Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

Sistema di gestione di sale cinematografiche

Matricola 0266759

Roberto Fardella

Indice

1. Descrizione del Minimondo

Si vuole realizzare il sistema informativo di una catena di cinema, che si occupi anche della gestione delle prenotazioni.

L'amministrazione della catena gestisce i cinema. Ciascun cinema ha un numero arbitrario di sale, identificato da un numero di sala. In ogni sala c'è un numero arbitrario di posti, ciascun individuato da una lettera per la fila ed un numero di posto.

In ogni sala vengono proiettati più film quotidianamente. Ciascun cinema può proiettare lo stesso film più volte in una giornata, in sale differenti. Ogni film ha una durata, un nome, è associato al cast degli attori protagonisti e una casa cinematografica. Lo stesso film, proiettato in orari differenti e in sale differenti, può avere un costo per il biglietto differente, proprio in relazione alla sala e all'orario in cui esso viene proiettato.

Il sistema di prenotazione è tale per cui gli utenti possono prenotarsi per una visione, scegliendo un posto disponibile. Dal momento dell'inizio della procedura di prenotazione, un cliente ha a disposizione 10 minuti per perfezionare la prenotazione. Dopo aver scelto il posto, al cliente è data la possibilità di inserire i dati relativi alla propria carta di credito (numero, intestatario, data di scadenza, codice CVV). Una volta inseriti questi dati, il sistema restituisce all'utente un codice di prenotazione.
Fino a 30 minuti dall'inizio della proiezione, il cliente ha la possibilità di annullare la sua prenotazione fornendo al sistema il codice di prenotazione.
La catena di cinema gestisce anche i propri dipendenti, divisi in maschere e proiezionisti. Gli amministratori della catena definiscono i turni di lavoro, di otto ore massimo. I turni sono gestiti su base settimanale. Un report permette di sapere agli amministratori della catena se qualche spettacolo è sprovvisto di proiezionista o se qualche cinema, in qualche fascia oraria di apertura, è sprovvisto di almeno due maschere per la verifica dei biglietti all'ingresso.

La verifica dei biglietti avviene da parte delle maschere mediante l'utilizzo del codice di prenotazione. Un biglietto utilizzato viene associato nel sistema al fatto che quel determinato posto, per quella determinata proiezione, è stato occupato dallo spettatore. Non è possibile utilizzare più volte lo stesso codice di prenotazione per accedere al cinema. A fini statistici, gli amministratori possono generare dei report mensili che mostrano per ciascun cinema e ciascuna sala quante prenotazioni sono state confermate, quante sono state annullate, e quante prenotazioni confermate non sono state utilizzate per accedere al cinema.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
3-4-5			Frase contorta (vedi riquadro sotto)
8-11			Frase contorta
9-10	associato	Ed è associato	Frase contorta
12	relazione	associazione	Sinonimo di associare, unifico I termini
26	Spettatore	Utente	Unificare sinonimo
10	Casa cinematografica	Casa di produzione cinematografica	Rendo più esplicito il termine
14	Visione	proiezione di un film	Rendere più esplicito il termine + unificare sinonimo
Da 15 a 19	Cliente	Utente	Unificare sinonimo
20	Poiezione	Proiezione del film	Rendere più esplicito il termine
22	La catena di cinema	l'amministrazione della catena	Unifico i sinonimi sotto il termine di "l'amministrazione della catena"
23	Turni	Turni di lavoro	Rendo più esplicito il termine
24	Spettacolo	Proiezione di un certo film	Rendo più esplicito il termine
27-28	Un biglietto utilizzato viene associato nel sistema al fatto che quel determinato posto, per quella determinata proiezione, è stato occupato dallo spettatore.	Un biglietto, utilizzato da parte di un utente per accedere ad una proiezione di un film, viene associato nel sistema in base al posto che esso occupa e dalla data e ora della proiezione di quel film.	Frase contorta

Specifica disambiguata

L'amministrazione della catena gestisce i cinema. Ciascun cinema ha un numero arbitrario di sale, dove ogni sala è identificata da un numero.
In ogni sala vengono proiettati più film quotidianamente. In ogni sala c'è un numero arbitrario di posti, dove ogni posto è identificato da una lettera per la fila ed un numero.

