapito	VARCHAR(45)	NN
TitoloDiStudio	VARCHAR(60)	NN
username	VARCHAR(45)	NN, UQ
Cognome	VARCHAR(45)	NN

Tabella <impiego>		
Attributo	**Tipo di dato**	**Attributi2**
Impiegato	VARCHAR(45)	PK, NN

DataInizio	DATE	PK, NN
DataFine	DATE	
Ruolo	VARCHAR(45)	NN
Centro	INT	NN

Tabella <login>		
Attributo	**Tipo di dato**	**Attributi2**
username	VARCHAR(45)	PK, NN
passw	VARCHAR(33)	NN
ruolo	ENUM("impiegato", "manager", "amministratore")	NN

Tabella <noleggia>		
Attributo	**Tipo di dato**	**Attributi2**
Cliente	VARCHAR(45)	PK, NN
FilmTitolo	VARCHAR(60)	PK, NN
FilmRegista	VARCHAR(45)	PK, NN
SettoreCodice	INT	NN
SettoreCentro	INT	NN
DataScadenza	DATE	NN

Tabella <orariogiornaliero>		
Attributo	**Tipo di dato**	**Attributi2**
Giorno	ENUM('Monday','Tuesday', 'Wednesday','Thursday',	PK, NN

	'Friday', 'Saturday')	
TurnoDiLavoroI	VARCHAR(45)	PK, NN
TurnoDiLavoroData	DATE	PK, NN
OraInizio	TIME	NN
OraFine	TIME	NN

Tabella <settore>		
Attributo	**Tipo di dato**	**Attributi2**
Codice	INT	PK, NN
Centro	INT	PK, NN

Tabella <telefonocentro>		
Attributo	**Tipo di dato**	**Attributi2**
Centro	INT	PK, NN
Numero	VARCHAR(15)	PK, NN

Tabella <turno di lavoro>		
Attributo	**Tipo di dato**	**Attributi2**
DataInizio	DATE	PK, NN
Impiegato	VARCHAR(45)	PK, NN
DataFine	DATE	

Tabella <reportOre>		
Attributo	**Tipo di dato**	**Attributi2**
DataInizio	DATE	PK,NN
Impiegato	VARCHAR(45)	PK,NN
Ore	Decimal	NN
DataFine	DATE	

Tabella <indirizzo cliente>		
Attributo	**Tipo di dato**	**Attributi2**
Cliente	VARCHAR(45)	PK,NN
Indirizzo	VARCHAR(60)	PK,NN

Tabella <emailcliente>		
Attributo	**Tipo di dato**	**Attributi2**
Cliente	VARCHAR(45)	PK,NN
Email	VARCHAR(45)	PK,NN

Tabella <telefonocliente>		
Attributo	**Tipo di dato**	**Attributi2**
Cliente	VARCHAR(45)	PK,NN
Numero	VARCHAR(15)	PK,NN

Indici

Tabella <attore>	
Indice <PRIMARY>	**Tipo3:**
FilmTitolo,FilmRegista,Nome	PR

Tabella <attore>	
Indice <Film_idx>	**Tipo3:**
FilmTitolo, FilmRegista	IDX

Tabella <centro>

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

Indice <PRIMARY>	**Tipo3:**
Codice	PR

Tabella <cliente>	
Indice <PRIMARY>	**Tipo3:**
CF	PR

Tabella <emailcentro>	
Indice <PRIMARY>	**Tipo3:**
Centro,email	PR

Tabella <emailcliente>	
Indice <PRIMARY>	**Tipo3:**
Cliente, Email	PR

Tabella <film>	
Indice <PRIMARY>	**Tipo3:**
FilmTitolo, FilmRegista	PR

Tabella <filmeffettivo>	
Indice <PRIMARY>	**Tipo3:**
FilmTitolo, FilmRegista, SettoreCodice, SettoreCentro	PR

Tabella <filmeffettivo>	
Indice <Settore_idx>	**Tipo3:**
SettoreCodice, SettoreCentro	INDEX

Tabella <impiegato>	
Indice <PRIMARY>	**Tipo3:**
CF	PR

Tabella <impiegato>	
Indice <username_UNIQUE>	**Tipo3:**
username	UQ

Tabella <impiego>	
Indice <PRIMARY>	**Tipo3:**
Impiegato, DataInizio	PR

Tabella <impiego>	
Indice <Centro_idx>	**Tipo3:**
Centro	IDX

Tabella <login>	
Indice <PRIMARY>	**Tipo3:**
username	PR

Tabella <noleggia >	
Indice <PRIMARY>	**Tipo3:**
Cliente,FilmTitolo,FilmRegista,SettoreCodice,SettoreCentro	PR

Tabella <noleggia>	
Indice <FilmNoleggiato_idx>	**Tipo3:**
FilmTitolo,FilmRegista,SettoreCodice,SettoreCentro	IDX

Tabella <noleggia>	
Indice <Cliente_idx>	**Tipo3:**
Cliente	IDX

Tabella <orariogiornaliero>	
Indice <PRIMARY>	**Tipo3:**
Giorno, TurnoDiLavoroI, TurnoDiLavoroData	PR

Tabella <orariogiornaliero>	
Indice <turnodilavoro_idx>	**Tipo3:**
TurnoDiLavoroI, TurnoDiLavoroData	IDX

Tabella <remake>	
Indice <PRIMARY>	**Tipo3:**
FilmTitolo1,FilmTitolo2,FilmRegista1,FilmRegista2	PR

Tabella <remake>	
Indice <FilmRiferito2_idx>	**Tipo3:**
FilmTitolo2, FilmRegista2	IDX

Tabella <reportOre>	
Indice <PRIMARY>	**Tipo3:**
DataInizio, Impiegato	PR

Tabella <settore>	
Indice <PRIMARY>	**Tipo3:**
Codice, Centro	PR

Tabella <settore>	
Indice <Centro_idx>	**Tipo3:**
Centro	IDX

Tabella <telefonocentro>	
Indice <PRIMARY>	**Tipo3:**
Centro, Numero	PR

Tabella <telefonocliente>	
Indice <PRIMARY>	**Tipo3:**
Cliente, Numero	PR

Tabella <turnodilavoro>	
Indice <PRIMARY>	**Tipo3:**
DataInizio, Impiegato	PR

Tabella <turnodilavoro>	
Indice <ImpiegoLavoro_idx>	**Tipo3:**
Impiegato	INDEX

Tabella <indirizzo cliente>	
Indice <PRIMARY>	**Tipo3:**
Cliente, Indirizzo	PR

Trigger

Al fine di garantire la regola aziendale relativa ai valori degli attributi OraInizio e OraFine della tabella orariogiornaliero è stato implementato un trigger di tipo before insert con il compito di analizzare I valori degli attributi per la nuova tupla inserita e assicura che venga rispettato il vincolo imposto. In caso contrario viene sollevata una eccezione ed il messaggio associato all'eccezione indica esplicitamente il motivo dell'errore.

Trigger:

```
DELIMITER $$
USE `progetto1`$$

DROP TRIGGER IF EXISTS `progetto1`.`END_LATE_START` $$
USE `progetto1`$$
CREATE DEFINER = CURRENT_USER TRIGGER END_LATE_START BEFORE INSERT
ON `orariogiornaliero` FOR EACH ROW
BEGIN
   if(new.OraFine<=new.OraInizio)
   then
      signal sqlstate '45001' set message_text="L'orario di fine deve essere successiva di almeno un
minuto a quella di inizio";
   end if;
END$$
DELIMITER ;
```

Un altro trigger è stato realizzato per assicurare che la data di fine noleggio sia successiva alla data corrente in modo tale da impedire la registrazione di un noleggio già scaduto

Trigger:

```
DELIMITER $$

USE `progetto1`$$
DROP TRIGGER IF EXISTS `progetto1`.`noleggia_BEFORE_INSERT` $$
USE `progetto1`$$
CREATE DEFINER = CURRENT_USER TRIGGER `progetto1`.`noleggia_BEFORE_INSERT`
BEFORE INSERT ON `noleggia` FOR EACH ROW
BEGIN
   if(NEW.DataScadenza = '' or NEW.DataScadenza < curdate())
   then
      signal sqlstate '45001' set message_text="La data di scadenza deve essere successiva alla data
corrente";
   end if;
END$$

DELIMITER ;
```

Eventi

All'interno dell'applicazione è stato realizzato un unico evento che viene schedulato ogni 24 ore (una volta al giorno dunque) realizzato per assolvere al seguente compito: aggiornare correttamente il numero di ore lavorate da un impiegato andando ad aggiungere le ore lavorate nella giornata odierna e resettare tutti i campi useful presenti al solo fine di gestire e coordinare le operazioni degli utenti. Ecco di seguito il codice dell'evento

```
SET GLOBAL event_scheduler=ON;
delimiter |
CREATE EVENT `progetto1`.`ResettaGiorni`
ON SCHEDULE EVERY 24 DAY_HOUR

DO BEGIN
declare done int default false;
declare var_Date date;
declare var_Impiegato varchar(45);
declare var_Inizio time;
declare var_Fine time;
declare cur cursor for
select `Impiegato` from `turno di lavoro` where `DataFine` is null;
```

```
declare continue handler for not found set done = true;
set transaction isolation level serializable;
start transaction;
if object_id('timbratura') is not null
then
open cur;
read_loop: loop
   fetch cur into var_Impiegato;
   if done
      then leave read_loop;
   end if;
   select timeFine,timeInizio from `timbratura` where Impiegato=var_Impiegato into
var_Fine,var_Inizio;
   if (var_Fine is not null and var_Inizio is not null)
   then
      update `reportOre`
      set Ore=Ore + HOUR(TIMEDIFF(var_Fine,var_Inizio))+
MINUTE(TIMEDIFF(var_Fine,var_Inizio))/100
      where Impiegato=var_Impiegato and `reportOre`.`DataFine` is null;
      select DataInizio
      from `reportOre`
      where Impiegato=var_Impiegato and `reportOre`.`DataFine` is null into var_Date;
      if (MONTH(var_Date)<MONTH(curdate()))
      then
         update `reportOre`
         set DataFine=curdate()
         where Impiegato=var_Impiegato and `reportOre`.`DataFine` is null;
         insert into `reportOre` (Impiegato,DataInizio,DataFine,Ore) values
(var_Impiegato,curdate(),NULL,0);
      end if;
   end if;
   end loop;
   close cur;
   drop table `timbratura`;
end if;
END
```

Viste

Non sono state previste viste.

Stored Procedures e transazioni

Al fine di inserire tutto il codice sql all'interno del dbms sono state realizzate un certo numero di Stored Procedure semplici per effettuare anche singole operazioni. Questo permette di limitare a determinati utenti la possibilità di poter eseguire tali operazioni e svincola il client dalla conoscenza interna della struttura del database.

inserisciSettore

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`inserisciSettore`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserisciSettore` (in var_S INT, in var_C INT)
BEGIN
   insert into `settore` values(var_S,var_C);
END$$

DELIMITER ;
```

Questa Stored Procedure è stata realizzata per fornire un punto di accesso all'applicazione client per inserire un nuovo settore all'interno di un centro dato. Per il settore viene data all'utente la possibilità di assegnare un numero seriale.

creaUtente

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`creaUtente`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `creaUtente` (in var_Username VARCHAR(45), in var_Password
VARCHAR(45), in var_Role ENUM("impiegato","manager","amministratore"))
BEGIN
   insert into `login` values (var_Username,md5(var_Password),var_Role);
END$$

DELIMITER ;
```

Questa Stored Procedure offre la possibilità di creare un nuovo utente all'interno del Sistema assegnando uno Username ed una Password all'utente. Va inoltre specificato il ruolo di quest'ultimo attraverso una enum (a ruoli diversi corrispondono privilegi diversi).

visualizzaCentri

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`visualizzaCentri`;

DELIMITER $$
```

```
USE `progetto1`$$
CREATE PROCEDURE `visualizzaCentri` ()
BEGIN
   select * from `centro`;
END$$

DELIMITER ;
```

Questa Stored Procedure è stata realizzata per permettere un punto di accesso al client per la visualizzazione di tutti i centri di servizio registrati nel Sistema.

visualizzaImpiegati

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`visualizzaImpiegati`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `visualizzaImpiegati` ()
BEGIN
   select * from `impiegato`;
END$$

DELIMITER ;
```

Questa Stored Procedure permette di visualizzare l'intero organico registrato presso il Sistema per la catena di servizio.

inserireFilmCatalogo

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`inserireFilmCatalogo`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserireFilmCatalogo` (in var_Titolo VARCHAR(60), in var_Regista
VARCHAR(45), in var_Tipo ENUM("other","nuovo","classico"),in var_Anno DATE, in
var_Attori VARCHAR(1000),in var_Costo DOUBLE,in var_RemakeTitoli VARCHAR(600), in
var_RemakeRegista VARCHAR(450))
BEGIN
   declare var_Actor VARCHAR(45);
   declare var_Titolo1 VARCHAR(45);
   declare var_Regista1 VARCHAR(45);
```

```
    declare contA int default 1;
        declare exit handler for sqlexception
      begin
        rollback;
set autocommit=1;
        resignal;
      end;
   set autocommit=0;
   set transaction isolation level read uncommitted;
   start transaction;
      insert into `film` values (var_Titolo,var_Regista,var_Tipo,YEAR(var_Anno),var_Costo);
      insert_loop: loop
        select SPLIT_STR(var_Attori,";",contA) into var_Actor;
        if(var_Actor='')
          then leave insert_loop;
        end if;
        insert into `attore` values(var_Titolo,var_Regista,var_Actor);
        set contA=contA+1;
      end loop;
      set contA=1;
      insert2_loop:loop
        select SPLIT_STR(var_RemakeTitoli,";",contA) into var_Titolo1;
        select SPLIT_STR(var_RemakeRegista,";",contA) into var_Regista1;
        if(var_Titolo1='' or var_Regista1='')
          then leave insert2_loop;
        end if;
        insert into `remake` values(var_Titolo,var_Regista,var_Titolo1,var_Regista1);
        set contA=contA+1;
      end loop;
      commit;
   set autocommit=1;
END$$

DELIMITER ;
```

Questa Stored Procedure è stata realizzata per consentire l'inserimento di un nuovo film nel catalogo della catena di servizio. Si noti l'utilizzo della transazione per consentire l'inserimento di tutte le informazioni (lista di attori, film di cui è remake e così via) come una operazione atomica. Le variabili di ingress var_Attori e var_RemakeTitoli sono delle stringe di nomi di attori o di film separati da ";". Si noti che il simbolo ";" è ragionevolmente non presente in nomi di attori o di film tuttavia la stored procedure non è in grado di distinguere se un ";" è un simbolo di delimitatore di nome o se fa parte di un nome stesso data l'assenza di una escape sequence per rendere la cosa più efficiente dato l'utilizzo per la quale è predisposta e la bassa probabilità di presenza di nomi che contengano ";". Dal momento che questa procedura non esegue operazioni di lettura (se non delle

select sul valore ritornato dalla funzione SPLIT_STR) si è scelto il livello di isolamento più basso possibile.

inserisciCentro

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`inserisciCentro`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserisciCentro` (in var_Indirizzo varchar(60),in var_Resp
varchar(45), in var_Telefono varchar(100), in var_Email varchar(450) )
BEGIN
   declare var_Centro int;
   declare var_A int;
   declare var_S varchar(45);
        declare exit handler for sqlexception
      begin
         rollback;
        set autocommit=1;
        resignal;
      end;
   set autocommit=0;
   set transaction isolation level read uncommitted;
   start transaction;
   if(var_Email ='' or var_Telefono ='')
   then
      signal sqlstate '45001' set message_text="Devi inserire almeno una email e un telefono";
   end if;
   insert into `centro`(Indirizzo,Responsabile)
   values (var_Indirizzo,var_Resp);
   set var_Centro=last_insert_id();
   set var_A = 0;
   insert_loop1: loop
   select SPLIT_STR(var_Email,';',var_A) into var_S;
   if(var_S= '')
      then
      leave insert_loop1;
   end if;
   insert into `emailcentro` (Centro,Email)
   values (var_Centro,var_S);
   set var_A = var_A+1;
   end loop;
   set var_A = 0;
   insert_loop2: loop
   select SPLIT_STR(var_Telefono,';',var_A) into var_S;
   if(var_S = '')
   then
```

```
        leave insert_loop2;
    end if;
    insert into `telefonocentro` (Centro,Numero)
    values (var_Centro,var_S);
    set var_A = var_A +1;
    end loop;
    commit;
    set autocommit=1;
END$$

DELIMITER ;
```

Questa Stored Procedure è stata realizzata per consentire l'inserimento di un nuovo centro all'interno della catena di servizio. Anche qui è stata realizzata una transazione per rendere atomico l'inserimento del centro dato che prevede più operazioni di insert differenti.

_InserisciTurno

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`_InserisciTurno`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `_InserisciTurno` (in var_Impiegato VARCHAR(45), in list_day
VARCHAR(150), in list_hour VARCHAR(150))
BEGIN
    declare cont int default 1;
    declare contH int default 1;
    declare impiegoDataInizio date;
    declare var_Day varchar(45);
    declare var_HourS time;
    declare var_HourE time;
    if(list_day='' or list_hour='')
    then
        signal sqlstate '45001' set message_text="Devi inserire almeno un giorno e almeno un orario";
    end if;
    update `turno di lavoro`
    set DataFine=curdate()
    where Impiegato=var_Impiegato and DataFine is null;
    insert into `turno di lavoro` (DataInizio,Impiegato)
    values (curdate(),var_Impiegato);
    insert_loop: loop
        select SPLIT_STR(list_day,";",cont) into var_Day;
        if(var_Day='')
            then leave insert_loop;
        end if;
        select SPLIT_STR(list_hour,";",contH) into var_HourS;
```

```
        select SPLIT_STR(list_hour,";",contH+1) into var_HourE;
        insert into `orariogiornaliero` values
(var_Day,var_Impiegato,curdate(),var_HourS,var_HourE);
        set cont=cont+1;
        set contH=contH+2;
    end loop;
END$$

DELIMITER ;
```

Questa procedura non è stata realizzata per essere utilizzata direttamente da un client ma per non ripetere più volte lo stesso codice usato in punti differenti di Stored Procedure differenti. Ciò che viene eseguito all'interno di questo codice è l'inserimento all'interno della tabella orariogiornaliero dell'orario per un turno di lavoro dato un impiegato. Dunque list_day list_hour sono stringhe di valori separate da ";" e sono rispettivamente una lista dei giorni e una lista di orari. Il Sistema fa corrispondere a ciascun giorno in list_day i primi due valori ancora non letti di list_hour, questi ultimi vengono interpretati come segue: primo valore orario di inizio secondo valore orario di fine. E' responsabilità del client assicurarsi che i valori presenti all'interno della stringa list_hour rispettino tutti la seguente sintassi minutes:seconds altrimenti l'operazione non andrà a buon fine.

inserisciTurnoDiLavoro

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`inserisciTurnoDiLavoro`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserisciTurnoDiLavoro` (in var_Impiegato VARCHAR(45), in
list_day VARCHAR(150), in list_hour VARCHAR(150))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        set autocommit=1;
        resignal;
    end;
    set autocommit=0;
    set transaction isolation level repeatable read;
    start transaction;
    call _InserisciTurno(var_Impiegato,list_day,list_hour);
    if (not exists (select * from `reportOre` where Impiegato=var_Impiegato and DataFine is null))
    then
```

```
    insert into `reportOre` (Impiegato,DataInizio,DataFine,Ore) values
(var_Impiegato,curdate(),NULL,0);
  end if;
  commit;
  set autocommit=1;
END$$

DELIMITER ;
```

Procedura realizzata per consentire l'inserimento di un nuovo turno di lavoro all'interno del Sistema. La Stored Procedure _InserisciTurno è stata discussa già in precedenza e si preoccupa di registrare correttamente le informazioni all'interno della tabella "orariogiornaliero". Se bene apparentemente non strettamente necessario si è scelto come livello di isolamento repeatable read dal momento che l'operazione viene eseguita con una frequenza attesa di 1 volta al mese che è comunque relativamente bassa e si vuole anche garantire la correttezza delle operazioni eseguite.

noleggiaFilm

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`noleggiaFilm`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `noleggiaFilm` (in var_Cliente varchar(45), in var_Titolo varchar(60),
in var_Regista varchar(45), in var_Settore INT, in var_DataS DATE, in var_Username
VARCHAR(45))
BEGIN
  declare var_Centro INT;
  declare var_Impiegato varchar(45);
  declare numero int;
  declare exit handler for sqlexception
    begin
      rollback;
      set autocommit=1;
      resignal;
    end;
  set autocommit=0;
  set transaction isolation level repeatable read;
  start transaction;
    select CF from `impiegato` where username=var_Username into var_Impiegato;
    select Centro from `impiego` where Impiegato=var_Impiegato and DataFine is null into
var_Centro;
    select numCopie from `filmeffettivo` where FilmTitolo=var_Titolo and
FilmRegista=var_Regista and SettoreCentro=var_Centro and SettoreCodice=var_Settore into
numero;
    if(numero<=0)
```

```
        then
            signal sqlstate '45001' set message_text=" Non ci sono copie disponibili";
        end if;
        update `filmeffettivo`
        set numCopie=numero-1
        where FilmTitolo=var_Titolo and FilmRegista=var_Regista and SettoreCentro=var_Centro
and SettoreCodice=var_Settore;
        insert into
noleggia(Cliente,FilmTitolo,FilmRegista,SettoreCodice,SettoreCentro,DataScadenza)
        values(var_Cliente,var_Titolo,var_Regista,var_Settore,var_Centro,var_DataS);
        commit;
        set autocommit=1;
END$$