Ogni film ha una durata, un nome.
Ciascun cinema può proiettare lo stesso film più volte in una giornata, in sale differenti.
Un Film è associato al cast degli attori protagonisti e ad una casa di produzione cinematografica.
Lo stesso film, proiettato in orari differenti e in sale differenti, può avere un costo per il biglietto differente, a seconda della sala e all'orario in cui esso viene proiettato.
Il sistema di prenotazione è tale per cui gli utenti possono prenotarsi per una proiezione di un film, scegliendo un posto disponibile. Dal momento dell'inizio della procedura di prenotazione, un utente ha a disposizione 10 minuti per perfezionare la prenotazione. Dopo aver scelto il posto, all'utente è data la possibilità di inserire i dati relativi alla propria carta di credito (numero, intestatario, data di scadenza, codice CVV). Una volta inseriti questi dati, il sistema restituisce all'utente un codice di prenotazione.
Fino a 30 minuti dall'inizio della proiezione del film, lo spettatore ha la possibilità di annullare la sua prenotazione fornendo al sistema il codice di prenotazione.

L'amministrazione della catena gestisce anche i propri dipendenti, divisi in maschere e proiezionisti. Gli amministratori della catena definiscono i turni di lavoro, di otto ore massimo. I turni di lavoro sono gestiti su base settimanale. Un report permette di sapere agli amministratori della catena se qualche spettacolo è sprovvisto di proiezionista o se qualche cinema, in qualche fascia oraria di apertura, è sprovvisto di almeno due maschere per la verifica dei biglietti all'ingresso.

La verifica dei biglietti avviene da parte delle maschere mediante l'utilizzo del codice di prenotazione. Un biglietto, utilizzato da parte di uno spettatore per accedere ad una proiezione di un film, viene associato nel sistema in base al posto che esso occupa in una sala di un certo cinema e dalla data e ora della proiezione di quel film.
Non è possibile utilizzare più volte lo stesso codice di prenotazione per accedere al cinema.
A fini statistici, gli amministratori possono generare dei report mensili che mostrano per ciascun cinema e ciascuna sala quante prenotazioni sono state confermate, quante sono state annullate, e quante prenotazioni confermate non sono state utilizzate per accedere al cinema.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Cinema	Cinema costituito da un certo numero di sale con un numero arbitrario di posti.		Cinema, Film, Biglietto, Spettatore
Prenotazione	Spettatori che effettuano la prenotazione di una proiezione di un film e di un posto disponibile per tale visione.		Biglietto, Proiezione Film
Dipendente	Dipendenti che lavorano presso un cinema. Si dividono in maschere e proiezionisti. Gli amministratori definiscono i turni di lavoro dei dipendenti.		Cinema, Proiezione Film, Biglietto
Biglietto	Biglietto acquistato da		Cinema, Proiezione

	un utente tramite prenotazione		Film
Film	Film disponibili per la visione in un cinema		Cinema, Proiezione Film
Proiezione Film	Film proiettato in una sala di un cinema	spettacolo	Sala, Dipendente ,Film, Biglietto

Raggruppamento dei requisiti in insiemi omogenei

Frasi di carattere generale
Si vuole realizzare il sistema informativo di una catena di cinema, che si occupi anche della gestione delle prenotazioni.

Frasi relative ai Cinema
Ciascun cinema può proiettare lo stesso film più volte in una giornata, in sale differenti.
Ciascun cinema ha un numero arbitrario di sale, dove ogni sala è identificata da un numero. In ogni sala vengono proiettati più film quotidianamente. In ogni sala c'è un numero arbitrario di posti, dove ogni posto è identificato da una lettera per la fila ed un numero.

Frasi relative ai Film e alla loro proiezione
Ogni film ha una durata, un nome. Un film è associato al cast degli attori protagonisti e ad una casa di produzione cinematografica. Lo stesso film, proiettato in orari differenti e in sale differenti, può avere un costo per il biglietto differente, a seconda della sala e all'orario in cui esso viene proiettato.

Frasi relative alla prenotazione

Il sistema di prenotazione è tale per cui gli utenti possono prenotarsi per una proiezione di un film, scegliendo un posto disponibile.
Dopo aver scelto il posto, all'utente è data la possibilità di inserire i dati relativi alla propria carta di credito (numero, intestatario, data di scadenza, codice CVV).
Dal momento dell'inizio della procedura di prenotazione, un utente ha a disposizione 10 minuti per perfezionare la prenotazione. Una volta inseriti questi dati, il sistema restituisce all'utente un codice di prenotazione.
Fino a 30 minuti dall'inizio della proiezione del film, l'utente ha la possibilità di annullare la sua prenotazione fornendo al sistema il codice di prenotazione.
Non è possibile utilizzare più volte lo stesso codice di prenotazione per accedere al cinema.

Frasi relative ai dipendenti
L'amministrazione della catena gestisce anche i propri dipendenti, divisi in maschere e proiezionisti. Gli amministratori della catena definiscono i turni di lavoro, di otto ore massimo. I turni di lavoro sono gestiti su base settimanale.

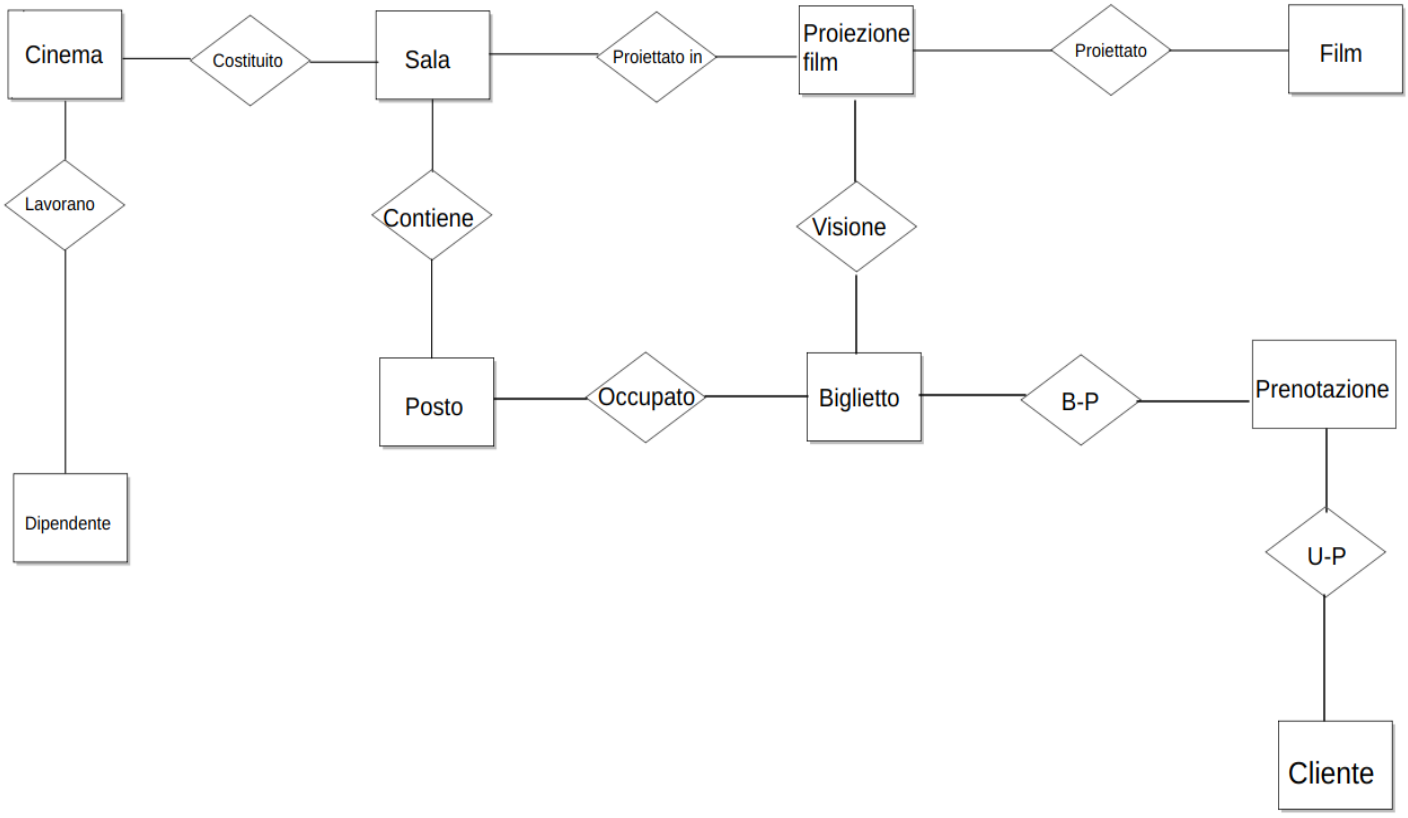
Frasi relative ai biglietti
La verifica dei biglietti avviene da parte delle maschere mediante l'utilizzo del codice di prenotazione. Un biglietto, utilizzato da parte di un utente per accedere ad una proiezione di un film, viene associato nel sistema in base al posto che esso occupa e dalla data e ora della proiezione di quel film.