DELIMITER ;
```

Procedura realizzata per consentire il noleggio di un film. L'operazione prevede l'inserimento di una nuova tupla nella tabella "noleggia" e l'aggiornamento del numero di copie attualmente disponibili per un dato film. Operazione transazionale eseguita a livello di isolamento repeatable read per evitare aggiornamenti fantasma causati ad esempio da altre operazioni transazionali di noleggio film operanti sulla stessa tupla. In caso non siano più disponibili copie da noleggiare viene sollevata una eccezione.

registraImpiegato

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`registraImpiegato`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `registraImpiegato` (in var_CF VARCHAR(45), in var_Nome
VARCHAR(45), in var_Recapito VARCHAR(45), in var_TitolodiStudio VARCHAR(60), in
var_Username VARCHAR(45),in var_Password VARCHAR(45), in list_day VARCHAR(150),
in list_hour VARCHAR(150), in var_Centro INT, in var_Ruolo VARCHAR(45),in var_Cognome
VARCHAR(45))
BEGIN
    declare exit handler for sqlexception
    begin
        rollback;
        set autocommit=1;
        resignal;
    end;
```

```
   set autocommit=0;
   set transaction isolation level read committed;
   START TRANSACTION;
      insert into `login` values(var_Username,md5(var_Password),"impiegato");
      insert into `impiegato`
values(var_CF,var_Nome,var_Recapito,var_TitolodiStudio,var_Username,var_Cognome);
      update `impiego`
      set DataFine=curdate()
      where Impiegato=var_CF and DataFine is null;
      insert into `impiego` (Impiegato,DataInizio,DataFine,Ruolo,Centro)
      values(var_CF,curdate(),null,var_Ruolo,var_Centro);
      call _InserisciTurno(var_CF,list_day,list_hour);
      commit;
   set autocommit=1;
END$$

DELIMITER ;
```

Questa procedura permette la registrazione di un nuovo impiegato presso la catena di servizio e prevede la creazione di un account, la registrazione di un impiego presso un dato centro e l'inserimento di un nuovo turno di lavoro valido per l'impiegato in questione. L'adozione di una procedura transazionale complessa consente l'inserimento di più operazioni in maniera atomica portando sempre ad una situazione stabile e coerente del Sistema. Ad esempio non possono esistere account di impiegati a cui non corrisponde un centro di servizio o a cui non corrisponde un turno di lavoro valido. Si noti come la non coerenza di queste informazioni porterebbe l'utente appena inserito a non poter correttamente operare all'interno del Sistema dato che molte delle operazioni affidate ad un impiegato vengono eseguite a partire dallo username e dall'impiego attuale di quest'ultimo. Il livello di isolamento read committed è più che sufficiente dal momento che non sono richieste letture multiple su uno stesso record.

registraImpiego

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`registraImpiego`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `registraImpiego` (in var_Impiegato VARCHAR(45), in var_Centro
INT,  in var_Ruolo VARCHAR(45), in list_day VARCHAR(150), in list_hour VARCHAR(150))
BEGIN
   declare old_Centro INT;
   declare exit handler for sqlexception
```

```
    begin
        rollback;
        set autocommit=1;
        resignal;
    end;
    set autocommit=0;
    set transaction isolation level repeatable read;
    start transaction;
    select Centro from `impiego` where Impiegato=var_Impiegato and DataFine is null into
old_Centro;
    update `impiego`
    set DataFine=curdate()
    where Impiegato=var_Impiegato and DataFine is null;
    insert into `impiego` (Impiegato,DataInizio,DataFine,Ruolo,Centro)
    values(var_Impiegato,curdate(),null,var_Ruolo,var_Centro);
    call _InserisciTurno(var_Impiegato,list_day,list_hour);
    if (var_Centro<>old_Centro)
    /*
        Questo indica un cambiamento nel luogo di lavoro quindi lo registro.
    */
    then
        update `reportOre`
        set DataFine=curdate()
        where Impiegato=var_Impiegato and DataFine is null;
        insert into `reportOre` (Impiegato,DataInizio,DataFine,Ore) values
(var_Impiegato,curdate(),NULL,0);
    end if;
    commit;
    set autocommit=1;
END$$

DELIMITER ;
```

Questa procedura è stata realizzata per permettere la registrazione di un nuovo impiego per un impiegato. Va utilizzata dunque noto un impiegato all'interno del Sistema per ad esempio spostare di sede un impiegato o modificarne il ruolo presso il centro a cui è assegnato.

stampaFilmScaduti()

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`stampaFilmScaduti`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `stampaFilmScaduti` ()
BEGIN

    set transaction read only;
```

```
    set transaction isolation level repeatable read;
    set autocommit=0;
    start transaction;
      select `centro`.`Codice`, `noleggia`.`FilmTitolo`,
`noleggia`.`Cliente`,`cliente`.`Nome`,`cliente`.`Cognome`
        from `noleggia` join `centro` on `noleggia`.`SettoreCentro`=`centro`.`Codice` join
`cliente` on `noleggia`.`Cliente`=`cliente`.`CF`
        where `noleggia`.`DataScadenza`<curdate();
      commit;
    set autocommit=1;
END$$

DELIMITER ;
```

Questa procedura permette di visualizzare tutti I titoli di film associati ad un noleggio scaduto ricercando tra i noleggi eseguiti presso tutti i centri della catena di servizio. Per ogni risultato verrà visualizzato il codice del centro, il titolo del film, il nome e cognome del cliente.

reportMensile

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`reportMensile`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `reportMensile` (in var_Impiegato VARCHAR(45), in var_Month
SMALLINT, in var_Year SMALLINT)
BEGIN
    declare var_Centro INT;
    declare var_Data date;
    declare done int default false;
    declare cur2 cursor for
    select `turno di lavoro`.`DataInizio` from `turno di lavoro` where
    MONTH(`turno di lavoro`.`DataInizio`)=var_Month and `turno di
lavoro`.`Impiegato`=var_Impiegato;
declare continue handler for not found set done = true;

  declare exit handler for sqlexception
  begin
    rollback;  -- rollback any changes made in the transaction
    set autocommit=1;
    resignal;  -- raise again the sql exception to the caller
  end;
  drop temporary table if exists `centroOre` ;
  create temporary table `centroOre`(
    `centro` INT ,
    `Ore` DECIMAL
    );
```

```
   set transaction isolation level serializable;
   set autocommit=0;
   start transaction;
      insert into `centroOre`
      select `Centro`,sum(`Ore`)
      from `impiego` join `reportOre` on `impiego`.`Impiegato`=`reportOre`.`Impiegato`
      where `impiego`.`Impiegato`=var_Impiegato and
      MONTH(`impiego`.`DataInizio`)<=var_Month and
      `reportOre`.`DataInizio`>=`impiego`.`DataInizio` and
(`reportOre`.`DataFine`<=`impiego`.`DataFine` or `impiego`.`DataFine` is null) and
MONTH(`reportOre`.`DataInizio`)=var_Month and YEAR(`reportOre`.`DataInizio`)=var_Year
      group by `Centro`;
      select `centro`.`Codice`, `centro`.`Indirizzo`, `centroOre`.`Ore`  from `centroOre` join `centro`
on `centroOre`.`centro`=`centro`.`Codice`;
      set done=false;
      open cur2;
other_loop: loop
      fetch cur2 into var_Data;
      if done
      then
         leave other_loop;
      end if;
      select `turno di lavoro`.`DataInizio`,`turno di lavoro`.`DataFine` from `turno di lavoro` where
`turno di lavoro`.`DataInizio`=var_Data and `turno di lavoro`.`Impiegato`=var_Impiegato;
      select `orariogiornaliero`.`Giorno`, `orariogiornaliero`.`OraInizio`,
`orariogiornaliero`.`OraFine` from
      `orariogiornaliero` where `orariogiornaliero`.`TurnoDiLavoroData`=var_Data and
`orariogiornaliero`.`TurnoDiLavoroI`=var_Impiegato;
      end loop;
      close cur2;
      commit;
      drop temporary table `centroOre`;
      set autocommit=1;
END$$

DELIMITER ;
```

Questa procedura è stata realizzata per visualizzare dato un impiegato un report mensile sulle ore lavorate e sui luoghi di lavoro. Nel corso della stored procedure viene creata una tabella temporanea chiamata “centroOre” per memorizzare le coppie centro di servizio e il numero di ore lavorate presso quel centro per il mese e anno richiesti. Per ricostruire questa informazione si accede alla tabella reportOre estraendo da quest’ultimo le ore spese presso il centro x sfruttando le date di inizio e fine impiego presso quell centro. Ad esempio supponiamo che un impiegato Tizio ha prestato servizio presso il centro x nel periodo che va dal 1 al 15 del mese y e presso il centro z per il periodo che va dal 16 al 31 del mese y. Allora per come è costruita la tabella reportOre avremo un record il

cui range temporale (ovvero la differenza tra la data di fine e inizio validità) ricade nel range temporale dell'impiego presso il centro x e un record il cui range temporale ricade in quello del centro y e così è possibile ricostruire l'informazione. Si noti che nella situazione in cui l'impiegato opera nel periodo dal 16 al 31 del mese y sempre presso il centro x ma con ruolo differente (quindi un impiego differente) allora la tabella reportOre avrà un unico record per il periodo che va dal 1 al 31 del mese y. In entrambi i casi si può sempre risalire all'informazione desiderata.
Successivamente viene aperto un cursore sui turni di lavoro dell'impiegato nel mese richiesto e questi vengono mostrati attraverso delle select per visualizzare il periodo di validità (del turno) e gli orari di lavoro.

reportAnnuale

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`reportAnnuale`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `reportAnnuale` (in var_Impiegato varchar(45), in var_Year
SMALLINT,in var_Turno BOOL)
BEGIN
   declare var_Centro INT;
   declare var_Data DATE;
      declare done int default false;
      declare cur2 cursor for
      select `turno di lavoro`.`DataInizio` from `turno di lavoro` where
      YEAR(`turno di lavoro`.`DataInizio`)=var_Year and `turno di
lavoro`.`Impiegato`=var_Impiegato;
      declare cur cursor for
      select distinct `Centro` from `impiego` where `impiego`.`Impiegato`=var_Impiegato and
YEAR(`impiego`.`DataInizio`)=var_Year;
      declare continue handler for not found set done = true;

   declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      set autocommit=1;
      resignal;  -- raise again the sql exception to the caller
   end;
   drop temporary table if exists `centroOre` ;
   create temporary table `centroOre`(
      `centro` INT PRIMARY KEY,
      `Ore` NUMERIC
      );
```

```
   set transaction isolation level serializable;
   set autocommit=0;
   start transaction;
   open cur;
   read_loop: loop
      fetch cur into var_Centro;
      if done
      then leave read_loop;
      end if;
      insert into `centroOre`
      select  `Centro`,sum(`Ore`)
      from `impiego` join `reportOre` on `impiego`.`Impiegato`=`reportOre`.`Impiegato`
      where `Centro`=var_Centro and `impiego`.`Impiegato`=var_Impiegato and
      YEAR(`impiego`.`DataInizio`)=var_Year and
      `reportOre`.`DataInizio`>=`impiego`.`DataInizio` and
(`reportOre`.`DataFine`<=`impiego`.`DataFine` or `impiego`.`DataFine` is null)
      group by `Centro`;
      end loop;
      close cur;
         select `centro`.`Codice`, `centro`.`Indirizzo`, `centroOre`.`Ore`  from `centroOre` join
`centro` on `centroOre`.`centro`=`centro`.`Codice`;
               set done=false;
      if(var_Turno)
      then
         open cur2;
   other_loop: loop
         fetch cur2 into var_Data;
         if done
         then
            leave other_loop;
         end if;
         select `turno di lavoro`.`DataInizio`,`turno di lavoro`.`DataFine` from `turno di lavoro`
where `turno di lavoro`.`DataInizio`=var_Data and `turno di lavoro`.`Impiegato`=var_Impiegato;
         select `orariogiornaliero`.`Giorno`, `orariogiornaliero`.`OraInizio`,
`orariogiornaliero`.`OraFine` from
         `orariogiornaliero` where `orariogiornaliero`.`TurnoDiLavoroData`=var_Data and
`orariogiornaliero`.`TurnoDiLavoroI`=var_Impiegato;
         end loop;
         close cur2;
      end if;
      commit;
      drop temporary table `centroOre`;
      set autocommit=1;
END$$

DELIMITER ;
```

Questa procedura permette di visualizzare (dato un impiegato e un anno di riferimento) un report

annuale sulle ore lavorate e sui luoghi di lavoro dell'impiegato nell'anno di riferimento. Molti aspetti di questa procedura sono affini a quelli descritti per la procedura "reportMensile" le differenze sono più che altro a livello logico dato che in questo caso si intende come periodo di validità per un impiego l'intero anno di riferimento, pertanto si va ad utilizzare la funzione YEAR (anziché MONTH) per estrarre le informazioni di interesse dalle date memorizzate e poi procedere al confronto. Inoltre poiché i turni di lavoro annuali sono in numero elevato (data la frequenza attesa si prospettano 12 turni di lavoro all'anno) si è deciso di inserire una possibilità di scelta attraverso un parametro booleano che se pari a true mostrerà anche i turni di lavoro similmente a quanto fatto con il report mensile.

timbra

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`timbra`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `timbra` (in var_username varchar(45))
BEGIN
   declare var_nuovo time default null;
   declare var_Impiegato varchar(45);
        declare exit handler for sqlexception
      begin
         rollback;
         set autocommit=1;
         resignal;
      end;
   set autocommit=0;
   set transaction isolation level repeatable read;
   start transaction;
   create  table if not exists `timbratura`(
      Impiegato VARCHAR(45) PRIMARY KEY,
      TimeInizio TIME,
      TimeFine TIME
      );
   select CF from `impiegato` where username=var_username into var_Impiegato;
   select timeInizio from `timbratura` where Impiegato=var_Impiegato into var_nuovo;
   if(var_nuovo is null)
   then
      insert into `timbratura` values(var_Impiegato,curtime(),NULL);
   else
      update `timbratura`
      set  timeFine=curtime()
      where Impiegato=var_Impiegato;
   end if;
```

```
    commit;
    set autocommit=1;
END$$

DELIMITER ;
```

Questa procedura permette dato un impiegato di eseguire una operazione simile all'azione fisica di "timbratura del cartellino". Si utilizza una tabella "timbratura" che viene rimossa dall'evento implementato. Tale tabella deve memorizzare gli orari di timbratura per tutti gli impiegati. Questi dati sono utili al fine del calcolo delle ore di lavoro ma non si ritengono utili ai fini del report finale e per questo non vengono memorizzate in modo permanente. Si noti che in questo caso non sarebbe possibile costruire una tabella temporanea dal momento che quest'ultima esiste solo nell'ambito della singola connessione (non ha scope globale quindi i dati non sarebbero accessibili dall'evento). In particolare questa procedura ha un duplice comportamento. Se la procedura viene eseguita per la prima volta nell'arco della giornata allora andrà ad inserire all'interno della tabella un record per l'impiegato in questione memorizzando come orario di inizio l'orario corrente. Altrimenti andrà ad aggiornare il valore dell'attributo "TimeFine". Si noti come la procedura non aggiorna il valore delle ore lavorate nella tabella reportOre, questo per una scelta progettuale ben precisa: si vuole evitare 55roblem legati ad una chiamata alla procedura eseguita nel momento sbagliato che non sia più reversibile (e.g. l'impiegato crede di aver terminato il turno ma poi gli viene chiesto di trattenersi qualche ora in più di straordinario). Non è quindi responsabilità di questa procedura aggiornare tale informazione ma bensì dell'evento giornaliero "ResettaGiorni" discusso nell'apposita sezione. Si può notare inoltre che la procedura non effettua controlli sul fatto che l'orario di inizio e fine siano compresi entro l'arco orario previsto dal turno di lavoro questo coerentemente alla scelta precedentemente discussa, in tal modo si permettono "straordinari" ai dipendenti che vengono comunque conteggiati nelle ore lavorate senza imporre alcuna operazione ai responsabili (e.g. manager).

Login

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`login`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `login` (in var_Username VARCHAR(45), in var_Password
VARCHAR(45), out var_Role INT)
BEGIN
      declare var_User ENUM("amministratore","impiegato","manager");
```

```
    select ruolo from `login` where username=var_Username and passw=md5(var_Password)
into var_User;
    if(var_User = "amministratore")
    then
      set var_Role=1;
    else
      if(var_User="impiegato")
      then set var_Role=2;
      else
        if(var_User="manager")
        then set var_Role=3;
        else
          set var_Role=4;
          end if;
          end if;
          end if;
END$$

DELIMITER ;
```

Discutiamo adesso una procedura molto importante dal punto di vista della sicurezza dell'applicazione. La procedura in questione permette all'utente di eseguire il login al Sistema ed autenticarsi e restituisce al chiamante il ruolo previsto per questo utente (il quale corrisponde ad una delle 4 tipologie di utenti previsti). E' responsabilità del chiamante gestire opportunamente questo valore di ritorno, ad esempio modificare l'utenza connessa al dbms eseguendo uno switch all'utenza restituita dalla procedura di login (che è esattamente ciò che è stato implementato nel client). La procedura di login è per un client che si connette all'applicazione il primo punto di accesso (e l'unico se non possiede credenziali valide). In questa sezione si vuole anche discutere un altro aspetto legato alla sicurezza: la scelta di consentire ad un impiegato di accedere e modificare unicamente i dati relativi al centro presso cui è impiegato consente di circoscrivere eventuali danni causabili da un intrusore che entra in possesso delle credenziali di tale impiegato. Infatti la procedura di login assegna il ruolo di "impiegato" il quale come visto nella apposita sezione ha diversi privilegi di esecuzione su stored procedure. Supponiamo che un intrusore entri in possesso delle credenziali di accesso per l'utente Tizio impiegato al centro x, la scelta presa fa si che l'intrusore sia in grado di operare solo relativamente al centro x senza danneggiare altri centri nella catena.

restituisciFilm

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`restituisciFilm`;
```

```
DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `restituisciFilm` (in var_Cliente VARCHAR(45), in var_Titolo
VARCHAR(60), in var_Regista VARCHAR(45), in var_Settore VARCHAR(45), in
var_Username VARCHAR(45))
BEGIN
      declare var_Centro int;
      declare var_Impiegato varchar(45);
        declare exit handler for sqlexception
      begin
        rollback;
    set autocommit=1;
        resignal;
      end;
      set autocommit=0;
      start transaction;
      select CF from `impiegato` where username=var_Username into var_Impiegato;
      select Centro from `impiego` where Impiegato=var_Impiegato and DataFine is null into
var_Centro;
      delete from `noleggia` where Cliente=var_Cliente and FilmTitolo=var_Titolo and
FilmRegista=var_Regista and SettoreCodice=var_Settore and SettoreCentro=var_Centro;
      update `filmeffettivo` set numCopie=numCopie+1 where FilmTitolo=var_Titolo and
FilmRegista=var_Regista and SettoreCodice=var_Settore and SettoreCentro=var_Centro;
      commit;
      set autocommit=1;
END$$

DELIMITER ;
```

Questa procedura permette di assolvere al compito di registrare la restituzione di un film noleggiato da parte di un cliente. Il tutto viene gestito andando semplicemente a rimuovere il noleggio dalla tabella “noleggia” ed incrementando il contatore del numero di copie disponibili per il film in questione. La necessità di una transazione è evidente si vuole evitare di rimuovere il noleggio senza effettivamente registrare nuovamente la copia come disponibile al noleggio.

visualizzaTurno

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`visualizzaTurno`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `visualizzaTurno` (in var_Username VARCHAR(45))
BEGIN
   declare var_Imp VARCHAR(45);
```

```
    declare var_T DATE;
       declare exit handler for sqlexception
       begin
          rollback;
         set autocommit=1;
          resignal;
       end;
    set autocommit=0;
    set transaction read only;
    start transaction;
    select CF from `impiegato` where username=var_Username into var_Imp;
    select DataInizio from `turno di lavoro` where Impiegato=var_Imp and DataFine is null into
var_T;
    select Giorno, OraInizio, OraFine
    from `orariogiornaliero` where TurnoDiLavoroI=var_Imp and TurnoDiLavoroData=var_T;
    commit;
    set autocommit=1;
END$$

DELIMITER ;
```

Questa procedura permette di prendere visione del turno di lavoro attualmente valido per un impiegato a partire dallo username di quest'ultimo. Si è deciso di effettuare questa operazione a partire dallo username e non dal codice fiscale perché si suppone che a richiedere l'operazione sia l'impiegato stesso. Di fatto l'unico utente che può eseguire tale procedura è l'utente "impiegato" e a livello client è stato implementato un meccanismo di accesso al turno che sfrutta lo username dell'utente loggato (dunque non richiede altre informazioni in ingresso) al fine di proteggere la privacy degli utenti (non puoi conoscere il turno di un impiegato che non sia tu stesso).

La seguente è una funzione e non una Stored Procedure vera e proprio. Viene riportata qui perché non vi è una apposita sezione ed avendone presentato esempi di utilizzo nelle precedenti store procedure ho ritenuto opportuno inserirla qui e non in appendice.