3. Progettazione concettuale

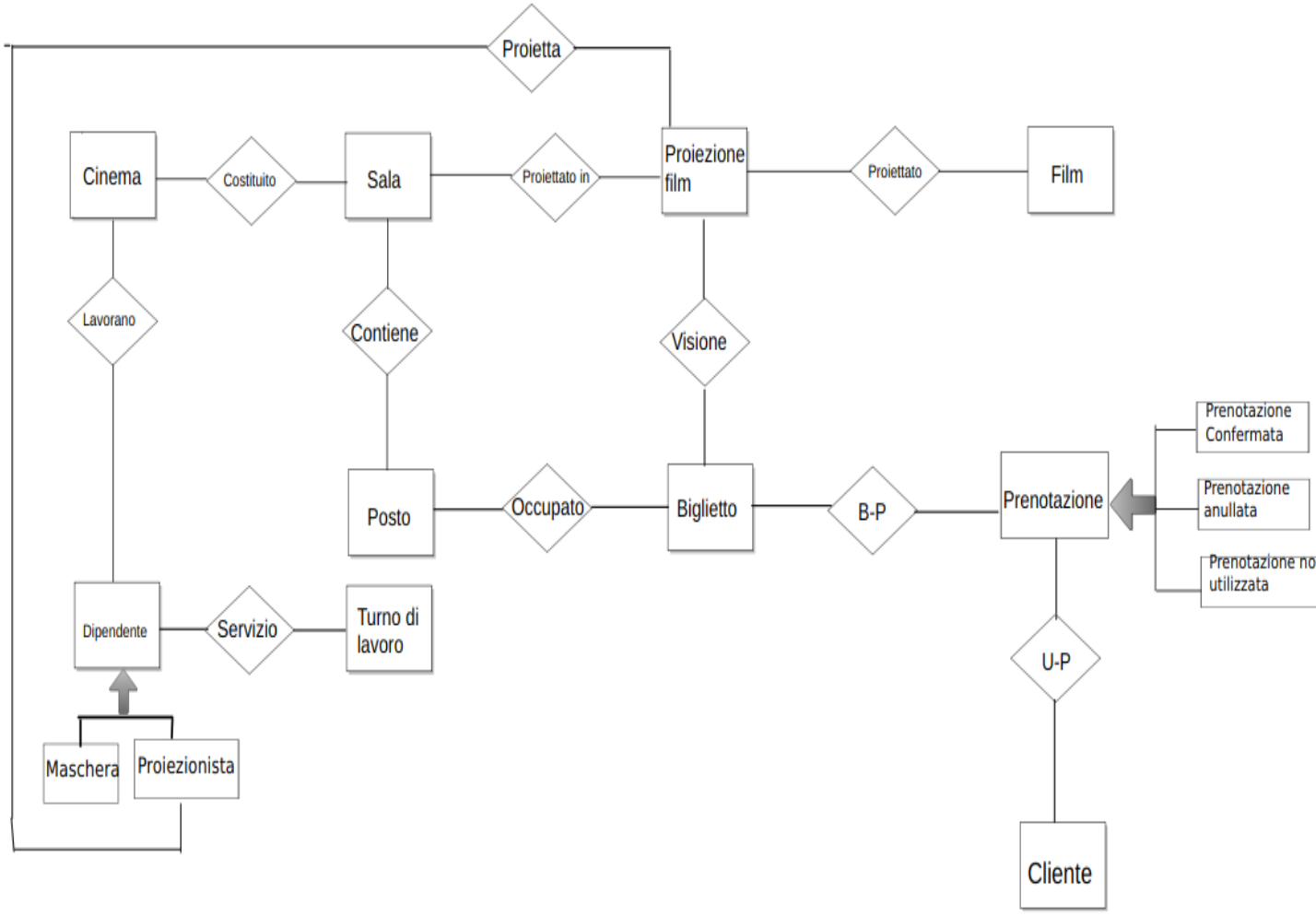
Costruzione dello schema E-R

Dalle analisi dei requisiti e dopo averli raggruppati in modo omogeneo, si è proseguito nella costruzione dello Schema E-R adottando un procedimento misto.