SPLIT_STR

```
USE `progetto1`;
DROP function IF EXISTS `progetto1`.`SPLIT_STR`;

DELIMITER $$
USE `progetto1`$$
CREATE FUNCTION  SPLIT_STR(
      x VARCHAR(255),
      delim VARCHAR(12),
      pos INT
```

```
)
RETURNS VARCHAR(255)
RETURN REPLACE(SUBSTRING(SUBSTRING_INDEX(x, delim, pos),
LENGTH(SUBSTRING_INDEX(x, delim, pos -1)) + 1),
delim, '');$$

DELIMITER ;
```

Questa funzione è stata realizzata per risolvere il seguente problema: Come utilizzare un tipo di dato aggregato (e.g. array) in SQL anche se questo linguaggio non ha un tale supporto? La soluzione è l'utilizzo di una stringa (un varchar per l'appunto che nella funzione è x) strutturata come stringa di valori separate da un delimitatore (nella funzione chiamato delim). Per tanto è stata realizzata la funzione SPLIT_STR per ottenere data una stringa come prima definita e specificato il delimitatore e un valore progressivo (come l'indice in un array) il valore corrispondente a quella posizione. Esempi di utilizzo nell'ambito dell'applicazione sono stati già riportati nelle store procedure. Di seguito un esempio: SPLIT_STR("Lunedì;Martedì;Mercoledì","";",2) restituisce come risultato "Martedì".

registraCliente

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`registraCliente`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `registraCliente` (in var_CF VARCHAR(45), in var_Nome
VARCHAR(45), in var_Indirizzi VARCHAR(300),in var_Cognome VARCHAR(45),in
var_Telefoni VARCHAR(100), in var_Email VARCHAR(200),in var_DataNascita DATE )
BEGIN
   declare contA int default 1;
   declare var_T VARCHAR(45);
   declare var_E VARCHAR(45);
   declare var_I VARCHAR(60);
      declare exit handler for sqlexception
      begin
         rollback;
         set autocommit=1;
         resignal;
      end;
   set autocommit=0;
   set transaction isolation level read uncommitted;
   start transaction;
   if(var_Telefoni = '' and var_Email = '')
   then
```

```
        signal sqlstate '45001' set message_text="Devi fornire almeno uno tra telefoni ed email";
    end if;
    if(var_Indirizzi='')
    then
        signal sqlstate '45001' set message_text="Devi fornire almeno un indirizzo";
    end if;
    insert into `cliente` values(var_CF,var_Nome,var_Cognome,var_DataNascita);
    insert_loop1: loop
    select SPLIT_STR(var_Telefoni,";",contA) into var_T;
    if(var_T='')
        then leave insert_loop1;
    end if;
    insert into `telefonocliente` values(var_CF,var_T);
    set contA=contA+1;
    end loop;
    set contA=1;
    insert_loop2: loop
    select SPLIT_STR(var_Email,";",contA) into var_E;
    if(var_E='')
        then leave insert_loop2;
    end if;
    insert into `emailcliente` values(var_CF,var_E);
    set contA=contA+1;
    end loop;
    set contA =1;
    insert_loop3: loop
    select SPLIT_STR(var_Indirizzi,";",contA) into var_I;
    if(var_I = '')
        then leave insert_loop3;
    end if;
    insert into `indirizzo cliente` values (var_CF,var_I);
    set contA = contA+1;
    end loop;
    commit;
    set autocommit=1;
END$$

DELIMITER ;
```

Questa store procedure è stata realizzata per consentire la registrazione di nuovi client presso la catena di servizio. In accordo con le specifiche fornite il cliente può fornire all'atto della registrazione un numero arbitrario di indirizzi, numeri telefonici ed email. Per questo la store procedure fa uso della funzione SPLIT_STR già descritta ampiamente in precedenza per consentire all'atto della registrazione l'inserimento del cliente e delle informazioni fornite come una operazione atomica. Questo impedisce per tanto che l'utente venga inserito nel Sistema senza che i dati ad esso associati (email,telefono e indirizzo) vengano registrati, situazione che porterebbe ad

una struttura incoerente della base di dati. Gli if-statement previsti inoltre evitano che venga registrato un cliente che non fornisce indirizzi e/o almeno un recapito all'atto della registrazione. Dovendo prevedere una dimensione limitata per le variabili di input corrispondenti agli indirizzi,email e telefono sono state previste anche store procedure per l'inserimento singolo di un indirizzo,email o telefono da effettuare successivamente alla registrazione (ad esempio nel caso in cui si voglia fornire più indirizzi di quelli realmente disponibili o il cliente vuole aggiungere informazioni in un momento successivo a quello della registrazione).

inserisciFilmSettore

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`inserisciFilmSettore`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserisciFilmSettore` (in var_Titolo VARCHAR(60), in var_Regista
VARCHAR(45), in var_Settore INT,in var_Username varchar(45), in var_Copie INT, in
var_posizione VARCHAR(45))
BEGIN
   declare var_Centro int;
   declare var_Impiegato varchar(45);
   declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      set autocommit=1;
      resignal;  -- raise again the sql exception to the caller
   end;
   set transaction isolation level read committed;
   set autocommit = 0;
   start transaction;
   select CF from `impiegato` where username=var_Username into var_Impiegato;
   select Centro from `impiego` where Impiegato=var_Impiegato and DataFine is null into
var_Centro;
   insert into `filmeffettivo` values (var_Titolo, var_Regista, var_Settore,var_Centro, var_Copie,
var_Posizione);
   commit;
   set autocommit = 1;
END$$

DELIMITER ;
```

Questa store procedure è stata realizzata per consentire la registrazione di un film presente presso la catena all'interno di un determinato settore in un dato centro. Sostanzialmente per mezzo di questa stored procedure è possibile registrare un nuovo "film effettivo" a partire da un film presente nel

catalogo della catena. Per quando riguarda il livello di isolamento scelto si è ritenuto read committed sufficiente in quanto questa procedura ha si bisogno di leggere ma non esegue letture ripetute su uno stesso dato.

visualizzaFilmCentro

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`visualizzaFilmCentro`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `visualizzaFilmCentro` (in var_Username varchar(45))
BEGIN
   declare var_Impiegato varchar(45);
   declare var_Centro int;
   select CF from `impiegato` where username=var_Username into var_Impiegato;
   select Centro from `impiego` where Impiegato=var_Impiegato and DataFine is null into
var_Centro;
   select FilmTitolo as `Titolo`,FilmRegista as `Regista`,SettoreCodice as `Settore`,numCopie as
`Numero di copie`,posizione as `Posizione`,costo as `Costo` from `filmeffettivo` join `film` on
FilmTitolo = Titolo and FilmRegista = Regista where SettoreCentro=var_Centro;
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire l'accesso al catalogo dei film disponibili presso il centro per il quale lavora l'impiegato a cui corrisponde lo username in var_Username, questo in accordo con la scelta di limitare il permesso di accesso e gestione dei film di un centro x solo ad un impiegato realmente assegnato a tale centro.

rimuoviFilm

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`rimuoviFilm`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `rimuoviFilm` (in var_FilmTitolo VARCHAR(60), in var_FilmRegista
VARCHAR(45))
BEGIN
   delete from film where Titolo=var_FilmTitolo and Regista=var_FilmRegista;
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire la rimozione di un film dal catalogo della catena

rimuoviManager

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`rimuoviManager`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `rimuoviManager` (in var_Username VARCHAR(45))
BEGIN
   delete from login where username=var_Username and ruolo="manager";
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire la rimozione dell'account associato ad un manager.

rimuoviImpiegato

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`rimuoviImpiegato`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `rimuoviImpiegato` (in var_Username VARCHAR(45))
BEGIN
   delete from login where username=var_Username and ruolo=”impiegato”;
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire la rimozione dell'account associato ad un impiegato.

rimuoviFilmSettore

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`rimuoviFilmSettore`;
```

```
DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `rimuoviFilmSettore` (in var_Titolo VARCHAR(60), in var_Regista
VARCHAR(45), in var_Settore INT, in var_Username VARCHAR(45))
BEGIN
   declare var_Centro int;
   declare var_CF VARCHAR(45);
      declare exit handler for sqlexception
      begin
         rollback;
         set autocommit=1;
         resignal;
      end;
   set autocommit = 0;
   set transaction isolation level read committed;
   start transaction;
   select CF from `impiegato` where username=var_Username into var_CF;
   select Centro from `impiego` where Impiegato=var_CF and DataFine is null into var_Centro;
   delete from `filmeffettivo` where FilmTitolo=var_Titolo and FilmRegista=var_Regista and
SettoreCodice=var_Settore and SettoreCentro=var_Centro;
   commit;
   set autocommit = 1;
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire la rimozione di un film da un settore in un determinato centro

trovaCliente

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`trovaCliente`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `trovaCliente` (in var_Cliente VARCHAR(45))
BEGIN
   set autocommit = 0;
   set transaction read only;
   set transaction isolation level read committed;
   start transaction;
   select * from `cliente` where `cliente`.`CF`=var_Cliente;
   select Numero from `telefonocliente` where `telefonocliente`.`Cliente`=var_Cliente;
   select Email from `emailcliente` where `emailcliente`.`Cliente`=var_Cliente;
   select Indirizzo from `indirizzo cliente` where `indirizzo cliente`.`Cliente` =var_Cliente;
   select FilmTitolo,FilmRegista,DataScadenza from `noleggia` where
`noleggia`.`Cliente`=var_Cliente;
```

```
   commit;
   set autocommit=1;
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per fornire un punto di accesso al client per la visualizzazione di tutti i dati legati ad un impiegato e i suoi noleggi in corso.

visualizzaNoleggi

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`visualizzaNoleggi`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `visualizzaNoleggi` (in var_Username VARCHAR(45))
BEGIN
   declare var_Centro int;
   select Centro from impiego join impiegato on impiego.Impiegato = impiegato.CF where
username=var_Username and DataFine is null into var_Centro;
   select Cliente,FilmTitolo as `Titolo`,FilmRegista as `Regista`,SettoreCodice as
`Settore`,DataScadenza as `Data di Scadenza`
   from `noleggia` where SettoreCentro = var_Centro;
END$$

DELIMITER ;
```

inserisciEmailCliente

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`inserisciEmailCliente`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserisciEmailCliente` (in var_Email VARCHAR(45), in var_CF
VARCHAR(45))
BEGIN
   insert into emailcliente values(var_CF,var_Email);
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire l'inserimento di ulteriori indirizzi email per un dato cliente successivamente alla sua registrazione. Si noti infatti che l'attributo Cliente di emailcliente è legato all'attributo CF di Cliente tramite vincolo di foreign key.

inserisciTelefonoCliente

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`inserisciTelefonoCliente`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserisciTelefonoCliente` (in var_Telefono VARCHAR(45), in var_CF
VARCHAR(45))
BEGIN
   insert into telefonocliente values(var_CF,var_Telefono);
END$$

DELIMITER ;
```

Come per la precedente stored procedure descritta anche questa è stata introdotta per consentire l'aggiunta di ulteriori numeri di telefono

inserisciTelefonoCentro

```
DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserisciTelefonoCentro` (in var_Centro INT, in var_Telefono
VARCHAR(15))
BEGIN
        declare exit handler for sqlexception
      begin
         rollback;
         resignal;
      end;
   insert into telefonocentro values (var_Centro,var_Telefono);
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire l'inserimento di ulteriori numeri di telefono per un dato centro registrato nella base dati.

inserisciEmailCentro

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`inserisciEmailCentro`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserisciEmailCentro` (in var_Centro INT, in var_Email
VARCHAR(45))
BEGIN
   insert into emailcentro values (var_Centro,var_Email);
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire l'inserimento di ulteriori indirizzi email per un dato centro registrato nella base dati.

inserisciIndirizzoCliente

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`inserisciIndirizzoCliente`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `inserisciIndirizzoCliente` (in var_Indirizzo VARCHAR(60), in
var_Cliente VARCHAR(45))
BEGIN
   insert into `indirizzo cliente` values (var_Cliente,var_Indirizzo);
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire l'inserimento di ulteriori indirizzi per un dato cliente registrato nella base dati.

rimuoviCliente

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`rimuoviCliente`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `rimuoviCliente` (in var_Cliente VARCHAR(45))
BEGIN
   delete from cliente where CF = var_Cliente;
```

```
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire la rimozione di un cliente registrato presso il centro. Si noti come a causa dei vincoli imposti sulle chiavi esterne nella tabella "Noleggia" questa operazione andrà a buon fine soltanto se il cliente non ha noleggi attualmente in corso. Di fatto in caso contrario si perderebbero tali informazioni. Si sarebbe potuto scegliere una strada meno rigorosa ad esempio consentendo la rimozione e impostando a null il valore dell'attributo nelle tuple corrispondenti in Noleggia, ma dal momento che Cliente rappresenta parte della chiave primaria questo non è possibile. In generale tuttavia questa situazione non sarebbe praticabile in quanto si perderebbe l'informazione relativa a chi ha eseguito tale noleggio.

aggiornaDisponibilita

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`aggiornaDisponibilita`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `aggiornaDisponibilita` (in var_Username VARCHAR(45), in
var_Titolo VARCHAR(60), in var_Regista VARCHAR(45), in var_Settore INT, in var_Copie
INT)
BEGIN
   declare var_Centro INT;
   declare exit handler for sqlexception
      begin
         rollback;
         set autocommit=1;
         resignal;
      end;
   set autocommit = 0;
   start transaction;
   select Centro from `impiego` where DataFine is null and Impiegato = (select CF from
`impiegato` where username=var_Username) into var_Centro;
   update `filmeffettivo` set numCopie = numCopie + var_Copie
   where FilmTitolo = var_Titolo and FilmRegista = var_Regista and SettoreCodice = var_Settore
and SettoreCentro = var_Centro;
   commit;
   set autocommit = 1;
END$$

DELIMITER ;
```

Questa stored procedure è stata realizzata per consentire ad un impiegato di "ordinare" nuove copie per un film attualmente disponibile presso il centro. Sostanzialmente la stored procedure permette di aggiornare il numero di copie disponibili per un film incrementandolo di var_Copie

visualizzaFilmMancanti

```
USE `progetto1`;
DROP procedure IF EXISTS `progetto1`.`visualizzaFilmMancanti`;

DELIMITER $$
USE `progetto1`$$
CREATE PROCEDURE `visualizzaFilmMancanti` (in var_Username VARCHAR(45),in
var_Settore INT)
BEGIN
   select * from `film` where (Titolo,Regista) not in (select FilmTitolo,FilmRegista from
`filmeffettivo` where SettoreCentro = (select Centro from `impiego` where DataFine is null and
Impiegato = (select CF from `impiegato` where username=var_Username)) and SettoreCodice =
var_Settore);
END$$

DELIMITER ;
```

Questa stored procedure permette di visualizzare tutti e soli i film disponibili presso la catena ma non ancora inseriti all'interno dello specifico centro presso il quale è impiegato l'impiegato avente var_Username come username.

Appendice: Implementazione

Codice SQL per instanziare il database

```
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,N
O_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema progetto1
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `progetto1` ;

-- -----------------------------------------------------
-- Schema progetto1
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `progetto1` DEFAULT CHARACTER SET utf8 ;
USE `progetto1` ;

-- -----------------------------------------------------
-- Table `progetto1`.`film`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`film` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`film` (
  `Titolo` VARCHAR(60) NOT NULL,
  `Regista` VARCHAR(45) NOT NULL,
  `Tipo` ENUM("other", "nuovo", "classico") NOT NULL COMMENT 'Must be: Other, New or
Classic',
  `AnnoPubblicazione` YEAR NOT NULL,
  `Costo` DECIMAL NOT NULL,
  PRIMARY KEY (`Titolo`, `Regista`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`attore`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`attore` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`attore` (
  `FilmTitolo` VARCHAR(60) NOT NULL,
  `FilmRegista` VARCHAR(45) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`FilmTitolo`, `FilmRegista`, `Nome`),
```

```
  INDEX `Film_idx` (`FilmTitolo` ASC, `FilmRegista` ASC) VISIBLE,
  CONSTRAINT `FilmRecitato`
    FOREIGN KEY (`FilmTitolo` , `FilmRegista`)
    REFERENCES `progetto1`.`film` (`Titolo` , `Regista`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`centro`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`centro` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`centro` (
  `Codice` INT NOT NULL AUTO_INCREMENT,
  `Indirizzo` VARCHAR(60) NOT NULL,
  `Responsabile` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Codice`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`cliente` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`cliente` (
  `CF` VARCHAR(45) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `Data di Nascita` DATE NOT NULL,
  PRIMARY KEY (`CF`),
  UNIQUE INDEX `CF_UNIQUE` (`CF` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`emailcentro`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`emailcentro` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`emailcentro` (
  `Centro` INT NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Centro`, `Email`),
  CONSTRAINT `CentroAppartenenza`
```

```
  FOREIGN KEY (`Centro`)
  REFERENCES `progetto1`.`centro` (`Codice`)
  ON DELETE CASCADE
  ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`settore`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`settore` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`settore` (
  `Codice` INT NOT NULL,
  `Centro` INT NOT NULL,
  INDEX `Centro_idx` (`Centro` ASC) VISIBLE,
  PRIMARY KEY (`Codice`, `Centro`),
  CONSTRAINT `Centro`
    FOREIGN KEY (`Centro`)
    REFERENCES `progetto1`.`centro` (`Codice`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`filmeffettivo`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`filmeffettivo` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`filmeffettivo` (
  `FilmTitolo` VARCHAR(60) NOT NULL,
  `FilmRegista` VARCHAR(45) NOT NULL,
  `SettoreCodice` INT NOT NULL,
  `SettoreCentro` INT NOT NULL,
  `numCopie` INT NOT NULL,
  `posizione` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`FilmTitolo`, `FilmRegista`, `SettoreCodice`, `SettoreCentro`),
  INDEX `Settore_idx` (`SettoreCentro` ASC, `SettoreCodice` ASC) VISIBLE,
  CONSTRAINT `Settore`
    FOREIGN KEY (`SettoreCentro` , `SettoreCodice`)
    REFERENCES `progetto1`.`settore` (`Centro` , `Codice`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `Film`
    FOREIGN KEY (`FilmTitolo` , `FilmRegista`)
    REFERENCES `progetto1`.`film` (`Titolo` , `Regista`)
    ON DELETE CASCADE
```

```
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`login`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`login` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`login` (
  `username` VARCHAR(45) NOT NULL,
  `passw` VARCHAR(33) NOT NULL,
  `ruolo` ENUM("impiegato", "manager", "amministratore") NOT NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `progetto1`.`impiegato`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`impiegato` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`impiegato` (
  `CF` VARCHAR(45) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Recapito` VARCHAR(45) NOT NULL,
  `TitolodiStudio` VARCHAR(60) NOT NULL,
  `username` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`CF`),
  UNIQUE INDEX `username_UNIQUE` (`username` ASC) VISIBLE,
  CONSTRAINT `userkey`
    FOREIGN KEY (`username`)
    REFERENCES `progetto1`.`login` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`impiego`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`impiego` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`impiego` (
  `Impiegato` VARCHAR(45) NOT NULL,
  `DataInizio` DATE NOT NULL,
  `DataFine` DATE NULL DEFAULT NULL,
```

```
  `Ruolo` VARCHAR(45) NOT NULL,
  `Centro` INT NOT NULL,
  PRIMARY KEY (`Impiegato`, `DataInizio`),
  INDEX `Centro_idx` (`Centro` ASC) VISIBLE,
  CONSTRAINT `CentroDiImpiego`
    FOREIGN KEY (`Centro`)
    REFERENCES `progetto1`.`centro` (`Codice`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `Impiegato`
    FOREIGN KEY (`Impiegato`)
    REFERENCES `progetto1`.`impiegato` (`CF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`noleggia`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`noleggia` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`noleggia` (
  `Cliente` VARCHAR(45) NOT NULL,
  `FilmTitolo` VARCHAR(60) NOT NULL,
  `FilmRegista` VARCHAR(45) NOT NULL,
  `SettoreCodice` INT NOT NULL,
  `SettoreCentro` INT NOT NULL,
  `DataScadenza` DATE NOT NULL,
  PRIMARY KEY (`Cliente`, `FilmTitolo`, `FilmRegista`, `SettoreCodice`, `SettoreCentro`),
  INDEX `FilmNoleggiato_idx` (`FilmTitolo` ASC, `FilmRegista` ASC, `SettoreCodice` ASC,
`SettoreCentro` ASC) VISIBLE,
  INDEX `Cliente_idx` (`Cliente` ASC) VISIBLE,
  CONSTRAINT `FilmNoleggiato`
    FOREIGN KEY (`FilmTitolo` , `FilmRegista` , `SettoreCodice` , `SettoreCentro`)
    REFERENCES `progetto1`.`filmeffettivo` (`FilmTitolo` , `FilmRegista` , `SettoreCodice` ,
`SettoreCentro`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `cliente`
    FOREIGN KEY (`Cliente`)
    REFERENCES `progetto1`.`cliente` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


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

```
-- Table `progetto1`.`turno di lavoro`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`turno di lavoro` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`turno di lavoro` (
  `DataInizio` DATE NOT NULL,
  `Impiegato` VARCHAR(45) NOT NULL,
  `DataFine` DATE NULL,
  PRIMARY KEY (`DataInizio`, `Impiegato`),
  INDEX `ImpiegoLavoro_idx` (`Impiegato` ASC) VISIBLE,
  CONSTRAINT `ImpiegoLavoro`
    FOREIGN KEY (`Impiegato`)
    REFERENCES `progetto1`.`impiegato` (`CF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`orariogiornaliero`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`orariogiornaliero` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`orariogiornaliero` (
  `Giorno` ENUM('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday') NOT NULL,
  `TurnoDiLavoroI` VARCHAR(45) NOT NULL,
  `TurnoDiLavoroData` DATE NOT NULL,
  `OraInizio` TIME NOT NULL,
  `OraFine` TIME NOT NULL COMMENT 'Non Capisco perchè time non funziona.',
  PRIMARY KEY (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`),
  INDEX `turnodilavoro_idx` (`TurnoDiLavoroI` ASC, `TurnoDiLavoroData` ASC) VISIBLE,
  CONSTRAINT `turnodilavoro`
    FOREIGN KEY (`TurnoDiLavoroI` , `TurnoDiLavoroData`)
    REFERENCES `progetto1`.`turno di lavoro` (`Impiegato` , `DataInizio`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`telefonocentro`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`telefonocentro` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`telefonocentro` (
  `Centro` INT NOT NULL,
  `Numero` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`Centro`, `Numero`),
```