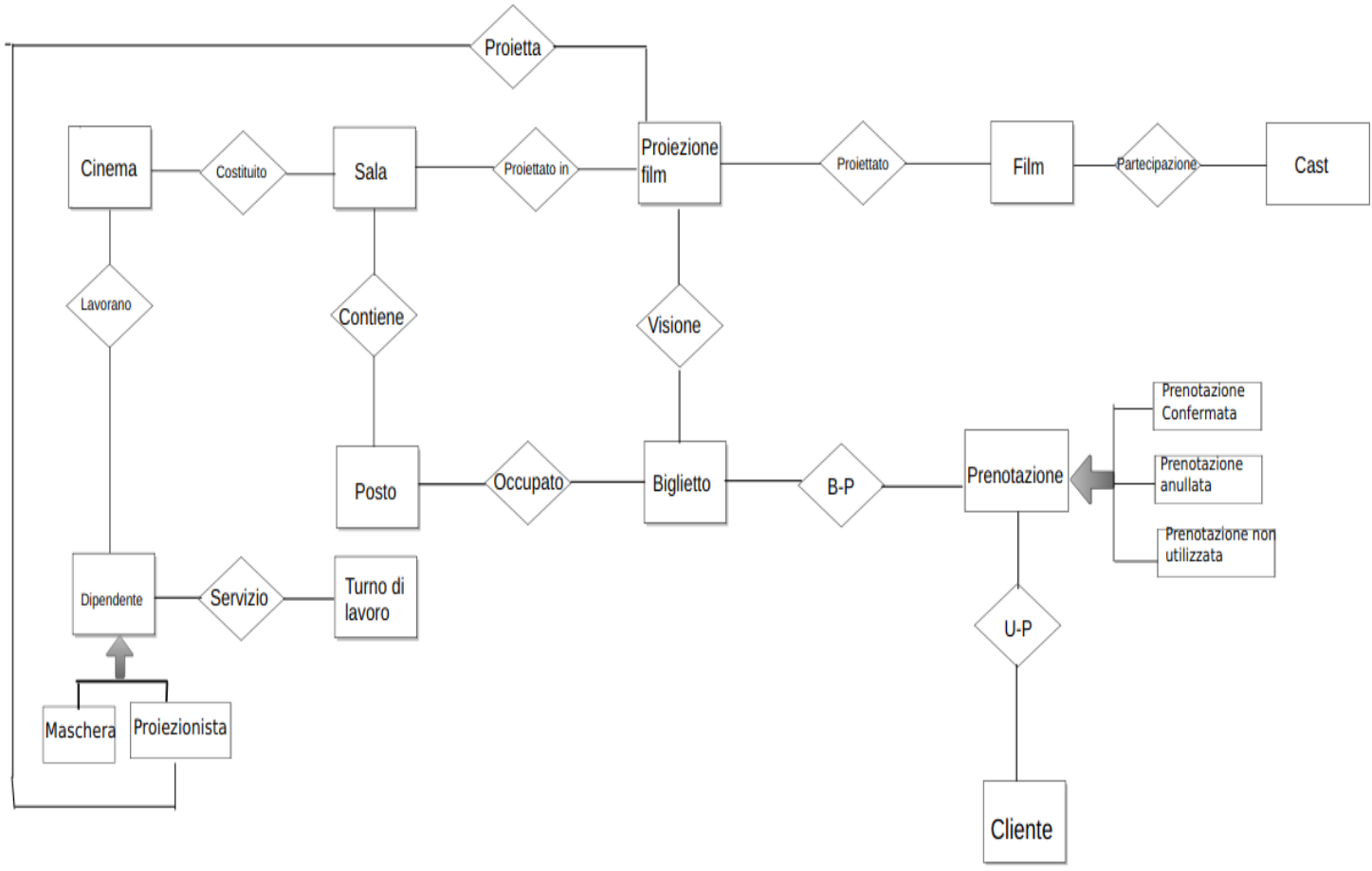
Partendo quindi dai raggruppamenti effettuati sulle specifiche disambiguate viene prodotto conseguentemente lo scheletro dello schema ottenendo 9 entità e 7 relazioni.



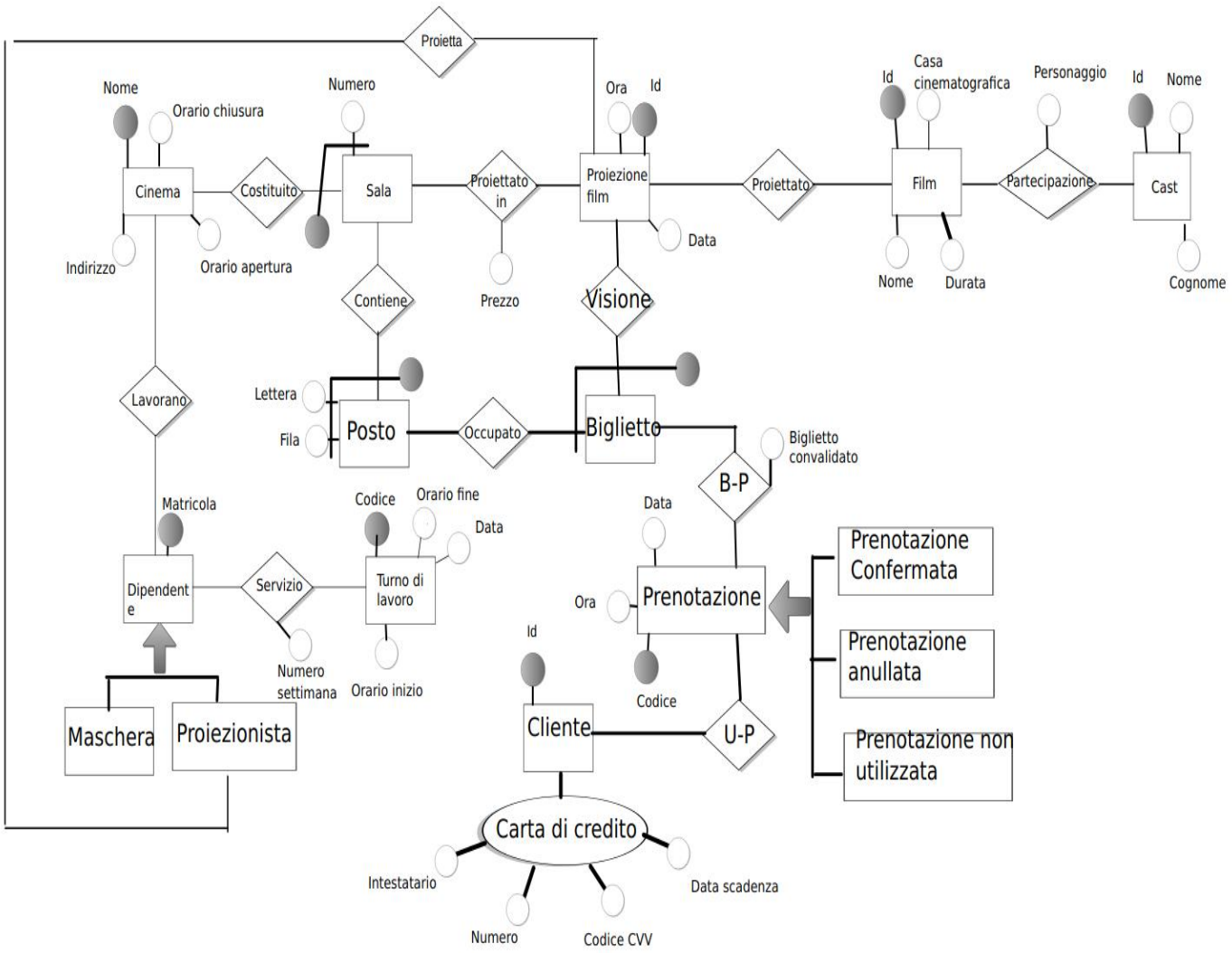
Proseguendo nella raffinazione di tale scheletro dello schema, andiamo ad aggiungere le due generalizzazioni totali sulle entita Prenotazione e Dipendente.
All'entita "Dipendente" in particolare, abbiamo anche estratto dalle specifiche I turni di lavoro dei dipententi mettendoli in associazione (denominata "Servizio").
Per rappresentare al meglio il mondo di riferimento descritto dalle specifiche date, viene messo in associazione Proiezionista con l'entita Proiezione film, denominandola "Proietta".