```
  CONSTRAINT `CentroApp`
    FOREIGN KEY (`Centro`)
    REFERENCES `progetto1`.`centro` (`Codice`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;


-- -----------------------------------------------------
-- Table `progetto1`.`remake`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`remake` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`remake` (
  `FilmTitolo1` VARCHAR(60) NOT NULL,
  `FilmRegista1` VARCHAR(45) NOT NULL,
  `FilmTitolo2` VARCHAR(60) NOT NULL,
  `FilmRegista2` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`FilmTitolo1`, `FilmRegista1`, `FilmTitolo2`, `FilmRegista2`),
  INDEX `FilmRiferito2_idx` (`FilmTitolo2` ASC, `FilmRegista2` ASC) VISIBLE,
  CONSTRAINT `FilmRiferito`
    FOREIGN KEY (`FilmTitolo1` , `FilmRegista1`)
    REFERENCES `progetto1`.`film` (`Titolo` , `Regista`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `FilmRiferito2`
    FOREIGN KEY (`FilmTitolo2` , `FilmRegista2`)
    REFERENCES `progetto1`.`film` (`Titolo` , `Regista`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `progetto1`.`telefonocliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`telefonocliente` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`telefonocliente` (
  `Cliente` VARCHAR(45) NOT NULL,
  `Numero` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`Cliente`, `Numero`),
  CONSTRAINT `ClienteTel`
    FOREIGN KEY (`Cliente`)
    REFERENCES `progetto1`.`cliente` (`CF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `progetto1`.`emailcliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`emailcliente` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`emailcliente` (
  `Cliente` VARCHAR(45) NOT NULL,
  `Email` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Cliente`, `Email`),
  CONSTRAINT `ClienteE`
    FOREIGN KEY (`Cliente`)
    REFERENCES `progetto1`.`cliente` (`CF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `progetto1`.`reportOre`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`reportOre` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`reportOre` (
  `DataInizio` DATE NOT NULL,
  `Impiegato` VARCHAR(45) NOT NULL,
  `Ore` DECIMAL NOT NULL DEFAULT 0,
  `DataFine` DATE NULL COMMENT 'Tabella necessaria per memorizzare esattamente quante ore l\'impiegato ha lavorato in quale mese. N.B. DataFine può essere il 31 del mese oppure un valore prima se l\'impiegato cambia centro di lavoro nel mentre.',
  PRIMARY KEY (`DataInizio`, `Impiegato`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `progetto1`.`indirizzo cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `progetto1`.`indirizzo cliente` ;

CREATE TABLE IF NOT EXISTS `progetto1`.`indirizzo cliente` (
  `Cliente` VARCHAR(45) NOT NULL,
  `Indirizzo` VARCHAR(60) NOT NULL,
  PRIMARY KEY (`Cliente`, `Indirizzo`),
  CONSTRAINT `Cliente_Indirizzo_FK`
    FOREIGN KEY (`Cliente`)
    REFERENCES `progetto1`.`cliente` (`CF`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
DELIMITER ;
```

```
SET SQL_MODE = '';
DROP USER IF EXISTS login;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'login' IDENTIFIED BY 'login';

GRANT EXECUTE ON procedure `progetto1`.`login` TO 'login';
SET SQL_MODE = '';
DROP USER IF EXISTS impiegato;
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 'impiegato' IDENTIFIED BY 'impiegato';

GRANT EXECUTE ON procedure `progetto1`.`registraCliente` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`noleggiaFilm` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`timbra` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`inserisciFilmSettore` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`restituisciFilm` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`stampaFilmScaduti` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`visualizzaTurno` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`visualizzaFilmCentro` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`rimuoviFilmSettore` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`trovaCliente` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`visualizzaNoleggi` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`inserisciEmailCliente` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`inserisciTelefonoCliente` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`inserisciIndirizzoCliente` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`rimuoviCliente` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`visualizzaFilmMancanti` TO 'impiegato';
GRANT EXECUTE ON procedure `progetto1`.`aggiornaDisponibilita` TO 'impiegato';
SET SQL_MODE = '';
DROP USER IF EXISTS manager;
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 'manager' IDENTIFIED BY 'manager';

GRANT EXECUTE ON procedure `progetto1`.`registraImpiego` TO 'manager';
GRANT EXECUTE ON procedure `progetto1`.`registraImpiegato` TO 'manager';
GRANT EXECUTE ON procedure `progetto1`.`creaUtente` TO 'manager';
GRANT EXECUTE ON procedure `progetto1`.`reportMensile` TO 'manager';
GRANT EXECUTE ON procedure `progetto1`.`inserisciTurnoDiLavoro` TO 'manager';
GRANT EXECUTE ON procedure `progetto1`.`visualizzaImpiegati` TO 'manager';
GRANT EXECUTE ON procedure `progetto1`.`visualizzaCentri` TO 'manager';
GRANT EXECUTE ON procedure `progetto1`.`reportAnnuale` TO 'manager';
GRANT EXECUTE ON procedure `progetto1`.`rimuoviImpiegato` TO 'manager';
SET SQL_MODE = '';
DROP USER IF EXISTS amministratore;
```

```
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 'amministratore' IDENTIFIED BY 'amministratore';

GRANT EXECUTE ON procedure `progetto1`.`registraImpiego` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`registraImpiegato` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`creaUtente` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`reportMensile` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`inserisciTurnoDiLavoro` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`visualizzaImpiegati` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`visualizzaCentri` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`reportAnnuale` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`rimuoviImpiegato` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`creaUtente` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`inserisciCentro` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`inserisciSettore` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`inserireFilmCatalogo` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`rimuoviManager` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`rimuoviFilm` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`inserisciTelefonoCentro` TO 'amministratore';
GRANT EXECUTE ON procedure `progetto1`.`inserisciEmailCentro` TO 'amministratore';

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

-- ----------------------------------------------------
-- Data for table `progetto1`.`film`
-- ----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`film` (`Titolo`, `Regista`, `Tipo`, `AnnoPubblicazione`, `Costo`)
VALUES ('Titanic', 'James Cameron', 'classico', 1997, 30);
INSERT INTO `progetto1`.`film` (`Titolo`, `Regista`, `Tipo`, `AnnoPubblicazione`, `Costo`)
VALUES ('Matrix', ' Andy e Larry Wachowski', 'other', 1999, 20);
INSERT INTO `progetto1`.`film` (`Titolo`, `Regista`, `Tipo`, `AnnoPubblicazione`, `Costo`)
VALUES ('Non ci resta che piangere', 'Roberto Benigni e Massimo Troisi', 'classico', 1984, 15);
INSERT INTO `progetto1`.`film` (`Titolo`, `Regista`, `Tipo`, `AnnoPubblicazione`, `Costo`)
VALUES ('Pensavo fosse amore... invece era un calesse', 'Massimo Troisi', 'classico', 1991, 15);
INSERT INTO `progetto1`.`film` (`Titolo`, `Regista`, `Tipo`, `AnnoPubblicazione`, `Costo`)
VALUES ('Tolo Tolo', 'Checco Zalone', 'nuovo', 2020, 40);
INSERT INTO `progetto1`.`film` (`Titolo`, `Regista`, `Tipo`, `AnnoPubblicazione`, `Costo`)
VALUES ('Totò, Peppino e la... malafemmina', 'Camillo Mastrocinque', 'classico', 1956, 5);
INSERT INTO `progetto1`.`film` (`Titolo`, `Regista`, `Tipo`, `AnnoPubblicazione`, `Costo`)
VALUES ('Harry Potter e la pietra filosofale', 'Chris Columbus', 'other', 2001, 10);
INSERT INTO `progetto1`.`film` (`Titolo`, `Regista`, `Tipo`, `AnnoPubblicazione`, `Costo`)
VALUES ('Harry Potter e la camera dei segreti', 'Chris Columbus', 'other', 2002, 10);
INSERT INTO `progetto1`.`film` (`Titolo`, `Regista`, `Tipo`, `AnnoPubblicazione`, `Costo`)
VALUES ('Pinocchio', 'Matteo Garrone', 'nuovo', 2019, 30);
```

```
COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`attore`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Titanic', 'James Cameron', 'Leonardo Di Caprio');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Titanic', 'James Cameron', 'Kate Winslet');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Matrix', ' Andy e Larry Wachowski', 'Keanu Reeves');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Matrix', ' Andy e Larry Wachowski', 'Laurence Fishburne');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Non ci resta che piangere', 'Roberto Benigni e Massimo Troisi', 'Roberto Benigni');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Non ci resta che piangere', 'Roberto Benigni e Massimo Troisi', 'Massimo Troisi');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Pensavo fosse amore... invece era un calesse', 'Massimo Troisi', 'Massimo Troisi');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Pensavo fosse amore... invece era un calesse', 'Massimo Troisi', 'Francesca Neri');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Tolo Tolo', 'Checco Zalone', 'Checco Zalone');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Tolo Tolo', 'Checco Zalone', 'Souleymane Sylla');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Totò, Peppino e la... malafemmina', 'Camillo Mastrocinque', 'Totò');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Totò, Peppino e la... malafemmina', 'Camillo Mastrocinque', 'Peppino De Filippo');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Harry Potter e la pietra filosofale', 'Chris Columbus', 'Daniel Radcliffe');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Harry Potter e la pietra filosofale', 'Chris Columbus', 'Rupert Grint');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Harry Potter e la pietra filosofale', 'Chris Columbus', 'Emma Watson');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Harry Potter e la camera dei segreti', 'Chris Columbus', 'Daniel Radcliffe');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Harry Potter e la camera dei segreti', 'Chris Columbus', 'Rupert Grint');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Harry Potter e la camera dei segreti', 'Chris Columbus', 'Emma Watson');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Pinocchio', 'Matteo Garrone', 'Roberto Benigni');
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Pinocchio', 'Matteo Garrone', 'Federico Ielapi');
```

```
INSERT INTO `progetto1`.`attore` (`FilmTitolo`, `FilmRegista`, `Nome`) VALUES ('Pinocchio', 'Matteo Garrone', 'Rocco Papaleo');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`centro`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`centro` (`Codice`, `Indirizzo`, `Responsabile`) VALUES (DEFAULT, 'Via Corte Da Capo', 'Matteo');
INSERT INTO `progetto1`.`centro` (`Codice`, `Indirizzo`, `Responsabile`) VALUES (DEFAULT, 'Via Pietre Bianche', 'Siria');
INSERT INTO `progetto1`.`centro` (`Codice`, `Indirizzo`, `Responsabile`) VALUES (DEFAULT, 'Via Principale', 'Antonio');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`cliente`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`cliente` (`CF`, `Nome`, `Cognome`, `Data di Nascita`) VALUES ('CCCMTT99H15I676', 'Matteo', 'Ciccaglione', '1999-06-15');
INSERT INTO `progetto1`.`cliente` (`CF`, `Nome`, `Cognome`, `Data di Nascita`) VALUES ('BNMSRI98G47J702', 'Siria', 'Buonamano', '1998-04-16');
INSERT INTO `progetto1`.`cliente` (`CF`, `Nome`, `Cognome`, `Data di Nascita`) VALUES ('CCCMCC72H5I676', 'Marcello', 'Ciccaglione', '1972-04-05');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`emailcentro`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`emailcentro` (`Centro`, `Email`) VALUES (1, 'matteoc8@live.com');
INSERT INTO `progetto1`.`emailcentro` (`Centro`, `Email`) VALUES (2, 'siriabuonamano@outlook.it');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`settore`
```

```
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`settore` (`Codice`, `Centro`) VALUES (1, 1);
INSERT INTO `progetto1`.`settore` (`Codice`, `Centro`) VALUES (2, 1);
INSERT INTO `progetto1`.`settore` (`Codice`, `Centro`) VALUES (3, 1);
INSERT INTO `progetto1`.`settore` (`Codice`, `Centro`) VALUES (4, 1);
INSERT INTO `progetto1`.`settore` (`Codice`, `Centro`) VALUES (5, 1);
INSERT INTO `progetto1`.`settore` (`Codice`, `Centro`) VALUES (1, 2);
INSERT INTO `progetto1`.`settore` (`Codice`, `Centro`) VALUES (2, 2);
INSERT INTO `progetto1`.`settore` (`Codice`, `Centro`) VALUES (1, 3);
INSERT INTO `progetto1`.`settore` (`Codice`, `Centro`) VALUES (2, 3);

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`filmeffettivo`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Titanic', 'James Cameron', 1, 1, 10, 'Scaffale
in alto');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Titanic', 'James Cameron', 2, 2, 5, 'Secondo
piano');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Matrix', ' Andy e Larry Wachowski', 2, 1, 5,
'Primo piano in fondo');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Matrix', ' Andy e Larry Wachowski', 1, 3, 10,
'All\'ingresso');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Non ci resta che piangere', 'Roberto Benigni e
Massimo Troisi', 3, 1, 20, 'All\'ingresso');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Non ci resta che piangere', 'Roberto Benigni e
Massimo Troisi', 2, 3, 10, 'Scaffale a destra');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Non ci resta che piangere', 'Roberto Benigni e
Massimo Troisi', 1, 2, 10, 'All\'ingresso');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Pensavo fosse amore... invece era un calesse',
'Massimo Troisi', 3, 1, 30, 'All\'ingresso');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Pensavo fosse amore... invece era un calesse',
'Massimo Troisi', 1, 2, 2, 'All\'ingresso');
```

```
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Tolo Tolo', 'Checco Zalone', 4, 1, 25, 'In
fondo a destra');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Tolo Tolo', 'Checco Zalone', 1, 3, 1, 'Vicino
all\'uscita');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Totò, Peppino e la... malafemmina', 'Camillo
Mastrocinque', 1, 2, 10, 'Vicino all\'uscita');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Harry Potter e la pietra filosofale', 'Chris
Columbus', 3, 1, 40, 'Vicino all\'uscita');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Harry Potter e la pietra filosofale', 'Chris
Columbus', 1, 3, 1, 'All\'ingresso');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Harry Potter e la camera dei segreti', 'Chris
Columbus', 3, 1, 30, 'In fondo');
INSERT INTO `progetto1`.`filmeffettivo` (`FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `numCopie`, `posizione`) VALUES ('Pinocchio', 'Matteo Garrone', 1, 3, 5, 'In
fondo');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`login`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('mario98',
'ce86d7d02a229acfaca4b63f01a1171b', 'impiegato');
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('marco72',
'ce86d7d02a229acfaca4b63f01a1171b', 'impiegato');
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('matt',
'ce86d7d02a229acfaca4b63f01a1171b', 'manager');
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('admin',
'ce86d7d02a229acfaca4b63f01a1171b', 'amministratore');
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('franco',
'ce86d7d02a229acfaca4b63f01a1171b', 'impiegato');
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('giuseppe',
'ce86d7d02a229acfaca4b63f01a1171b', 'impiegato');
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('luigi',
'ce86d7d02a229acfaca4b63f01a1171b', 'impiegato');
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('fabio',
'ce86d7d02a229acfaca4b63f01a1171b', 'impiegato');
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('alex',
'ce86d7d02a229acfaca4b63f01a1171b', 'manager');
INSERT INTO `progetto1`.`login` (`username`, `passw`, `ruolo`) VALUES ('matteo',
'ce86d7d02a229acfaca4b63f01a1171b', 'amministratore');
```

```
COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`impiegato`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`impiegato` (`CF`, `Nome`, `Recapito`, `TitolodiStudio`, `username`, `Cognome`) VALUES ('SMI89DMRD007', 'Mario', '3348993006', 'Laurea in Medicina', 'mario98', 'Rossi');
INSERT INTO `progetto1`.`impiegato` (`CF`, `Nome`, `Recapito`, `TitolodiStudio`, `username`, `Cognome`) VALUES ('CTF0483IMN42', 'Marco', '3278977455', 'Laurea in Ingegneria ', 'marco72', 'Verdi');
INSERT INTO `progetto1`.`impiegato` (`CF`, `Nome`, `Recapito`, `TitolodiStudio`, `username`, `Cognome`) VALUES ('FRCMLN893IMN42', 'Franco', '3457898231', 'Diploma', 'franco', 'Magno');
INSERT INTO `progetto1`.`impiegato` (`CF`, `Nome`, `Recapito`, `TitolodiStudio`, `username`, `Cognome`) VALUES ('GSPXXS986HJK52', 'Giuseppe', '3246789002', 'Diploma', 'giuseppe', 'Filiberto');
INSERT INTO `progetto1`.`impiegato` (`CF`, `Nome`, `Recapito`, `TitolodiStudio`, `username`, `Cognome`) VALUES ('LGXFDR993HGN58', 'Luigi', '3339944002', 'Terza media', 'luigi', 'Griguoli');
INSERT INTO `progetto1`.`impiegato` (`CF`, `Nome`, `Recapito`, `TitolodiStudio`, `username`, `Cognome`) VALUES ('FBXMLN89HDS78', 'Fabio', 'fabio@gmail.com', 'Laurea', 'fabio', 'Tarantino');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`impiego`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('SMI89DMRD007', '2021-04-01', '2021-05-01', 'prova', 1);
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('CTF0483IMN42', '2021-04-01', '2021-05-01', 'magaziniere', 1);
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('CTF0483IMN42', '2021-05-01', NULL, 'magaziniere', 3);
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('FRCMLN893IMN42', '2021-06-01', NULL, 'magaziniere', 2);
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('GSPXXS986HJK52', '2021-04-01', '2021-05-01', 'cassiere', 3);
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('GSPXXS986HJK52', '2021-05-01', NULL, 'cassiere', 1);
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('LGXFDR993HGN58', '2021-05-01', '2021-06-01', 'magaziniere', 1);
```

```
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('LGXFDR993HGN58', '2021-06-01', NULL, 'cassiere', 1);
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('FBXMLN89HDS78', '2021-05-01', NULL, 'cassiere', 3);
INSERT INTO `progetto1`.`impiego` (`Impiegato`, `DataInizio`, `DataFine`, `Ruolo`, `Centro`)
VALUES ('SMI89DMRD007', '2021-05-01', NULL, 'cassiere', 1);

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`noleggia`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`noleggia` (`Cliente`, `FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `DataScadenza`) VALUES ('CCCMTT99H15I676', 'Pensavo fosse amore... invece
era un calesse', 'Massimo Troisi', 3, 1, '2021-06-28');
INSERT INTO `progetto1`.`noleggia` (`Cliente`, `FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `DataScadenza`) VALUES ('CCCMCC72H5I676', 'Pensavo fosse amore... invece
era un calesse', 'Massimo Troisi', 3, 1, '2021-06-19');
INSERT INTO `progetto1`.`noleggia` (`Cliente`, `FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `DataScadenza`) VALUES ('CCCMTT99H15I676', 'Harry Potter e la pietra
filosofale', 'Chris Columbus', 1, 3, '2021-07-01');
INSERT INTO `progetto1`.`noleggia` (`Cliente`, `FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `DataScadenza`) VALUES ('BNMSRI98G47J702', 'Pinocchio', 'Matteo Garrone', 1,
3, '2021-06-15');
INSERT INTO `progetto1`.`noleggia` (`Cliente`, `FilmTitolo`, `FilmRegista`, `SettoreCodice`,
`SettoreCentro`, `DataScadenza`) VALUES ('BNMSRI98G47J702', 'Harry Potter e la pietra
filosofale', 'Chris Columbus', 3, 1, '2021-06-21');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`turno di lavoro`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-04-01', 'SMI89DMRD007', '2021-05-01');
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-05-01', 'SMI89DMRD007', '2021-06-01');
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-06-01', 'SMI89DMRD007', NULL);
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-04-01', 'CTF0483IMN42', '2021-05-01');
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-05-01', 'CTF0483IMN42', '2021-06-01');
```

```
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-06-01', 'CTF0483IMN42', NULL);
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-06-01', 'FRCMLN893IMN42', NULL);
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-04-01', 'GSPXXS986HJK52', '2021-05-01');
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-05-01', 'GSPXXS986HJK52', '2021-06-01');
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-06-01', 'GSPXXS986HJK52', NULL);
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-05-01', 'LGXFDR993HGN58', '2021-06-01');
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-06-01', 'LGXFDR993HGN58', NULL);
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-05-01', 'FBXMLN89HDS78', '2021-06-01');
INSERT INTO `progetto1`.`turno di lavoro` (`DataInizio`, `Impiegato`, `DataFine`) VALUES
('2021-06-01', 'FBXMLN89HDS78', NULL);

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`orariogiornaliero`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Monday', 'SMI89DMRD007', '2021-04-01', '08:30', '10:30');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Monday', 'SMI89DMRD007', '2021-05-01', '08:30', '10:30');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Wednesday', 'SMI89DMRD007', '2021-05-01', '15:00', '20:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Monday', 'SMI89DMRD007', '2021-06-01', '08:30', '10:30');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Tuesday', 'SMI89DMRD007', '2021-06-01', '08:30', '15:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Wednesday', 'CTF0483IMN42', '2021-04-01', '08:30', '15:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Tuesday', 'CTF0483IMN42', '2021-04-01', '08:30', '16:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Thursday', 'CTF0483IMN42', '2021-05-01', '12:00', '20:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Friday', 'CTF0483IMN42', '2021-05-01', '10:00', '16:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Tuesday', 'CTF0483IMN42', '2021-06-01', '10:00', '16:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`,
`OraInizio`, `OraFine`) VALUES ('Thursday', 'CTF0483IMN42', '2021-06-01', '12:00', '18:00');
```