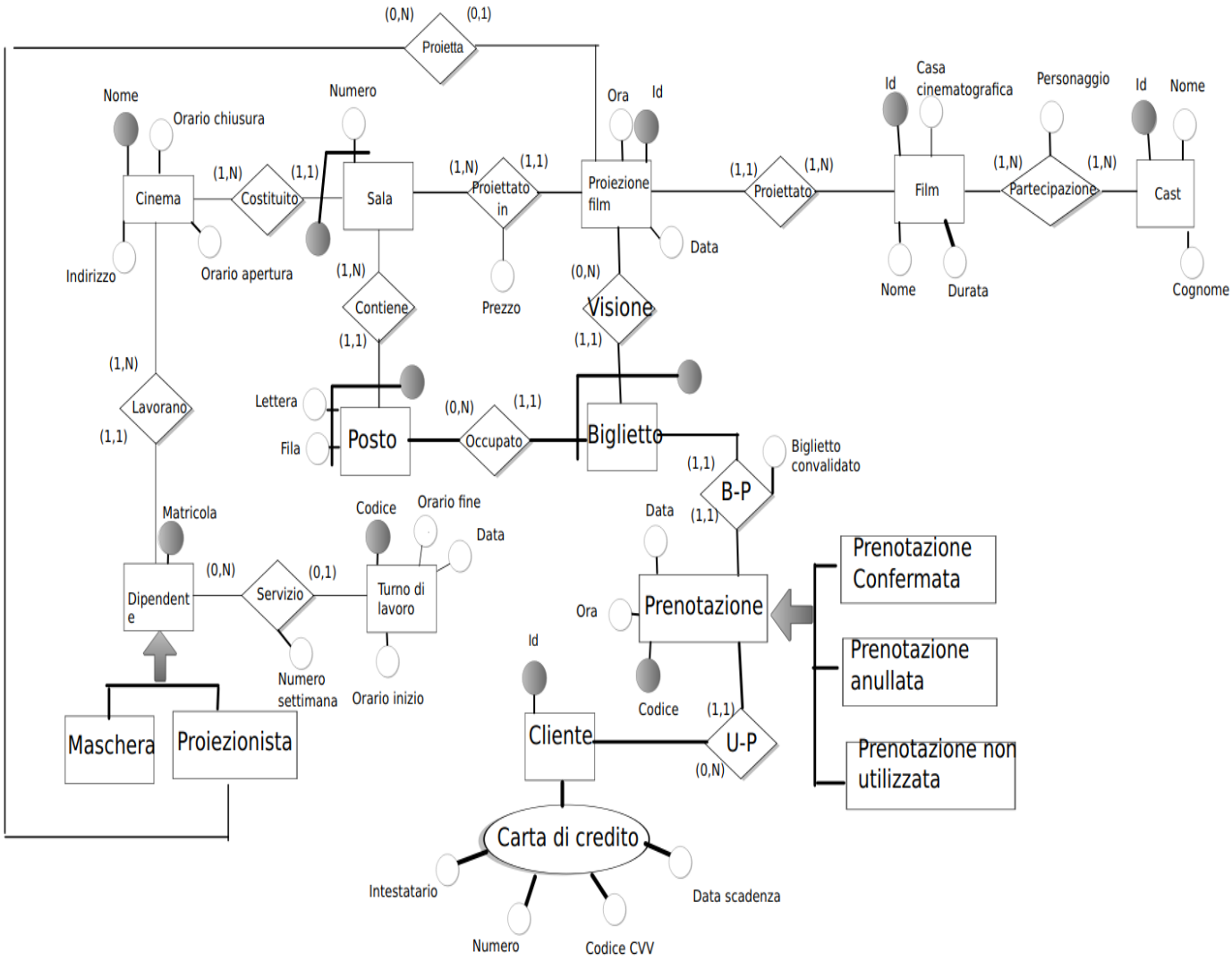
Un ultimo osservazione, prima di proseguire per l'inserimento degli attributi per ogni entità e relazione, è di intendere il cast degli attori protagonisti come un'entità poiché è probabile che nel caso di un film, possono esserci uno o più attori protagonisti e, nel caso del cast degli attori protagonisti, possono partecipare a uno o più film.



Successivamente, abbiamo posto per ogni entita e associazione I relativi attributi che sono stati dedotti dalle specifiche analizzate.
Ad alcune entita, come Proiezione film. Film, Cast e Cliente, è stato introdotto un identificatore univoco, dovuto al fatto che l'insieme degli attributi raccolti per ogni entita, dalle specifiche, non erano sufficienti per identificarne univocamente le occorrenze.



Lo schema complessivo viene da quanto segue, incluse le cardinalita tra le associazioni.



Regole aziendali

1) Non è possibile utilizzare più volte lo stesso codice di prenotazione per accedere al cinema.

2) Gli amministratori della catena definiscono i turni di lavoro, di otto ore massimo.

3) Dal momento dell'inizio della procedura di prenotazione, un utente ha a disposizione 10 minuti per perfezionare la prenotazione.
Una volta inseriti questi dati, il sistema restituisce all'utente un codice di prenotazione. Fino a 30 minuti dall'inizio della proiezione del film, l'utente ha la possibilità di annullare la sua prenotazione