```
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Monday', 'FRCMLN893IMN42', '2021-06-01', '10:00', '17:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Friday', 'FRCMLN893IMN42', '2021-06-01', '12:00', '18:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Monday', 'GSPXXS986HJK52', '2021-04-01', '10:00', '15:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Friday', 'GSPXXS986HJK52', '2021-04-01', '12:00', '16:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Monday', 'GSPXXS986HJK52', '2021-05-01', '10:00', '18:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Thursday', 'GSPXXS986HJK52', '2021-06-01', '10:00', '17:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Wednesday', 'GSPXXS986HJK52', '2021-06-01', '08:30',
'12:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Monday', 'LGXFDR993HGN58', '2021-05-01', '08:30', '12:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Friday', 'LGXFDR993HGN58', '2021-05-01', '12:00', '16:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Wednesday', 'LGXFDR993HGN58', '2021-06-01', '10:00',
'17:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Thursday', 'LGXFDR993HGN58', '2021-06-01', '08:30', '12:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Monday', 'FBXMLN89HDS78', '2021-05-01', '08:30', '17:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Monday', 'FBXMLN89HDS78', '2021-06-01', '12:00', '17:00');
INSERT INTO `progetto1`.`orariogiornaliero` (`Giorno`, `TurnoDiLavoroI`, `TurnoDiLavoroData`, `OraInizio`, `OraFine`) VALUES ('Friday', 'FBXMLN89HDS78', '2021-06-01', '17:00', '20:00');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`telefonocentro`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`telefonocentro` (`Centro`, `Numero`) VALUES (1, '3920569863');
INSERT INTO `progetto1`.`telefonocentro` (`Centro`, `Numero`) VALUES (2, '3272838525');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`telefonocliente`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
```

```
INSERT INTO `progetto1`.`telefonocliente` (`Cliente`, `Numero`) VALUES
('CCCMTT99H15I676', '3242890543');
INSERT INTO `progetto1`.`telefonocliente` (`Cliente`, `Numero`) VALUES
('CCCMTT99H15I676', '3672910543');
INSERT INTO `progetto1`.`telefonocliente` (`Cliente`, `Numero`) VALUES
('CCCMCC72H5I676', '3334445555');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`emailcliente`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`emailcliente` (`Cliente`, `Email`) VALUES ('CCCMTT99H15I676',
'matteo@gmail.com');
INSERT INTO `progetto1`.`emailcliente` (`Cliente`, `Email`) VALUES ('BNMSRI98G47J702',
'siria@outlook.it');
INSERT INTO `progetto1`.`emailcliente` (`Cliente`, `Email`) VALUES ('CCCMTT99H15I676',
'matteo@outlook.it');
INSERT INTO `progetto1`.`emailcliente` (`Cliente`, `Email`) VALUES ('CCCMCC72H5I676',
'marcello@gmail.com');

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`reportOre`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-04-01', 'CTF0483IMN42', 50, '2021-05-01');
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-05-01', 'CTF0483IMN42', 100, '2021-06-01');
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-06-01', 'CTF0483IMN42', 30, NULL);
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-04-01', 'SMI89DMRD007', 60, '2021-05-01');
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-05-01', 'SMI89DMRD007', 100, '2021-06-01');
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-06-01', 'SMI89DMRD007', 30, NULL);
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-06-01', 'FRCMLN893IMN42', 25, NULL);
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-04-01', 'GSPXXS986HJK52', 40, '2021-05-01');
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-05-01', 'GSPXXS986HJK52', 60, '2021-06-01');
```

```
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-06-01', 'GSPXXS986HJK52', 30, NULL);
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-05-01', 'LGXFDR993HGN58', 45, '2021-06-01');
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-06-01', 'LGXFDR993HGN58', 10, NULL);
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-05-01', 'FBXMLN89HDS78', 200, '2021-06-01');
INSERT INTO `progetto1`.`reportOre` (`DataInizio`, `Impiegato`, `Ore`, `DataFine`) VALUES
('2021-06-01', 'FBXMLN89HDS78', 30, NULL);

COMMIT;


-- -----------------------------------------------------
-- Data for table `progetto1`.`indirizzo cliente`
-- -----------------------------------------------------
START TRANSACTION;
USE `progetto1`;
INSERT INTO `progetto1`.`indirizzo cliente` (`Cliente`, `Indirizzo`) VALUES
('CCCMTT99H15I676', 'Via a caso');
INSERT INTO `progetto1`.`indirizzo cliente` (`Cliente`, `Indirizzo`) VALUES
('BNMSRI98G47J702', 'Via tizio caio sempronio');
INSERT INTO `progetto1`.`indirizzo cliente` (`Cliente`, `Indirizzo`) VALUES
('CCCMCC72H5I676', 'Via verdi');
INSERT INTO `progetto1`.`indirizzo cliente` (`Cliente`, `Indirizzo`) VALUES
('CCCMTT99H15I676', 'Via betulla');
INSERT INTO `progetto1`.`indirizzo cliente` (`Cliente`, `Indirizzo`) VALUES
('CCCMCC72H5I676', 'Via rossi');

COMMIT;
```

Codice del Front-End

Il codice in c è strutturato in più file quindi si riporta il codice per ogni file preceduto dal nome del file stesso.

File inout.c

```
#include<stdio.h>
#include<stdlib.h>
#include <windows.h>
#include <stdbool.h>
#include "defines.h"
int getInput(int lenght, char* string, bool hide) {
    const char BACKSPACE = 8;
    const char RETURN = 13;
    unsigned char ch = 0;
```

```
    DWORD con_mode;
    DWORD dwRead;
    int i = 0;
    HANDLE hIn = GetStdHandle(STD_INPUT_HANDLE);

    GetConsoleMode(hIn, &con_mode);
    SetConsoleMode(hIn, con_mode & ~(ENABLE_ECHO_INPUT | ENABLE_LINE_INPUT));
insert:     while (ReadConsoleA(hIn, &ch, 1, &dwRead, NULL) && ch != RETURN) {
            if (ch == BACKSPACE)
            {
                if (i != 0) {
                    printf("\b \b");
                    string[i - 1] = '\0';
                    i--;
                }
            }
            else {
                if (i < lenght) {
                    string[i] = ch;
                    if (hide)
                        printf("*");
                    else
                        printf("%c", ch);
                    i++;
                }
            }

        }
    printf("\n");
    if (i == 0) {
        printf("You must enter at least one character\n");
        goto insert;
    }
    string[i] = '\0';
    SetConsoleMode(hIn, con_mode);
    if (strlen(string) <= 1)
        return 0;
    return 1;
}
char multiChoice(char* question, char options[], int num_Options) {
    int i = 0, j = 0;
    char* answer = (char*)malloc(sizeof(char) * 2);
    char* poss = (char*)malloc(sizeof(char) * num_Options + 2);
    for (i = 0; i < num_Options; i++) {
        poss[j] = options[i];
        j++;
        poss[j] = '/';
        j++;
    }
    poss[j - 1] = '\0';
```

```
    while (true) {
        printf("\n %s: \n[%s]\n", question, poss);
        getInput(1, answer, false);
        for (i = 0; i < num_Options; i++) {
            if (answer[0] == options[i])
                return answer[0];
        }
        printf("Stick to the options provided\n");
    }
}
    int inserisciOrario(char* question, char* string, int length) {
        printf(question);
        const char BACKSPACE = 8;
        int found = 0;
        const char RETURN = 13;
        unsigned char ch = 0;
        DWORD con_mode;
        DWORD dwRead;
        int i = 0;
        HANDLE hIn = GetStdHandle(STD_INPUT_HANDLE);
        int puntini = 1;
        GetConsoleMode(hIn, &con_mode);
        SetConsoleMode(hIn, con_mode & ~(ENABLE_ECHO_INPUT | ENABLE_LINE_INPUT));
loop:       while (ReadConsoleA(hIn, &ch, 1, &dwRead, NULL) && ch != RETURN) {
            if ((ch > 57 || ch < 48) && ch!= BACKSPACE) {
                //Do nothing
            }
            else {
                if (ch == BACKSPACE)
                {
                    if (i == 4) {
                        printf("\b \b");
                        i--;
                    }
                    if (i != 0) {
                        printf("\b \b");
                        string[i - 1] = '\0';
                        i--;
                    }
                }
                else {
                    if (i == 2) {
                        string[i] = ':';
                        i++;
                        puntini = 0;
                    }
                    if (!puntini) {
                        printf(":");
                        puntini = 1;
                    }
```

```
                printf("%c", ch);
                if (i == 5) {
                    printf("\b \b");
                }
                else {
                    string[i] = ch;
                    i++;
                }
            }
        }
    }
    printf("\n");
    if (i < 5)
        string[i] = '\0';
    else
        string[5] = '\0';
    i = 0;
    while (string[i] != '\0') {
        if (string[i] == ':')
            found = 1;
        i++;
    }
    if (!found) {
        i = 0;
        printf("Please insert a valid date (you must type at least 3 number)\n");
        goto loop;
    }
    SetConsoleMode(hIn, con_mode);
    if (strlen(string) <= 1)
        return 0;
    return 1;
}
  int yesOrNo(char* question) {
    printf("%s\n", question);
    printf("Yes (Y) or No (N)?\n");
    char risp[2];
    do {
        getInput(1, risp, false);
        if (risp[0] == 'N' || risp[0] == 'n') {
            return 0;
        }
        if (risp[0] == 'Y' || risp[0] == 'y')
            return 1;
        printf("Please stick to the directions\n");
    } while (true);
  }
  void leggiNumeri(int length, char* string) {
redo:  getInput(length, string, false);
    int c = 0;
    for (c = 0; c < strlen(string); c++) {
```

```
        if (string[c] < 48 || string[c]>57) {
            printf("You have entered alphabetic characters\n Error\n");
            goto redo;
        }
    }
}
int areYouSure() {
    return yesOrNo("\nAre you sure you want to continue with this operation?\n");
}
```

File utils.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "defines.h"
void _dump_result_set(MYSQL* conn, MYSQL_STMT* stmt, char* title);

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, HANDLE handle)
{
    MYSQL_FIELD *field;
    unsigned int i, j;
    DWORD dummy;
    mysql_field_seek(res_set, 0);
    WriteFile(handle, (LPVOID)"+", strlen("+"), &dummy, NULL);
    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++) {
            WriteFile(handle, (LPVOID)"-", strlen("-"), &dummy, NULL);
        }
        WriteFile(handle, (LPVOID)"+", strlen("+"), &dummy, NULL);
    }
    WriteFile(handle, (LPVOID)"\n", strlen("\n"), &dummy, NULL);
```

```
}

static void dump_result_set_header(MYSQL_RES *res_set, HANDLE handle)
{
   MYSQL_FIELD *field;
   unsigned long col_len;
   unsigned int i;
   char outputBuffer[1024];
   DWORD dummy;
   /* 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,handle);
   WriteFile(handle, (LPVOID)"|", strlen("|"), &dummy, NULL);
   mysql_field_seek (res_set, 0);
   for (i = 0; i < mysql_num_fields(res_set); i++) {
      field = mysql_fetch_field(res_set);
      sprintf(outputBuffer, " %-*s |", (int)field->max_length, field->name);
      WriteFile(handle, outputBuffer, strlen(outputBuffer), &dummy, NULL);
   }
   WriteFile(handle, (LPVOID)"\n", strlen("\n"), &dummy, NULL);

   print_dashes(res_set,handle);
}
noleggi* dump_noleggi(MYSQL* conn, MYSQL_STMT* stmt, char* title) {
   MYSQL_BIND results[5];
   noleggi* head = NULL;
   noleggi* next_row;
   noleggi* prev_row = NULL;
   MYSQL_FIELD* fields;
   MYSQL_RES* rs_metadata;
   MYSQL_TIME* date;
   int num_fields;
   int seq = 0;
   int status;
   int i;
   if (mysql_stmt_store_result(stmt)) {
```

```
    fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
    fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
    exit(0);
  }
  num_fields = mysql_stmt_field_count(stmt);
  if (num_fields > 0) {
    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,GetStdHandle(STD_OUTPUT_HANDLE));
    fields = mysql_fetch_fields(rs_metadata);
    memset(results, 0, sizeof(results));
    results[0].buffer = malloc(sizeof(char) * 46);
    results[0].buffer_length = 46;
    results[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    results[1].buffer = malloc(sizeof(char) * 46);
    results[1].buffer_length = 46;
    results[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    results[2].buffer = malloc(sizeof(char)*46);
    results[2].buffer_length = 46;
    results[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    results[3].buffer = malloc(sizeof(int) + 1);
    results[3].buffer_length = sizeof(int) + 1;
    results[3].buffer_type = MYSQL_TYPE_LONG;
    results[4].buffer = malloc(sizeof(MYSQL_TIME)+1);
    results[4].buffer_length = sizeof(MYSQL_TIME)+1;
    results[4].buffer_type = MYSQL_TYPE_DATE;
    if (mysql_stmt_bind_result(stmt, results)) {
      finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n", true);
    }
    while (true) {
      status = mysql_stmt_fetch(stmt);
      if (status == 1 || status == MYSQL_NO_DATA)
        break;
      putchar('|');
      printf("%d)", seq);
      next_row = (noleggi*)malloc(sizeof(noleggi));
      next_row->next = NULL;
      if (head == NULL)
        head = next_row;
      for (i = 0; i < 5; i++) {
        switch (i) {
        case 0:
          strcpy(next_row->cliente, (char*)results[i].buffer);
          printf(" %-*s |", (int)fields[i].max_length, (char*)results[i].buffer);
          break;
        case 1:
          strcpy(next_row->title, (char*)results[i].buffer);
```

```
                printf(" %-*s |", (int)fields[i].max_length, (char*)results[i].buffer);
                break;
            case 2:
                strcpy(next_row->regista, (char*)results[i].buffer);
                printf(" %-*s |", (int)fields[i].max_length, (char*)results[i].buffer);
                break;
            case 3:
                next_row->settore = *(int*)results[i].buffer;
                printf(" %-*d |", (int)fields[i].max_length, *(int*)results[i].buffer);
                break;
            case 4:
                strcpy(next_row->data, (char*)results[i].buffer);
                date = (MYSQL_TIME*)results[i].buffer;
                printf(" %d-%02d-%02d |", date->year, date->month, date->day);
                break;
            }
        }
        if (prev_row != NULL)
            prev_row->next = next_row;
        prev_row = next_row;
        putchar('\n');
        print_dashes(rs_metadata, GetStdHandle(STD_OUTPUT_HANDLE));
        seq++;
    }
    mysql_free_result(rs_metadata); /* free metadata */
    for (; mysql_next_result(conn) == 0;) {
    }
    /* free output buffers */
    for (i = 0; i < num_fields; i++) {
        free(results[i].buffer);
    }
    return head;

  }
  return NULL;
}
void free_list_film(list_film* head) {
  list_film* temp;
  list_film* prev;
  temp = head;
  while (temp != NULL) {
    prev = temp;
    temp = temp->next;
    free(prev);
  }
}
void free_list_noleggi(noleggi* head) {
  noleggi* temp;
  noleggi* prev;
  temp = head;
```

```
    while (temp != NULL) {
        prev = temp;
        temp = temp->next;
        free(prev);
    }
}
list_film* dump_result_set_film(MYSQL* conn, MYSQL_STMT* stmt, char* title) {
    MYSQL_BIND results[6];
    list_film* head = NULL;
    list_film* next_row;
    list_film* prev_row = NULL;
    MYSQL_FIELD* fields;
    MYSQL_RES* rs_metadata;
    int num_fields;
    int seq = 0;
    int status;
    int i;
    if (mysql_stmt_store_result(stmt)) {
        fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
        exit(0);
    }
    num_fields = mysql_stmt_field_count(stmt);
    if (num_fields > 0) {
        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, GetStdHandle(STD_OUTPUT_HANDLE));
        fields = mysql_fetch_fields(rs_metadata);
        memset(results, 0, sizeof(results));
        results[0].buffer = malloc(sizeof(char)*46);
        results[0].buffer_length = 46;
        results[0].buffer_type = MYSQL_TYPE_VAR_STRING;
        results[1].buffer = malloc(sizeof(char) * 46);
        results[1].buffer_length = 46;
        results[1].buffer_type = MYSQL_TYPE_VAR_STRING;
        results[2].buffer = malloc(sizeof(int)+1);
        results[2].buffer_length = sizeof(int)+1;
        results[2].buffer_type = MYSQL_TYPE_LONG;
        results[3].buffer = malloc(sizeof(int) + 1);
        results[3].buffer_length = sizeof(int) + 1;
        results[3].buffer_type = MYSQL_TYPE_LONG;
        results[4].buffer = malloc(sizeof(char) * 46);
        results[4].buffer_length = 46;
        results[4].buffer_type = MYSQL_TYPE_VAR_STRING;
        results[5].buffer = malloc(sizeof(double) + 1);
        results[5].buffer_type = MYSQL_TYPE_DOUBLE;
        results[5].buffer_length = sizeof(float) + 1;
```

```
if (mysql_stmt_bind_result(stmt, results)) {
    finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n", true);
}
while (true) {
    status = mysql_stmt_fetch(stmt);
    if (status == 1 || status == MYSQL_NO_DATA)
        break;
    putchar('|');
    printf("%d)", seq);
    next_row = (list_film*)malloc(sizeof(list_film));
    next_row->next = NULL;
    if (head == NULL)
        head = next_row;
    for (i = 0; i < 5; i++) {
        switch (i) {
        case 0:
            strcpy(next_row->title, (char*)results[i].buffer);
            printf(" %-*s |", (int)fields[i].max_length, (char*)results[i].buffer);
            break;
        case 1:
            strcpy(next_row->regista, (char*)results[i].buffer);
            printf(" %-*s |", (int)fields[i].max_length, (char*)results[i].buffer);
            break;
        case 2:
            next_row->settore = *(int*)results[i].buffer;
            printf(" %-*d |", (int)fields[i].max_length, *(int*)results[i].buffer);
            break;
        case 3:
            next_row->copie = *(int*)results[i].buffer;
            printf(" %-*d |", (int)fields[i].max_length, *(int*)results[i].buffer);
            break;
        case 4:
            strcpy(next_row->posizione,(char*)results[i].buffer);
            printf(" %-*s |", (int)fields[i].max_length, (char*)results[i].buffer);
            break;
        case 5:
            printf(" %-*f |", (int)fields[i].max_length, *(float*)results[i].buffer);
            break;
        }
    }
    if (prev_row != NULL)
        prev_row->next = next_row;
    prev_row = next_row;
    putchar('\n');
    print_dashes(rs_metadata, GetStdHandle(STD_OUTPUT_HANDLE));
    seq++;
}
mysql_free_result(rs_metadata); /* free metadata */
for (; mysql_next_result(conn) == 0;) {
}
```

```
        /* free output buffers */
        for (i = 0; i < num_fields; i++) {
            free(results[i].buffer);
        }
        return head;

    }
    return NULL;
}
int dump_result_set_with_list(MYSQL* conn, MYSQL_STMT* stmt, char* title,char** list, int
maxRow) {
    int i;
    char stringa[1024];
    int seq = 0;
    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;
    if (mysql_stmt_store_result(stmt)) {
        fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
        exit(0);
    }
    num_fields = mysql_stmt_field_count(stmt);
    if (num_fields > 0) {
        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, GetStdHandle(STD_OUTPUT_HANDLE));

        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_NEWDECIMAL:
            attr_size = sizeof(long double);
            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;
```

```
if (seq + 1 > maxRow) {
   char** temp = list;
   list = (char**)malloc(sizeof(char*) * 3 * seq);
   int count = 0;
   for (count = 0; count < maxRow; count++) {
      list[count] = (char*)malloc(sizeof(char) * strlen(temp[count]));
      strcpy(list[count], temp[count]);
   }
}
putchar('|');
printf("%d)", seq);
for (i = 0; i < num_fields; i++) {

   if (rs_bind[i].is_null_value) {
      printf(" %-*s |", (int)fields[i].max_length, "NULL");
      continue;
   }
   if (i == 0) {
      list[seq] = (char*)malloc(sizeof(char) * strlen((char*)rs_bind[i].buffer));
      strcpy(list[seq], (char*)rs_bind[i].buffer);
   }
   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_TIME:
      date = (MYSQL_TIME*)rs_bind[i].buffer;
      printf("    %d:%d       |", date->hour, date->minute);
      break;
   case MYSQL_TYPE_YEAR:
      date = (MYSQL_TIME*)rs_bind[i].buffer;
      printf("%d|", date->year);
      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(" %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
               break;

            default:
               printf("ERROR: Unhandled type (%d)\n", rs_bind[i].buffer_type);
               abort();
            }
         }
         putchar('\n');
         print_dashes(rs_metadata, GetStdHandle(STD_OUTPUT_HANDLE));
         seq++;
      }

      mysql_free_result(rs_metadata); /* free metadata */
      for (; mysql_next_result(conn) == 0;) {
      }
      /* free output buffers */
      for (i = 0; i < num_fields; i++) {
         free(rs_bind[i].buffer);
      }
      free(rs_bind);
   }
   return seq;
}
void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)
{
   HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
   _dump_result_set(conn, stmt, title,hOut);
      for (; mysql_next_result(conn) == 0;) {
      }
}
void dump_multiple_rs(MYSQL* conn, MYSQL_STMT* stmt, char* title,HANDLE handle) {
   _dump_result_set(conn, stmt, title,handle);
}
void _dump_result_set(MYSQL* conn, MYSQL_STMT* stmt, char* title, HANDLE handle) {
   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;
```

```
DWORD dummy;
char outputBuffer[4096];
/* 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 */
   WriteFile(handle, (LPVOID)title, strlen(title), &dummy, NULL);
   WriteFile(handle, (LPVOID)"\n", strlen("\n"), &dummy, NULL);

   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,handle);
   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_NEWDECIMAL:
            attr_size = sizeof(long double);
            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;
        WriteFile(handle, (LPVOID)"|", strlen("|"), &dummy, NULL);

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

            if (rs_bind[i].is_null_value) {
```

```
        sprintf(outputBuffer, " %-*s |", (int)fields[i].max_length, "NULL");
        WriteFile(handle, outputBuffer, strlen(outputBuffer), &dummy, NULL);
        continue;
    }

    switch (rs_bind[i].buffer_type) {

    case MYSQL_TYPE_VAR_STRING:
    case MYSQL_TYPE_DATETIME:
        sprintf(outputBuffer, " %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
        WriteFile(handle, (LPVOID)outputBuffer, strlen(outputBuffer), &dummy, NULL);
        break;

    case MYSQL_TYPE_DATE:
    case MYSQL_TYPE_TIMESTAMP:

        date = (MYSQL_TIME*)rs_bind[i].buffer;
        sprintf(outputBuffer, " %d-%02d-%02d |", date->year, date->month, date->day);
        WriteFile(handle, (LPVOID)outputBuffer, strlen(outputBuffer), &dummy, NULL);
        break;
    case MYSQL_TYPE_TIME:
        date = (MYSQL_TIME*)rs_bind[i].buffer;
        sprintf(outputBuffer, "     %d:%d         |", date->hour, date->minute);
        WriteFile(handle, (LPVOID)outputBuffer, strlen(outputBuffer), &dummy, NULL);
        break;
    case MYSQL_TYPE_YEAR:
        date = (MYSQL_TIME*)rs_bind[i].buffer;
        sprintf(outputBuffer, "%hu|", date->year);
        WriteFile(handle, (LPVOID)outputBuffer, strlen(outputBuffer), &dummy, NULL);
        break;
    case MYSQL_TYPE_STRING:
        sprintf(outputBuffer, " %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
        WriteFile(handle, (LPVOID)outputBuffer, strlen(outputBuffer), &dummy, NULL);
        break;

    case MYSQL_TYPE_FLOAT:
    case MYSQL_TYPE_DOUBLE:
        sprintf(outputBuffer, " %.02f |", *(double*)rs_bind[i].buffer);
        WriteFile(handle, (LPVOID)outputBuffer, strlen(outputBuffer), &dummy, NULL);
        break;

    case MYSQL_TYPE_LONG:
    case MYSQL_TYPE_SHORT:
    case MYSQL_TYPE_TINY:
        sprintf(outputBuffer, " %-*d |", (int)fields[i].max_length, *(int*)rs_bind[i].buffer);
        WriteFile(handle, (LPVOID)outputBuffer, strlen(outputBuffer), &dummy, NULL);
        break;

    case MYSQL_TYPE_NEWDECIMAL:
        sprintf(outputBuffer, " %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
```

```
                WriteFile(handle, (LPVOID)outputBuffer, strlen(outputBuffer), &dummy, NULL);
                break;

            default:
                printf("ERROR: Unhandled type (%d)\n", rs_bind[i].buffer_type);
                abort();
            }
        }
        WriteFile(handle, (LPVOID)"\n", strlen("\n"), &dummy, NULL);
        print_dashes(rs_metadata,handle);
    }

    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);

  }
}
```

File employer.c

```
#include"defines.h"
#include<stdio.h>
#include<stdlib.h>
char myUsername[46];
#define MAX_EMAIL_CLIENTE 200;
#define MAX_TELEFONO_CLIENTE 100;
void update_copy(MYSQL* conn) {
   MYSQL_STMT* preparedStmt;
   MYSQL_BIND param[5];
   MYSQL_BIND param1[1];
   MYSQL_STMT* preparedStmt2;
   MYSQL_TIME* date = (MYSQL_TIME*)malloc(sizeof(MYSQL_TIME));
   list_film* head = NULL;
   list_film* temp;
   if (date == NULL) {
      printf("Errore malloc\n");
      exit(0);
   }
   memset(date, 0, sizeof(date));
   int numOF = 0;
   char title[61];
   char regista[46];
   int settore;
   char numero[46];
   char settoreI[20];
```

```
int copyN;
char copy[10];
int i = 0;
if (!setup_prepared_stmt(&preparedStmt2, "call visualizzaFilmCentro(?)", conn)) {
    finish_with_stmt_error(conn, preparedStmt2, "Unable to procede\n", false);
}
memset(param1, 0, sizeof(param1));
param1[0].buffer = myUsername;
param1[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param1[0].buffer_length = strlen(myUsername);
if (mysql_stmt_bind_param(preparedStmt2, param1) != 0) {
    finish_with_stmt_error(conn, preparedStmt2, "Unable to procede\n", true);
}
if (mysql_stmt_execute(preparedStmt2) != 0) {
    print_stmt_error(preparedStmt2, "Unable to show film list\n");
    return;
}
else {
    head = dump_result_set_film(conn, preparedStmt2, "Film catalog\n");
    temp = head;
    while (temp != NULL) {
        numOF++;
        temp = temp->next;
    }
}
if (numOF == 0) {
    printf("There are no films\n");
    return;
}
do {
    printf("Enter the number of the movie you want to order\n");
    leggiNumeri(45, numero);
} while (atoi(numero) >= numOF);
temp = head;
i = 0;
while (i < atoi(numero)) {
    temp = temp->next;
    i++;
}
strcpy(title, temp->title);
strcpy(regista, temp->regista);
settore = temp->settore;
mysql_stmt_close(preparedStmt2);
free_list_film(head);
printf("Enter the number of copies to order\n");
leggiNumeri(9, copy);
copyN = atoi(copy);
memset(param, 0, sizeof(param));
param[0].buffer = myUsername;
param[0].buffer_length = strlen(myUsername);
```

```
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = title;
    param[1].buffer_length = strlen(title);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = regista;
    param[2].buffer_length = strlen(regista);
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[3].buffer = &settore;
    param[3].buffer_length = sizeof(settore);
    param[3].buffer_type = MYSQL_TYPE_LONG;
    param[4].buffer = &copyN;
    param[4].buffer_length = sizeof(copyN);
    param[4].buffer_type = MYSQL_TYPE_LONG;
    if (!setup_prepared_stmt(&preparedStmt, "call aggiornaDisponibilita(?,?,?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize request statement\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to order film\n");
        }
        else {
            printf("Properly order movie\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void remove_client(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[1];
    char code[46];
    printf("Enter the customer's tax code\n");
    getInput(45, code, false);
    memset(param, 0, sizeof(param));
    param[0].buffer = code;
    param[0].buffer_length = strlen(code);
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    if (!setup_prepared_stmt(&preparedStmt, "call rimuoviCliente(?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to set the statement\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Cannot set parameters\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to remove");
        }
        else {
```

```
            printf("Customer removed\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void handle_client(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[2];
    char result[61];
    char fiscal_code[46];
    char store_procedure[1024];
    char opt[4] = { '1','2','3','4' };
    char scelta = multiChoice("Choose: 1)Enter new telephone number\n,2) Enter new email\n,3)Enter
new address\n4) Exit", opt, 4);
    switch (scelta) {
    case'1':
        strcpy(store_procedure, "call inserisciTelefonoCliente(?,?)");
        printf("Enter the telephone number\n");
        leggiNumeri(10, result);
        break;
    case '2':
        strcpy(store_procedure, "call inserisciEmailCliente(?,?)");
        printf("Enter the email\n");
        getInput(45, result, false);
        break;
    case '3':
        strcpy(store_procedure, "call inserisciIndirizzoCliente(?,?)");
        printf("Enter the address\n");
        getInput(60, result, false);
        break;
    case '4':
        return;
    }
    printf("Enter the customer's tax code\n");
    getInput(45, fiscal_code, false);
    memset(param, 0, sizeof(param));
    param[1].buffer = fiscal_code;
    param[1].buffer_length = strlen(fiscal_code);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = result;
    param[0].buffer_length = strlen(result);
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    if (!setup_prepared_stmt(&preparedStmt, store_procedure, conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Failed to initialize statement\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Cannot set parameters\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
```

```
            print_stmt_error(preparedStmt, " Unable to insert");
        }
        else {
            printf("Update completed\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
int containsChar(char* string, char character) {
    int c = 0;
    while (string[c] != '\0') {
        if (string[c] == character) {
            return 1;
        }
        c++;
    }
    return 0;
}
void cercaCliente(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[1];
    char cf[46];
    printf("Enter the customer's tax code\n");
    getInput(45, cf, false);
    memset(param, 0, sizeof(param));
    param[0].buffer = cf;
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer_length = strlen(cf);
    if (!setup_prepared_stmt(&preparedStmt, "call trovaCliente(?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to procede\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param ", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, "Unable to found client");
        }
        else {
            printf("CLIENTE: %s\n", cf);
            dump_multiple_rs(conn, preparedStmt, "Customer
data",GetStdHandle(STD_OUTPUT_HANDLE));
            mysql_stmt_next_result(preparedStmt);
            dump_multiple_rs(conn, preparedStmt, "Telephone
numbers:",GetStdHandle(STD_OUTPUT_HANDLE));
            mysql_stmt_next_result(preparedStmt);
            dump_multiple_rs(conn, preparedStmt,
"Email:",GetStdHandle(STD_OUTPUT_HANDLE));
            mysql_stmt_next_result(preparedStmt);
```

```
            dump_multiple_rs(conn, preparedStmt, "Address:",
GetStdHandle(STD_OUTPUT_HANDLE));
            mysql_stmt_next_result(preparedStmt);
            dump_result_set(conn, preparedStmt, "Rentals in progress:");
        }
    }
    mysql_stmt_close(preparedStmt);
}

void rimuoviFilmSettore(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_STMT* preparedStmt2;
    MYSQL_BIND param1[1];
    list_film* head = NULL;
    list_film* temp;
    int numOF = 0;
    int i = 0;
    char numero[46];
    MYSQL_BIND param[4];
    char titolo[61];
    char regista[46];
    char settoreC[10];
    int settore;
    if (!setup_prepared_stmt(&preparedStmt2, "call visualizzaFilmCentro(?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt2, "Unable to procede\n", false);
    }
    memset(param1, 0, sizeof(param1));
    param1[0].buffer = myUsername;
    param1[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param1[0].buffer_length = strlen(myUsername);
    if (mysql_stmt_bind_param(preparedStmt2, param1) != 0) {
        finish_with_stmt_error(conn, preparedStmt2, "Unable to procede\n", true);
    }
    if (mysql_stmt_execute(preparedStmt2) != 0) {
        print_stmt_error(preparedStmt2, "Unable to show film list\n");
        return;
    }
    else {
        head = dump_result_set_film(conn, preparedStmt2, "Catalogo film\n");
        temp = head;
        while (temp != NULL) {
            numOF++;
            temp = temp->next;
        }
    }
    if (numOF == 0) {
        printf("There are no films\n");
        return;
    }
    do {
```

```
        printf("Enter the number of the movie you want to remove\n");
        leggiNumeri(45, numero);
    } while (atoi(numero) >= numOF);
    temp = head;
    i = 0;
    while (i < atoi(numero)) {
        temp = temp->next;
        i++;
    }
    strcpy(titolo, temp->title);
    strcpy(regista, temp->regista);
    settore = temp->settore;
    free_list_film(head);
    mysql_stmt_close(preparedStmt2);
    memset(param, 0, sizeof(param));
    param[0].buffer = titolo;
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer_length = strlen(titolo);
    param[1].buffer = regista;
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer_length = strlen(regista);
    param[2].buffer = &settore;
    param[2].buffer_type = MYSQL_TYPE_LONG;
    param[2].buffer_length = sizeof(settore);
    param[3].buffer = myUsername;
    param[3].buffer_length = strlen(myUsername);
    param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    if (!setup_prepared_stmt(&preparedStmt, "call rimuoviFilmSettore(?,?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to procede\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param ", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, "Unable to remove film");
        }
        else {
            printf("Film successfully removed\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void visualizzaTitoliScaduti(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    if (!setup_prepared_stmt(&preparedStmt, "call stampaFilmScaduti()",conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to procede\n", false);
    }
    if (mysql_stmt_execute(preparedStmt) != 0) {
        print_stmt_error(preparedStmt, " Unable to load list of film\n");
```

```
    }
    else {
        dump_result_set(conn, preparedStmt, "Expired rentals\n");
    }
    mysql_stmt_close(preparedStmt);
}
void restituisciFilm(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    noleggi* head = NULL;
    noleggi* temp;
    int numOF = 0;
    char numero[46];
    MYSQL_BIND param[5];
    char title[61];
    char regista[46];
    int settore;
    char settoreI[20];
    char cliente[46];
    MYSQL_BIND par1[1];
    memset(par1, 0, sizeof(par1));
    par1[0].buffer = myUsername;
    par1[0].buffer_length = strlen(myUsername);
    par1[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    if (!setup_prepared_stmt(&preparedStmt, "call visualizzaNoleggi(?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to procede\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, par1) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param\n", true);
    }
    if (mysql_stmt_execute(preparedStmt) != 0) {
        print_stmt_error(preparedStmt, " Unable to return film\n");
    }
    else {
        head=dump_noleggi(conn, preparedStmt, "Rentals in progress\n");
        if (head == NULL) {
            printf("There are no rentals in progress\n");
            return;
        }
        temp = head;
        while (temp != NULL) {
            numOF++;
            temp = temp->next;
        }
    }
    mysql_stmt_close(preparedStmt);
    if (numOF == 0) {
        printf("There are no films\n");
        return;
    }
    do {
```

```
        printf("Enter the number of the rental to be canceled\n");
        leggiNumeri(45, numero);
    } while (atoi(numero) > numOF);
    numOF = 0;
    temp = head;
    while (numOF < atoi(numero)) {
        temp = temp->next;
    }
    strcpy(cliente, temp->cliente);
    strcpy(title, temp->title);
    strcpy(regista, temp->regista);
    settore = temp->settore;
    free_list_noleggi(head);
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = cliente;
    param[0].buffer_length = strlen(cliente);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = title;
    param[1].buffer_length = strlen(title);
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = regista;
    param[2].buffer_length = strlen(regista);
    param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[4].buffer = myUsername;
    param[4].buffer_length = strlen(myUsername);
    param[3].buffer_type = MYSQL_TYPE_LONG;
    param[3].buffer = &settore;
    param[3].buffer_length = sizeof(settore);
    if (!setup_prepared_stmt(&preparedStmt, "call restituisciFilm(?,?,?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to procede\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to return film\n");
        }
        else {
            printf("Movie returned successfully\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void noleggiaFilm(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[6];
    MYSQL_BIND param1[1];
    MYSQL_STMT* preparedStmt2;
```

```
MYSQL_TIME* date = (MYSQL_TIME*)malloc(sizeof(MYSQL_TIME));
list_film* head = NULL;
list_film* temp;
if (date == NULL) {
    printf("Errore malloc\n");
    exit(0);
}
memset(date, 0, sizeof(date));
int numOF = 0;
char title[61];
char regista[46];
char cliente[46];
int settore;
char numero[46];
char settoreI[20];
char day[3];
char month[3];
char year[5];
int i = 0;
if (!setup_prepared_stmt(&preparedStmt2, "call visualizzaFilmCentro(?)", conn)) {
    finish_with_stmt_error(conn, preparedStmt2, "Unable to procede\n", false);
}
memset(param1, 0, sizeof(param1));
param1[0].buffer = myUsername;
param1[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param1[0].buffer_length = strlen(myUsername);
if (mysql_stmt_bind_param(preparedStmt2, param1) != 0) {
    finish_with_stmt_error(conn, preparedStmt2, "Unable to procede\n", true);
}
if (mysql_stmt_execute(preparedStmt2) != 0) {
    print_stmt_error(preparedStmt2, "Unable to show film list\n");
    return;
}
else {
    head=dump_result_set_film(conn, preparedStmt2, "Film catalog\n");
    temp = head;
    while (temp != NULL) {
        numOF++;
        temp = temp->next;
    }
}
if (numOF == 0) {
    printf("There are no films\n");
    return;
}
do {
    printf("Enter the number of the movie you want to rent\n");
    leggiNumeri(45, numero);
} while (atoi(numero) >= numOF);
temp = head;
```

```
i = 0;
while (i < atoi(numero)) {
    temp = temp->next;
    i++;
}
strcpy(title, temp->title);
strcpy(regista, temp->regista);
settore = temp->settore;
mysql_stmt_close(preparedStmt2);
free_list_film(head);
printf("Enter the customer\n");
getInput(45, cliente, false);
printf("Enter the expiration date\n");
printf("Enter the day\n");
leggiNumeri(2, day);
printf("Enter the month\n");
leggiNumeri(2, month);
printf("Enter the year\n");
leggiNumeri(4, year);
if (!setup_prepared_stmt(&preparedStmt, "call noleggiaFilm(?,?,?,?,?,?)", conn)) {
    finish_with_stmt_error(conn, preparedStmt, "Unable to initialize request statement\n", false);
}
date->day = atoi(day);
date->month = atoi(month);
date->year = atoi(year);
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = cliente;
param[0].buffer_length = strlen(cliente);
param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = title;
param[1].buffer_length = strlen(title);
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = regista;
param[2].buffer_length = strlen(regista);
param[3].buffer_type = MYSQL_TYPE_LONG;
param[3].buffer = &settore;
param[3].buffer_length = sizeof(settore);
param[4].buffer_type = MYSQL_TYPE_DATE;
param[4].buffer = date;
param[4].buffer_length = sizeof(date);
param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
param[5].buffer = myUsername;
param[5].buffer_length = strlen(myUsername);
if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
    finish_with_stmt_error(conn, preparedStmt, "Unable to bind param\n", true);
}
if (areYouSure()) {
    if (mysql_stmt_execute(preparedStmt) != 0) {
        print_stmt_error(preparedStmt, " Unable to rent film\n");
```

```
        }
        else {
            printf("Properly rented movie\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void inserisciFilm(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[6];
    MYSQL_BIND param1[2];
    char titolo[61];
    char regista[46];
    int settoreI;
    char settore[20];
    int copie;
    char copieC[10];
    char posizione[46];
    printf("Enter the sector code\n");
    leggiNumeri(19, settore);
    settoreI = atoi(settore);
    memset(param1, 0, sizeof(param1));
    param1[0].buffer = myUsername;
    param1[0].buffer_length = strlen(myUsername);
    param1[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param1[1].buffer = &settoreI;
    param1[1].buffer_type = MYSQL_TYPE_LONG;
    param1[1].buffer_length = sizeof(settoreI);
    if (!setup_prepared_stmt(&preparedStmt, "call visualizzaFilmMancanti(?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to procede\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param1) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param\n", true);
    }
    if (mysql_stmt_execute(preparedStmt) != 0) {
        print_stmt_error(preparedStmt, "Unable to show film list\n");
    }
    else {
        dump_result_set(conn, preparedStmt, "Films not present in the center\n");
    }
    mysql_stmt_close(preparedStmt);
    MYSQL_STMT* preparedStmt2;
    printf("Enter the title of the film to be recorded\n");
    getInput(60, titolo, false);
    printf("Enter the director of the film\n");
    getInput(45, regista, false);
    printf("Enter the number of copies\n");
    leggiNumeri(9, copieC);
    printf("Enter the location to register it\n");
    getInput(45, posizione, false);
```

```
    copie = atoi(copieC);
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = titolo;
    param[0].buffer_length = strlen(titolo);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = regista;
    param[1].buffer_length = strlen(regista);
    param[2].buffer_type = MYSQL_TYPE_LONG;
    param[2].buffer = &settoreI;
    param[2].buffer_length = sizeof(settoreI);
    param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[3].buffer = myUsername;
    param[3].buffer_length = strlen(myUsername);
    param[4].buffer_type = MYSQL_TYPE_LONG;
    param[4].buffer = &copie;
    param[4].buffer_length = sizeof(copie);
    param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[5].buffer = posizione;
    param[5].buffer_length = strlen(posizione);
    if (!setup_prepared_stmt(&preparedStmt2, "call inserisciFilmSettore(?,?,?,?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt2, "Unable to initialize request statement\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt2, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt2, "Unable to bind param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt2) != 0) {
            print_stmt_error(preparedStmt2, " Unable to insert film\n");
        }
        else {
            printf("Film entered correctly\n");
        }
    }
    mysql_stmt_close(preparedStmt2);
}
void registraCliente(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[7];
    char CF[46];
    char Name[46];
    char Surname[46];
    char day[3];
    char month[3];
    char year[5];
    char address[301];
    MYSQL_TIME* date = (MYSQL_TIME*)malloc(sizeof(MYSQL_TIME));
    strcpy(address, "");
    char var_Telefono[150];
    int goOn = 1;
```

```
    char tempAddress[61];
    char scelta;
    strcpy(var_Telefono, "");
    char var_Email[200];
    strcpy(var_Email, "");
    printf("Enter the customer's tax code\n");
    getInput(45, CF, false);
    printf("Enter the customer's name\n");
    getInput(45, Name, false);
    printf("Enter the customer's surname\n");
    getInput(45, Surname, false);
    printf("Enter the date of birth\n");
    printf("Enter the day\n");
    leggiNumeri(2, day);
    printf("Enter the month\n");
    leggiNumeri(2, month);
    printf("Enter the year\n");
    leggiNumeri(4, year);
    printf("Enter the customer's address\n");
    char optAddress[2] = { '1','2' };
    while (goOn) {
        scelta = multiChoice("Choose \n 1)Enter new address\n 2)End\n", optAddress, 2);
        switch (scelta) {
        case '1':
            do {
                printf("Enter the address, the symbol ; is not allowed \n");
                getInput(60, tempAddress, false);
            } while (containsChar(tempAddress, ';')==1);
            strcat(address, tempAddress);
            strcat(address, ";");
            break;
        case '2':
            goOn = 0;
            break;
        }
    }
        goOn = 1;
    printf("Now enter contact details for the customer\n");
    char temp[46];
    char options[3] = { '1','2','3' };
    while (goOn) {
        scelta = multiChoice("Choose \n 1)Enter telephone number\n 2)Enter email\n 3)End\n", options,
3);
        switch (scelta) {
        case '1':
            printf("Enter the telephone number\n");
            leggiNumeri(15, temp);
            if (strlen(var_Telefono) +strlen(temp)> 100) {
                printf("You have reached the maximum number of phone numbers that can be registered,
you will be able to register others later\n");
```

```
            }
            else {
                strcat(var_Telefono, temp);
                strcat(var_Telefono, ";");
            }
            break;
        case '2':
            printf("Enter the email\n");
            getInput(45, temp, false);
            if(strlen(var_Email)+strlen(temp)>200){
                printf("You have reached the maximum number of emails that can be registered, you will be able to register others later\n");
            }
            else {
                strcat(var_Email, temp);
                strcat(var_Email, ";");
            }
            break;
        case '3':
            goOn = 0;
            break;
        }
    }
    date->day = atoi(day);
    date->month = atoi(month);
    date->year = atoi(year);
    if (!setup_prepared_stmt(&preparedStmt, "call registraCliente(?,?,?,?,?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize insert statement\n", false);
    }
    memset(param, 0, sizeof(param));
    param[0].buffer = CF;
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer_length = strlen(CF);
    param[1].buffer = Name;
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer_length = strlen(Name);
    param[2].buffer = address;
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer_length = strlen(address);
    param[3].buffer = Surname;
    param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[3].buffer_length = strlen(Surname);
    param[4].buffer = var_Telefono;
    param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[4].buffer_length = strlen(var_Telefono);
    param[5].buffer = var_Email;
    param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[5].buffer_length = strlen(var_Email);
    param[6].buffer = date;
    param[6].buffer_type = MYSQL_TYPE_DATE;
```

```
    param[6].buffer_length = sizeof(date);
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to insert client\n");
        }
        else {
            printf("Customer entered correctly\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void timbraCartellino(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[1];
    memset(param, 0, sizeof(param));
    param[0].buffer = myUsername;
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer_length = strlen(myUsername);
    if (!setup_prepared_stmt(&preparedStmt, "call timbra(?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize request statement\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param for statement\n", true);
    }
    if (mysql_stmt_execute(preparedStmt) != 0) {
        print_stmt_error(preparedStmt, " Impossible to stamp the card\n");
    }
    else {
        printf("Operation succeded\n");
    }
    mysql_stmt_close(preparedStmt);
}
void vediTurno(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[1];
    memset(param, 0, sizeof(param));
    param[0].buffer = myUsername;
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer_length = strlen(myUsername);
    if (!setup_prepared_stmt(&preparedStmt, "call visualizzaTurno(?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize request statement\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param for statement\n", true);
    }
    if (mysql_stmt_execute(preparedStmt) != 0) {
        print_stmt_error(preparedStmt, " Unable to execute query for turno di lavoro\n");
```

```
    }
    else {
        dump_result_set(conn, preparedStmt, "Workshift");
    }
    mysql_stmt_close(preparedStmt);
}
void apriMenuCliente(MYSQL* conn) {
    char option[5] = { '1','2','3','4','5' };
    char choice;
    while (true) {
        choice = multiChoice("Customer menu\n Choose the operation: 1)View expired orders\n
2)Search for a customer\n 3) Insert email,telephone number or address\n4)Remove
customer\n5)Return to the main menu\n", option, 5);
        switch (choice) {
        case '1':
            visualizzaTitoliScaduti(conn);
            break;

        case '2':
            cercaCliente(conn);
            break;
        case '3':
            handle_client(conn);
            break;
        case '4':
            remove_client(conn);
        case '5':
            return;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
        }
    }
}
void menu_film(MYSQL* conn) {
    char scelta;
    char opt[3] = { '1','2','3' };
    while (true) {
        scelta = multiChoice("Choose the operation\n: 1)Order new copies for a film\n 2)Order a new
type of film\n 3)Exit\n",opt,3);
        switch (scelta) {
        case '1':
            update_copy(conn);
            break;
        case '2':
            inserisciFilm(conn);
            break;
        case '3':
            return;
        }
```

```
    }
}
void run_as_employer(MYSQL* conn, char* username) {
    strcpy(myUsername, username);
    char risp;
    printf("Switching to employer role...\n");
    int res;
    if (!parse_config("users/employer.json", &conf)) {
        fprintf(stderr, "Unable to load employer 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);
    }
    timbraCartellino(conn);
    char options[9] = { '1','2','3','4','5','6','7','8','9' };
    printf("\033[2J\033[H");
    char* question = "Choose the operation:\n 1) View workshift\n 2) Stamp card\n3)Register
customer\n 4)Rent movies\n5)Record film return\n6)Order movies\n7)Open customer
menu\n8)Remove film from center\n9)exit";
    while (true) {
        risp = multiChoice(question, options, 9);
        switch (risp) {
        case '1':
            vediTurno(conn);
            break;
        case '2':
            timbraCartellino(conn);
            break;
        case '3':
            registraCliente(conn);
            break;
        case '4':
            noleggiaFilm(conn);
            break;
        case '5':
            restituisciFilm(conn);
            break;
        case '6':
            menu_film(conn);
            break;
        case '7' :
            apriMenuCliente(conn);
            break;
        case '8':
            rimuoviFilmSettore(conn);
            break;
        case '9':
```

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

    }
}
```

File manager.c

```
#include "defines.h"
#include<stdio.h>
#include<stdlib.h>
void inserisciTurnoDiLavoro(MYSQL* conn, char* employer);
int is_valid_time(char* time) {
    char hour[3];
    char minute[3];
    int i = 0;
    if (strlen(time) != 5) {
        printf("Insert a valid time\n");
        return 0;
    }
    hour[0] = time[0];
    hour[1] = time[1];
    hour[2] = '\0';
    minute[0] = time[3];
    minute[1] = time[4];
    minute[2] = '\0';
    int hourI = atoi(hour);
    int minuteI = atoi(minute);
    if (hourI >= 0 && hourI <= 23 && minuteI >= 0 && minuteI <= 59) {
        return 1;
    }
    printf("Insert a valid time\n");
    return 0;
}
int parse_date(MYSQL* conn, MYSQL_STMT* stmt, char** date) {
    MYSQL_TIME date_inizio;
    MYSQL_TIME date_concluso;
    MYSQL_BIND param[2];
    int is_null = 0;
    if (mysql_stmt_store_result(stmt)) {
        fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
        fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
        exit(0);
    }
    memset(param, 0, sizeof(param));
```

```
    param[0].buffer_type = MYSQL_TYPE_DATE;
    param[0].buffer = &date_inizio;
    param[0].buffer_length = sizeof(date_inizio);
    param[1].buffer_type = MYSQL_TYPE_DATE;
    param[1].buffer = &date_concluso;
    param[1].buffer_length = sizeof(date_concluso);
    param[1].is_null = &is_null;
    if (mysql_stmt_bind_result(stmt, param)) {
        if (mysql_stmt_next_result == 0) {
            finish_with_stmt_error(conn, stmt, "Unable to bind column parameters\n", true);
        }
        else {
            return 0;
        }
    }
    mysql_stmt_fetch(stmt);
    date[0] = (char*)malloc(sizeof(char) * 12);
    sprintf(date[0], " %d-%02d-%02d", date_inizio.year, date_inizio.month, date_inizio.day);
    if (is_null) {
        date[1] = (char*)malloc(sizeof(char) * strlen("IN PROGRESS")+1);
        strcpy(date[1], "IN PROGRESS");
    }
    else {
        date[1] = (char*)malloc(sizeof(char) * 12);
        sprintf(date[1], " %d-%02d-%02d", date_concluso.year, date_concluso.month,
date_concluso.day);
    }
    return 1;
}
int visualizzaImpiegati(MYSQL* conn, char** cf,int max) {
    MYSQL_STMT* preparedStmt;
    if (!setup_prepared_stmt(&preparedStmt, "call visualizzaImpiegati()", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize employer list", false);
    }
    if (mysql_stmt_execute(preparedStmt) != 0) {
        print_stmt_error(preparedStmt, " Unable to read data\n");
    }
    int res=dump_result_set_with_list(conn, preparedStmt, "Employers\n",cf,max);
    mysql_stmt_close(preparedStmt);
    return res;
}
void rimuoviImpiegato(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[1];
    char username[46];
    char** cf = (char**)malloc(sizeof(char*) * 200);
    if (cf == NULL) {
        printf("Unable to execute malloc\n");
        exit(0);
    }
```

```
    int dim = visualizzaImpiegati(conn, cf, 200);
    if (dim == 0) {
        printf("There are no employers\n");
        return;
    }
    printf("Now insert employer username\n");
    getInput(45, username, false);
    memset(param, 0, sizeof(param));
    param[0].buffer = username;
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer_length = strlen(username);
    if (!yesOrNo("Are you sure you want to delete this employer?"))
        return;
    if (!setup_prepared_stmt(&preparedStmt, "call rimuoviImpiegato(?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize operation", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to bind param ", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, "Unable to remove employer");
        }
        else {
            printf("Employer successfully removed\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void reportAnnuale(MYSQL* conn) {
    int onFile = 0;
    HANDLE hOut = NULL;
    MYSQL_STMT* preparedStmt;
    HANDLE fileHandler = GetStdHandle(STD_OUTPUT_HANDLE);
    MYSQL_BIND param[3];
    int risp = yesOrNo("Do you want to see full list of employer?\n");
    char CF[46];
    char fileName[46];
    char** date = (char**)malloc(sizeof(char*) * 2);
    char header[1024];
    char month[6];
    int turno = 0;
    if (risp == 1) {
        char** cf = (char**)malloc(sizeof(char*) * 200);
        if (cf == NULL) {
            printf("Unable to execute malloc\n");
            exit(0);
        }
        char risposta[1024];
        int dim=visualizzaImpiegati(conn,cf,200);
```

```
    if (dim == 0) {
        printf("There are no employers\n");
        return;
    }
    do {
        printf("Enter the number of the chosen employee\n");
        leggiNumeri(dim, risposta);
    } while (atoi(risposta) >= dim);
    strcpy(CF, cf[atoi(risposta)]);
    free(cf);
}
else {
    printf("Enter the employee's tax code\n");
    getInput(45, CF, false);
}
printf("Enter the year\n");
leggiNumeri(5, month);
int m = atoi(month);
turno = yesOrNo("Do you also want to see all the work shifts for this employee?\n");
memset(param, 0, sizeof(param));
param[0].buffer = CF;
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer_length = strlen(CF);
param[1].buffer_type = MYSQL_TYPE_SHORT;
param[1].buffer = &m;
param[1].buffer_length = sizeof(m);
param[2].buffer_type = MYSQL_TYPE_SHORT;
param[2].buffer = &turno;
param[2].buffer_length = sizeof(turno);
if (!setup_prepared_stmt(&preparedStmt, "call reportAnnuale(?,?,?)", conn)) {
    finish_with_stmt_error(conn, preparedStmt, "Unable to initialize ", false);
}
if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
    finish_with_stmt_error(conn, preparedStmt, "Unable to bind param ", true);
}
if (areYouSure()) {
    if (mysql_stmt_execute(preparedStmt) != 0) {
        print_stmt_error(preparedStmt, "Unable to read data\n");
    }
    else {
        onFile = yesOrNo("Do you want to save results on a file?\n");
        if (onFile) {
            printf("Insert the file name, the file will be saved in the reports directory\n");
            getInput(45, fileName, false);
            char finalPath[60];
            strcpy(finalPath, "reports/");
            strcat(finalPath, fileName);
            fileHandler = CreateFile(
                finalPath,
                GENERIC_READ | GENERIC_WRITE,
```

```
                0,
                NULL,
                CREATE_ALWAYS,
                FILE_ATTRIBUTE_NORMAL,
                NULL
            );
        }
        while (conn->server_status & SERVER_PS_OUT_PARAMS) {
        }
        dump_multiple_rs(conn, preparedStmt, "Annual report\n",fileHandler);
        while (mysql_stmt_next_result(preparedStmt) == 0) {
            if (parse_date(conn, preparedStmt, date)) {
                sprintf(header, "Workshift From:%s to:%s", date[0], date[1]);
                if (mysql_stmt_next_result(preparedStmt) > 0) {
                    finish_with_stmt_error(conn, preparedStmt, "Unexpected condition", true);
                }
                dump_multiple_rs(conn, preparedStmt, header,fileHandler);
            }
            else {
                break;
            }
        }
        if (onFile) {
            CloseHandle(fileHandler);
            printf("Saved\n");
        }
    }
  }
  mysql_stmt_close(preparedStmt);
}
void reportMensile(MYSQL* conn) {
  HANDLE fileHandler = GetStdHandle(STD_OUTPUT_HANDLE);
  int onFile = 0;
  char fileName[46];
  MYSQL_STMT* preparedStmt;
  MYSQL_BIND param[3];
  char year[5];
  char header[1024];
  char** date = (char**)malloc(sizeof(char*) * 2);
  int contatoreTurni = 0;
  char CF[46];
  char month[3];
  int risp = yesOrNo("Do you want to see full list of employer?\n");
  if (risp == 1) {
    char** cf = (char**)malloc(sizeof(char*) * 200);
    if (cf == NULL) {
      printf("Unable to execute malloc\n");
      exit(0);
    }
    char risposta[1024];
```

```
    int dim = visualizzaImpiegati(conn, cf, 200);
    if (dim == 0) {
        printf("There are no employers\n");
        return;
    }
    do {
        printf("Enter the number of the chosen employee\n");
        leggiNumeri(dim, risposta);
    } while (atoi(risposta) >= dim);
    strcpy(CF, cf[atoi(risposta)]);
    free(cf);
}
else {
    printf("Enter the employee's tax code\n");
    getInput(45, CF, false);
}
printf("Enter the month\n");
leggiNumeri(2, month);
printf("Enter the year\n");
leggiNumeri(4, year);
int m = atoi(month);
int y = atoi(year);
printf("%d", m);
memset(param, 0, sizeof(param));
param[0].buffer = CF;
param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer_length = strlen(CF);
param[1].buffer_type = MYSQL_TYPE_SHORT;
param[1].buffer = &m;
param[1].buffer_length = sizeof(m);
param[2].buffer_type = MYSQL_TYPE_SHORT;
param[2].buffer = &y;
param[2].buffer_length = sizeof(y);
if (!setup_prepared_stmt(&preparedStmt, "call reportMensile(?,?,?)", conn)) {
    finish_with_stmt_error(conn, preparedStmt, "Unable to initialize ", false);
}
if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
    finish_with_stmt_error(conn, preparedStmt, "Unable to bind param ", true);
}
if (areYouSure()) {
    if (mysql_stmt_execute(preparedStmt) != 0) {
        print_stmt_error(preparedStmt, " Unable to read data\n");
    }
    else {
        onFile = yesOrNo("Do you want to save results on a file?\n");
        if (onFile) {
            printf("Insert the file name, the file will be saved in the reports directory\n");
            getInput(45, fileName, false);
            char finalPath[60];
            strcpy(finalPath, "reports/");
```

```
            strcat(finalPath, fileName);
            fileHandler = CreateFile(
               finalPath,
               GENERIC_READ | GENERIC_WRITE,
               0,
               NULL,
               CREATE_ALWAYS,
               FILE_ATTRIBUTE_NORMAL,
               NULL
            );
         }
         while (conn->server_status & SERVER_PS_OUT_PARAMS) {
         }
         dump_multiple_rs(conn, preparedStmt, "report Mensile\n",fileHandler);
         while (mysql_stmt_next_result(preparedStmt) == 0) {
            if (parse_date(conn, preparedStmt, date)) {
               sprintf(header, "Workshift From:%s to:%s", date[0], date[1]);
               if (mysql_stmt_next_result(preparedStmt) > 0) {
                  finish_with_stmt_error(conn, preparedStmt, "Unexpected condition", true);
               }
               dump_multiple_rs(conn, preparedStmt, header,fileHandler);
               contatoreTurni++;
            }
            else {
               break;
            }
         }
         if (onFile) {
            CloseHandle(fileHandler);
            printf("Saved\n");
         }
      }
   }
   mysql_stmt_close(preparedStmt);
}
void visualizzaCentri(MYSQL* conn) {
   MYSQL_STMT* preparedStmt;
   if (!setup_prepared_stmt(&preparedStmt, "call visualizzaCentri()", conn)) {
      finish_with_stmt_error(conn, preparedStmt, "Unable to initialize center list", false);
   }
   if (mysql_stmt_execute(preparedStmt) != 0) {
      print_stmt_error(preparedStmt, " Unable to read data\n");
   }
   dump_result_set(conn, preparedStmt, "Centers\n");
   mysql_stmt_close(preparedStmt);
}
int vectorContains(char c, char* vector, int length) {
   int i = 0;
   for (i = 0; i < length; i++) {
      if (c == vector[i])
```

```
            return 1;
        }
        return 0;
}
void leggiInfoTurno(char* days, char* hours,  MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    char scelta;
    char hour[6];
    strcpy(days, "");
    strcpy(hours, "");
    char options[7] = { '1','2','3','4','5','6','7' };
    char giorniScelti[7];
    int count = 0;
    int goOn = 1;
    while (goOn == 1) {
        while (1) {
            scelta = multiChoice("Choose a day
1)Monday\n2)Tuesday\n3)Wednesday\n4)Thursday\n5)Friday\n6)Saturday\n7)Exit", options, 7);
            if (vectorContains(scelta, giorniScelti, 7)) {
                printf("You have already chosen this day\n");
            }
            else {
                break;
            }
        }
        switch (scelta) {
        case '1':
            strcat(days, "Monday;");
            break;
        case '2':
            strcat(days, "Tuesday;");
            break;
        case '3':
            strcat(days, "Wednesday;");
            break;
        case '4':
            strcat(days, "Thursday;");
            break;
        case '5':
            strcat(days, "Friday;");
            break;
        case '6':
            strcat(days, "Saturday;");
            break;
        case '7':
            goOn = 0;
            break;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
```

```
        }
        giorniScelti[count] = scelta;
        count++;
        if (goOn == 1) {
            do {
                inserisciOrario("Enter start time\n", hour, 6);
            } while (!is_valid_time(hour));
            strcat(hours, hour);
            strcat(hours, ";");
            do {
                inserisciOrario("Enter end time\n", hour, 6);
            } while (!is_valid_time(hour));
            strcat(hours, hour);
            strcat(hours, ";");
        }

    }
}
void inserisciTurnoDiLavoro(MYSQL* conn,char* impiegato) {
    MYSQL_STMT* preparedStmt2;
    MYSQL_BIND param[3];

    char days[140];
    char hours[100];
    char employer[46];
    char** cf = (char**)malloc(sizeof(char*) * 200);
    if (cf == NULL) {
        printf("Unable to execute malloc\n");
        exit(0);
    }
    char risposta[1024];
    int dim = visualizzaImpiegati(conn, cf, 200);
    if (dim == 0) {
        printf("There are no employers\n");
        return;
    }
    do {
        printf("Enter the number of the chosen employee\n");
        leggiNumeri(dim, risposta);
    } while (atoi(risposta) >= dim);
    strcpy(employer, cf[atoi(risposta)]);
    free(cf);
    leggiInfoTurno(days, hours,  conn);


    if (!setup_prepared_stmt(&preparedStmt2, "call inserisciTurnoDiLavoro(?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt2, "Unable to initialize turno di lavoro insertion
statement\n", false);
    }
    memset(param, 0, sizeof(param));
```

```
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = employer;
    param[0].buffer_length = strlen(employer);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = days;
    param[1].buffer_length = strlen(days);
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = hours;
    param[2].buffer_length = strlen(hours);
    if (mysql_stmt_bind_param(preparedStmt2, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt2, "Unable to setup param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt2) != 0) {
            print_stmt_error(preparedStmt2, " Unable to execute insert\n");
        }
        else {
            printf("\nShift registered successfully\n");
        }
    }
    mysql_stmt_close(preparedStmt2);
}
void spostaImpiegato(MYSQL* conn, char* empl) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[5];
    char impiegato[46];
    int centro;
    char centroS[20];
    char role[45];
    char days[140];
    char hours[100];
    if (empl == NULL) {
        char** cf = (char**)malloc(sizeof(char*) * 200);
        if (cf == NULL) {
            printf("Unable to execute malloc\n");
            exit(0);
        }
        char risposta[1024];
        int dim = visualizzaImpiegati(conn, cf, 200);
        if (dim == 0) {
            printf("There are no employers\n");
            return;
        }
        do {
            printf("Enter the number of the chosen employee\n");
            leggiNumeri(dim, risposta);
        } while (atoi(risposta) >= dim);
        strcpy(impiegato, cf[atoi(risposta)]);
        free(cf);
    }
```

```
    else {
        strcpy(impiegato, empl);
    }
    visualizzaCentri(conn);
    printf("Insert center code\n");
    leggiNumeri(19, centroS);
    printf("Insert role\n");
    getInput(45, role, false);
    centro = atoi(centroS);
    printf("Now insert work shift\n");
    leggiInfoTurno(days, hours, conn);
    if (!setup_prepared_stmt(&preparedStmt, "call registraImpiego(?,?,?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize employer impiego insertion
statement\n", false);
    }
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = impiegato;
    param[0].buffer_length = strlen(impiegato);
    param[1].buffer_type = MYSQL_TYPE_LONG;
    param[1].buffer = &centro;
    param[1].buffer_length = sizeof(centro);
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = role;
    param[2].buffer_length = strlen(role);
    param[3].buffer = days;
    param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[3].buffer_length = strlen(days);
    param[4].buffer = hours;
    param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[4].buffer_length = strlen(hours);
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to setup param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to execute insert\n");
        }
        else {
            printf("\nJob successfully registered\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}

void registraImpiegato(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[11];
    char name[46];
    char cf[46];
```

```
    char cognome[46];
    char recapito[46];
    char titoloStudio[61];
    char username[46];
    char password[46];
    char centerC[10];
    int center;
    char hours[100];
    char days[140];
    char role[46];
    printf("Insert employer CF\n");
    getInput(45, cf, false);
    printf("Insert employer name\n");
    getInput(45, name, false);
    printf("Inserisci employer surname\n");
    getInput(45, cognome, false);
    char rec[2] = { '1','2' };
    char sceltaR=multiChoice("Now enter contact details for the employee, choose between 1)
Telephone number 2) Email address\n", rec, 2);
    if (sceltaR == '1') {
        printf("Enter the phone number\n");
        leggiNumeri(10, recapito, false);
    }
    else {
        printf("Enter the email address\n");
        getInput(45, recapito, false);
    }
    printf("Insert employer title\n");
    getInput(60, titoloStudio, false);
    printf("Insert employer username\n");
    getInput(45, username, false);
    printf("Insert employer password\n");
    getInput(45, password, true);
    printf("Now insert employer center\n");
    visualizzaCentri(conn);
    leggiNumeri(9, centerC);
    center = atoi(centerC);
    printf("Now insert employer role\n");
    getInput(45, role, false);
    printf("Now insert workshift\n");
    leggiInfoTurno(days, hours, conn);
    if (!setup_prepared_stmt(&preparedStmt, "call registraImpiegato(?,?,?,?,?,?,?,?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize employer insertion
statement\n", false);
    }
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = cf;
    param[0].buffer_length = strlen(cf);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
```

```
    param[1].buffer = name;
    param[1].buffer_length = strlen(name);
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = recapito;
    param[2].buffer_length = strlen(recapito);
    param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[3].buffer = titoloStudio;
    param[3].buffer_length = strlen(titoloStudio);
    param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[4].buffer = username;
    param[4].buffer_length = strlen(username);
    param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[5].buffer = password;
    param[5].buffer_length = strlen(password);
    param[6].buffer = days;
    param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[6].buffer_length = strlen(days);
    param[7].buffer = hours;
    param[7].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[7].buffer_length = strlen(hours);
    param[8].buffer = &center;
    param[8].buffer_type = MYSQL_TYPE_LONG;
    param[8].buffer_length = sizeof(center);
    param[9].buffer = role;
    param[9].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[9].buffer_length = strlen(role);
    param[10].buffer = cognome;
    param[10].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[10].buffer_length = strlen(cognome);
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to setup param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to execute insert\n");
        }
        else {
            printf("\nEmployee entered correctly\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}

void run_as_manager(MYSQL* conn,int isAdmin) {
    char risp;
    char** cf = (char**)malloc(sizeof(char*) * 200);
    if (cf == NULL) {
        printf("Unable to execute malloc\n");
        exit(0);
    }
```

```
    if (!isAdmin) {
        printf("Switching to manager role...\n");
        int res;
        if (!parse_config("users/manager.json", &conf)) {
            fprintf(stderr, "Unable to load manager 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);
        }
    }
    CreateDirectoryA(
        "reports",
        NULL
    );
    char options[8] = { '1','2','3','4','5','6','7','8' };
    printf("\033[2J\033[H");
    char* question = "Choose operation:\n 1) Register Employee \n 2) Move an Employee from office
or assign a new role \n3) Enter work shift \n 4) View monthly report \n5) View annual report \n6)
view Employees \n7) Remove Employee \n8) exit";
    while (true) {
        risp = multiChoice(question, options, 8);
        switch (risp) {
        case '1':
            registraImpiegato(conn);
            break;
        case '2':
            spostaImpiegato(conn,NULL);
            break;
        case '3':
            inserisciTurnoDiLavoro(conn,NULL);
            break;
        case '4':
            reportMensile(conn);
            break;
        case '5':
            reportAnnuale(conn);
            break;
        case '6':
            visualizzaImpiegati(conn, cf, 200);
            break;
        case '7':
            rimuoviImpiegato(conn);
            break;
        case '8':
            return;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
```

```
            abort();
        }

    }
}
```

File administrator.c

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

#include "defines.h"
#define MAX_TEL 100
#define MAX_EMAIL 450
int contains(char* string, char character) {
    int c = 0;
    while (string[c] != '\0') {
        if (string[c] == character) {
            return true;
        }
        c++;
    }
    return false;
}
void handle_center(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[2];
    char result[46];
    char center[40];
    int id;
    char store_procedure[1024];
    char opt[3] = { '1','2','3' };
    char scelta = multiChoice("Scegli 1)Enter new telephone number\n,2)Enter new email\n,3) Exit",
opt, 3);
    switch (scelta) {
    case'1':
        strcpy(store_procedure, "call inserisciTelefonoCentro(?,?)");
        printf("Enter the telephone number\n");
        leggiNumeri(10, result);
        break;
    case '2':
        strcpy(store_procedure, "call inserisciEmailCentro(?,?)");
        printf("Enter the email\n");
        getInput(45, result, false);
        break;
    case '3':
        return;
    }
```

```
    printf("Enter the code of the center\n");
    leggiNumeri(39, center);
    id = atoi(center);
    memset(param, 0, sizeof(param));
    param[0].buffer = &id;
    param[0].buffer_length = sizeof(id);
    param[0].buffer_type = MYSQL_TYPE_LONG;
    param[1].buffer = result;
    param[1].buffer_length = strlen(result);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    if (!setup_prepared_stmt(&preparedStmt, store_procedure, conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Failed to initialize statement\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Cannot set parameters\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to insert\n");
        }
        else {
            printf("Update completed\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void rimuoviManager(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[1];
    char usernameManager[46];
    printf("Enter the username of the manager to remove\n");
    getInput(45, usernameManager, false);
    memset(param, 0, sizeof(param));
    param[0].buffer = usernameManager;
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer_length = strlen(usernameManager);
    if (!setup_prepared_stmt(&preparedStmt, "call rimuoviManager(?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize  rimozione manager
statement\n", false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to setup param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to remove manager\n");
        }
        else {
            printf("\nManager removed\n");
        }
```

```
    }
    mysql_stmt_close(preparedStmt);
}
int containsPoint(char* string) {
    int i;
    int found = 0;
    //Return 0 if the string not contains point, 1 if the string contains one point and -1 if the string is
malformed
    for (i = 0; i < strlen(string); i++) {
        if (string[i] == '.' && !found) {
            found = 1;
        }
        else {
            if (string[i] < 48 || string[i]>57) {
                return -1;
            }
        }
    }
    if (found == 1)
        return -1;
    return found;
}
void inserisciFilmCatalogo(MYSQL* conn) {
    /*
        Il carattere ; non è ammesso perché utilizzato come delimitatore di stringa nelle store procedure
    */
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[8];
    char titolo[61];
    char regista[46];
    char yearC[5];
    char type[46];
    char costoC[10];
    double costo;
    char remakeTitolo[601];
    char remakeRegista[451];
    strcpy(remakeTitolo, "");
    strcpy(remakeRegista, "");
    printf("\nInsert title\n");
    getInput(60, titolo, false);
    printf("\nEnter the director\n");
    getInput(45, regista, false);
    char options[3] = { '1','2','3' };
    char scelta=multiChoice("Insert type 1) New\n 2) Classic\n 3) Other\n", options, 3);
    switch (scelta) {
        case '1':
            strcpy(type, "nuovo");
            break;
        case '2':
            strcpy(type, "classico");
```

```
            break;
        case '3':
            strcpy(type, "other");
            break;
    }
    printf("Enter the year of publication\n");
    leggiNumeri(4, yearC);
    printf("Enter the list of actors reporting the full name for each actor\n");
    int goOn = 1;
    char options1[2] = { '1','2' };
    char temp[46];
    char attori[1000];
    strcpy(attori, "");
    while (goOn) {
        scelta = multiChoice("Type 1 to insert an actor or 2 to end\n", options, 2);
        switch (scelta) {
            case '1':
                do {
                    printf("Enter the actor's full name (the character; is not allowed)\n");
                    getInput(45, temp, false);
                } while (contains(temp, ';'));
                strcat(attori, temp);
                strcat(attori, ";");
                break;
            case '2':
                goOn = 0;
                break;
        }
    }
    printf("Insert the films of which the inserted film is a remake\n");
    goOn = 1;
    while (goOn) {
        scelta = multiChoice("Enter 1 to insert a movie or 2 to end\n", options, 2);
        switch (scelta) {
        case '1':

            do {
                printf("Enter the movie title, the character; it is not allowed\n");
                getInput(60, temp, false);
            } while (contains(temp, ';'));
            strcat(remakeTitolo, temp);
            strcat(remakeTitolo, ";");
            do {
                printf("Enter the director of the film\n");
                getInput(45, temp, false);
                strcat(remakeRegista, temp);
                strcat(remakeRegista, ";");
            }while(contains(temp,';'));
            break;
        case '2':
```

```
            goOn = 0;
            break;
        }
    }
    do {
        printf("Inserisci il costo del film\n");
        getInput(9, costoC, false);
    } while (containsPoint(costoC) == -1);
    if (!containsPoint(costoC)) {
        strcat(costoC, ".");
        strcat(costoC, "0");
    }
    costo = atof(costoC);
    MYSQL_TIME* date = (MYSQL_TIME*)malloc(sizeof(MYSQL_TIME));
    if (date == NULL) {
        printf("Error malloc\n");
        exit(0);
    }
    date->day = 1;
    date->month = 1;
    date->year = atoi(yearC);
    memset(param, 0, sizeof(param));
    param[0].buffer = titolo;
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer_length = strlen(titolo);
    param[1].buffer = regista;
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer_length = strlen(regista);
    param[2].buffer = type;
    param[2].buffer_length = strlen(type);
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[3].buffer = date;
    param[3].buffer_type = MYSQL_TYPE_DATE;
    param[3].buffer_length = sizeof(date);
    param[4].buffer = attori;
    param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[4].buffer_length = strlen(attori);
    param[5].buffer_type = MYSQL_TYPE_DOUBLE;
    param[5].buffer = &costo;
    param[5].buffer_length = sizeof(costo);
    param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[6].buffer = remakeTitolo;
    param[6].buffer_length = strlen(remakeTitolo);
    param[7].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[7].buffer = remakeRegista;
    param[7].buffer_length = strlen(remakeRegista);
    if (!setup_prepared_stmt(&preparedStmt, "call inserireFilmCatalogo(?,?,?,?,?,?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize settore insertion statement\n",
false);
    }
```

```
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to setup param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to insert film\n");
        }
        else {
            printf("\nFilm correctly added\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void rimuoviFilm(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[2];
    char titolo[61];
    char regista[46];
    printf("\nEnter the title of the movie\n");
    getInput(60, titolo, false);
    printf("\nEnter the director\n");
    getInput(45, regista, false);
    memset(param, 0, sizeof(param));
    param[0].buffer = titolo;
    param[0].buffer_length = strlen(titolo);
    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = regista;
    param[1].buffer_length = strlen(regista);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    if (!setup_prepared_stmt(&preparedStmt, "call rimuoviFilm(?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize settore insertion statement\n",
false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to setup param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to remove film\n");
        }
        else {
            printf("\nFilm correctly removed\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void registraSettore(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[2];
    int centerCode;
```

```
    int settoreCode;
    char code[20];
    char settore[20];
    printf("\033[2J\033[H");
    printf("\nInsert center code\n");
    leggiNumeri(19, code);
    printf("\nInsert sector code\n");
    leggiNumeri(19, settore);
    centerCode = atoi(code);
    settoreCode = atoi(settore);
    if (!setup_prepared_stmt(&preparedStmt, "call inserisciSettore(?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize settore insertion statement\n",
false);
    }
    memset(param, 0, sizeof(param));
    param[1].buffer_type = MYSQL_TYPE_LONG;
    param[1].buffer = &centerCode;
    param[1].buffer_length = sizeof(centerCode);
    param[0].buffer_type = MYSQL_TYPE_LONG;
    param[0].buffer = &settoreCode;
    param[0].buffer_length = sizeof(settoreCode);
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to setup param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, " Unable to execute insert\n");
        }
        else {
            printf("\nSector entered correctly\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void registraCentro(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[4];
    int codeI;
    char indirizzo[61];
    char responsabile[46];
    char telefono[101];
    strcpy(telefono, "");
    char email[451];
    strcpy(email, "");
    char scelta;
    char temp[46];
    char opt[2] = { '1','2' };
    int goOn;
    char code[20];
    printf("\033[2J\033[H");
```

```
    printf("\nInsert Address\n");
    getInput(60, indirizzo, false);
    printf("\nInsert Responsible\n");
    getInput(45, responsabile, false);
    goOn = 1;
    printf("\nEnter phone numbers\n");
    while (goOn) {
        scelta = multiChoice("Choose 1) Enter telephone number\n2)End\n", opt, 2);
        switch (scelta) {
        case '1':
            printf("Enter the telephone number\n");
            leggiNumeri(10, temp);
            if (strlen(telefono) + strlen(temp) > MAX_TEL) {
                printf("You have exceeded the maximum number of phone numbers that can be entered,
you can enter the remaining ones later\n");
                goOn = 0;
            }
            else {
                strcat(telefono, temp);
                strcat(telefono, ";");
            }
            break;
        case '2':
            goOn = 0;
            break;
        }
    }
    goOn = 1;
    printf("\nEnter the email\n");
    while (goOn) {
        scelta = multiChoice("Choose 1) Enter email \n2) Finish\n", opt, 2);
        switch (scelta) {
        case '1':
            printf("Enter email\n");
            getInput(45, temp,false);
            if (strlen(email) + strlen(temp) > MAX_EMAIL) {
                printf("You have exceeded the maximum number of emails that can be entered, you can
enter the remaining ones later\n");
                goOn = 0;
            }
            else {
                strcat(email, temp);
                strcat(email, ";");
            }
            break;
        case '2':
            goOn = 0;
            break;
        }
    }
```

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


    param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[0].buffer = indirizzo;
    param[0].buffer_length = strlen(indirizzo);
    param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[1].buffer = responsabile;
    param[1].buffer_length = strlen(responsabile);
    param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[2].buffer = telefono;
    param[2].buffer_length = strlen(telefono);

    param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[3].buffer = email;
    param[3].buffer_length = strlen(email);
    if (!setup_prepared_stmt(&preparedStmt, "call inserisciCentro(?,?,?,?)", conn)) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to initialize center insertion statement\n",
false);
    }
    if (mysql_stmt_bind_param(preparedStmt, param) != 0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to setup param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, "An error has occured while adding center\n");
        }
        else {
            printf("\nCenter correctly added\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void registraUtente(MYSQL* conn) {
    MYSQL_STMT* preparedStmt;
    MYSQL_BIND param[3];
    char username[46];
    char password[46];
    char role[46];
    char options[2] = { '1','2' };
    char risp;
    printf("\033[2J\033[H");
    printf("\nInsert Username: \n");
    getInput(45, username, false);
    printf("\nInsert Password: \n");
    getInput(45, password, true);
    printf("\n Choose a role\n 1)administrator\n2)manager\n");
    risp = multiChoice("Select role ", options, 2);
    printf("%c", risp);
    switch (risp) {
```

```
    case '1':
        strcpy(role, "administrator");
        break;
    case '2':
        strcpy(role, "manager");
        break;
    default:
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
        abort();
    }
    if (!setup_prepared_stmt(&preparedStmt, "call creaUtente(?,?,?)",conn)) {
        finish_with_stmt_error(conn, preparedStmt, "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 = role;
    param[2].buffer_length = strlen(role);

    if (mysql_stmt_bind_param(preparedStmt, &param)!=0) {
        finish_with_stmt_error(conn, preparedStmt, "Unable to setup param\n", true);
    }
    if (areYouSure()) {
        if (mysql_stmt_execute(preparedStmt) != 0) {
            print_stmt_error(preparedStmt, "An error has occured while adding user\n");
        }
        else {
            printf("\nUser correctly added\n");
        }
    }
    mysql_stmt_close(preparedStmt);
}
void run_as_administrator(MYSQL* conn){
    char risp;
    printf("Switching to administrative role...\n");
    int res;
    if(!parse_config("users/administrator.json", &conf)) {
        fprintf(stderr, "Unable to load administrator configuration\n");
        exit(EXIT_FAILURE);
    }

    if(mysql_change_user(conn, conf.db_username, conf.db_password, conf.database)) {
```

```
        fprintf(stderr, "mysql_change_user() failed\n");
        exit(EXIT_FAILURE);
    }
    char options[9] = { '1','2','3','4','5','6','7','8','9'};
    printf("\033[2J\033[H");
    char* questions = "Choose the operation to perform: \n 1) registerUser, \n 2) registerCenter, \n 3) registerSector \n 4) Remove movie from catalog \n 5) Update chain catalog \n6) Remove Manager \n 7) Go to menu manager \n8) Manage center \n 9) exit\n";
    while (true) {

        risp = multiChoice(questions, options, 9);
        switch (risp) {
        case '1':
            registraUtente(conn);
            break;
        case '2':
            registraCentro(conn);
            break;
        case '3':
            registraSettore(conn);
            break;
        case '4':
            rimuoviFilm(conn);
            break;
        case '5':
            inserisciFilmCatalogo(conn);
            break;
        case '6':
            rimuoviManager(conn);
            break;
        case '7':
            run_as_manager(conn,1);
            break;
        case '8':
            handle_center(conn);
            break;
        case '9':
            return;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
        }

    }
}
```

File parse.c

```
#include <stddef.h>
```

```c
#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];
char* strndup(char* str, int chars)
{
   char* buffer;
   int n;

   buffer = (char*)malloc((chars + 1)*sizeof(char));
   if (buffer)
   {
      for (n = 0; ((n < chars) && (str[n] != 0)); n++) buffer[n] = str[n];
      buffer[n] = 0;
   }

   return buffer;
}
/**
 * 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]) {
#ifndef 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_INVAL;
        }
    }
#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_INVAL;
              }
              parser->pos++;
            }
            parser->pos--;
            break;
          /* Unexpected symbol */
          default:
            parser->pos = start;
            return JSMN_ERROR_INVAL;
        }
      }
    }
    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_INVAL;
            }
            token = &tokens[parser->toknext - 1];
            for (;;) {
                if (token->start != -1 && token->end == -1) {
                    if (token->type != type) {
                        return JSMN_ERROR_INVAL;
                    }
                    token->end = parser->pos + 1;
                    parser->toksuper = token->parent;
                    break;
                }
                if (token->parent == -1) {
                    if(token->type != type || parser->toksuper == -1) {
                        return JSMN_ERROR_INVAL;
                    }
                    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_INVAL;
                    }
                    parser->toksuper = -1;
                    token->end = parser->pos + 1;
                    break;
                }
            }
            /* Error if unmatched closing bracket */
            if (i == -1) return JSMN_ERROR_INVAL;
            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;
#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;
                            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
            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_INVAL;
#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;
    int res = fopen_s(&f,filename, "r+");
    if(res !=0) {
```

```
        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 strndup() to fetch string value */
            conf->host = strndup(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 = strndup(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 = strndup(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 = strndup(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;
}
```

File main.c

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

#include "defines.h"

typedef enum{
    ADMINISTRATOR=1,
    EMPLOYER,
    MANAGER,
    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; // IN
```

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

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

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

    if (mysql_stmt_bind_param(login_procedure, param) != 0) {
        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; // OUT
    param[0].buffer = &role;
    param[0].buffer_length = sizeof(role);

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

    // Retrieve output parameter
    if(mysql_stmt_fetch(login_procedure)) {
        print_stmt_error(login_procedure, "Could not buffer results");
        goto err;
    }

    mysql_stmt_close(login_procedure);
    return role;

    err:
    mysql_stmt_close(login_procedure);
    err2:
    return FAILED_LOGIN;
}

int main(){
    role_t role;
    if(!parse_config("users/login.json", &conf)) {
        fprintf(stderr, "Unable to load login configuration\n");
        exit(EXIT_FAILURE);
```

```
    }

    conn = (MYSQL *)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("Username: ");
    if (getInput(45, conf.username, false) == 0) {
        printf("Error\n");
    }
    printf("Password: ");
    if (getInput(45, conf.password, true) == 0)
        printf("Error\n");


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

    int res;

    switch(role) {
        case ADMINISTRATOR:
                run_as_administrator(conn);
                break;
        case MANAGER:
                run_as_manager(conn,0);
                break;
        case EMPLOYER:
            run_as_employer(conn, conf.username);
                break;
        case FAILED_LOGIN:
                fprintf(stderr, "Invalid credentials\n");
                exit(EXIT_FAILURE);
                break;

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

    mysql_close (conn);
```

```
    return 0;
}
```

File defines.h

```
#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;
typedef struct __list_film {
    char title[61];
    char regista[46];
    int settore;
    int copie;
    char posizione[46];
    struct __list_film* next;
}list_film;
typedef struct __noleggi {
    char cliente[46];
    char title[61];
    char regista[46];
    char data[12];
    int settore;
    struct __noleggi* next;
}noleggi;
extern int areYouSure();
extern void free_list_film(list_film* head);
extern void free_list_noleggi(noleggi* head);
extern noleggi* dump_noleggi(MYSQL* conn, MYSQL_STMT* stmt, char* title);
extern list_film* dump_result_set_film(MYSQL* conn, MYSQL_STMT* stmt, char* title);
extern int parse_config(char *path, struct configuration *conf);
extern int getInput( int lung, char *stringa, bool hide);
extern int yesOrNo(char *question);
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_employer(MYSQL *conn,char* username);
extern void run_as_manager(MYSQL *conn,int isAdmin);
extern void run_as_administrator(MYSQL *conn);
extern int inserisciOrario(char* question, char* string, int length);
extern void leggiNumeri(int length, char* string);
extern int dump_result_set_with_list(MYSQL* conn, MYSQL_STMT* stmt, char* title, char** list,
int maxRow);
extern void dump_multiple_rs(MYSQL* conn, MYSQL_STMT* stmt, char* title,HANDLE
handle);
extern int contains(char* string, char character);
```