fornendo al sistema il codice di prenotazione.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Cinema	Cinema facente parte di una catena	Nome, citta, OrarioApertura, OrarioChiusura	Nome
Sala	Sala di un cinema, possono essercene un numero arbitrario all'interno di un cinema	Numero	Numero,Cinema
Posto	Posto di una sala, possono essercene un numero arbitrario all'interno di una sala	LetteraFila, Numero	LetteraFila, Numero, NumeroSala,Cinema
Proiezione Film	Un film Proiettato in quella determinata data, ora e sala, può avere un prezzo del biglietto differente	Data, Ora, ID	ID
Film	Film proiettato in una sala di un cinema	Id, Nome,Durata, Casa cinematografica	Id
Cast	Cast degli attori protagonisti	Nome, Cognome, Id	Id
Dipendente	Dipendente che lavora all'interno di un cinema	Matricola,	Matricola
Maschera	Tipologia di dipendente, controlla I biglietti.	Matricola	Matricola
Proiezionista	Tipologia di dipendente, proietta I film	Matricola	Matricola

	nelle sale		
TurnoDiLavoro	Turni di lavoro dei dipendenti, vengono definiti dagli amministratori	Codice, Orario inizio, Orario fine, Data	Codice
Biglietto	Biglietto acquistato da un utente per vedere la proiezione di un film in qualche sala di un certo cinema		Lettera fila, Numero posto, Numero sala, Cinema
Prenotazione	Prenotazione effettuata da un utente dopo aver inserito tutti i dati necessari per la visione per l'ottenimento di un biglietto per la visione del film	Codice, Data, Ora	Codice
Cliente	Cliente che si prenota per una proiezione di un film. Il biglietto viene acquistato dal cliente mediante carta di credito	Id, CartaDiCredito(Numero, Intestatario, CodiceCVV, DataScadenza)	Id
Prenotazione Confermata	Prenotazione andata a buon fine nel sistema	Codice, Data, Ora	Codice
Prenotazione non utilizzata	Prenotazione confermata ma che non è stata usufruita per la visione del film	Codice, Data, Ora	Codice
Prenotazione annullata	Prenotazione che l'utente ha annullato nei tempi prestabiliti	Codice, Data, Ora	Codice

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo[1]	Volume atteso
Cinema	E	7
Sala	E	35
Posto	E	5.250
Dipendente	E	110
Maschera	E	60
Proiezionista	E	40
Biglietto	E	105.000
Prenotazione	E	105.000
Prenotazione confermata	E	100.000
Prenotazione annullata	E	3.500
Prenotazione non utilizzata	E	1.500
Turno di Lavoro	E	660
Utente	E	75.000
Proiezione Film	E	1.050
Film	E	300
AttoreProtagonista	E	100
Cast	R	300
Biglietto-Prenotazione(B-P)	R	105.000
Visione	R	105.000
Proietta	R	1050
Servizio	R	660
Proiettato in	R	1.050
Proiettato	R	1.050
Occupato	R	105.000
Contiene	R	5.250
Costituito	R	75
Lavorano	R	100
Utente-Prenotazione(U-P)	R	105.000

1 Indicare con E le entità, con R le relazioni

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
1	Inserisci i dati della carta di credito e memorizza il codice di prenotazione del biglietto prenotato precedentemente	700 al giorno
2	Inserisci le informazioni di un biglietto, ossia un posto disponibile per la prenotazione e la proiezione di un certo film che si desidera andare a vedere al cinema.	700 al giorno
3	Definire il turno di lavoro di un dipendente	1 a settimana
4	Stampa la seguente informazione: Mostrare se una proiezione di un certo film è sprovvisto di proiezionista o se qualche cinema, in qualche fascia oraria di apertura, è sprovvisto di almeno due maschere per la verifica dei biglietti all'ingresso.	35 al giorno
5	A fini statistici, stampare per le sale di ogni cinema, quante prenotazioni sono state confermate, quante sono state annullate, e quante prenotazioni confermate non sono state utilizzate per accedere al cinema.	1 al mese
6	Inserisci una Proiezione di un film	20 al giorno
7	Dato un codice di prenotazione, stampare le seguenti informazioni di un biglietto: Nome del cinema, sala, lettera e numero del posto, il prezzo del biglietto, data e ora della proiezione e il nome di quel film	652 al giorno
8	Inserisci un nuovo Cliente	350 al giorno
9	Annullare la prenotazione del biglietto	30 al giorno
10	Inserire una prenotazione come "Inutilizzata"	40 al mese

Costo delle operazioni

Per le seguenti operazioni si considerino che le scritture valgano un fattore doppio delle operazioni di lettura.

Per la 1° operazione, abbiamo un accesso in scrittura nell'entità Cliente (per memorizzare i dati della carta di credito), abbiamo un accesso in scrittura nell'associazione, Utente-prenotazione (per inserire il codice di prenotazione di quell'utente), un accesso in scrittura nell'entità Prenotazione, un accesso in scrittura nell' associazione Biglietto-Prenotazione (per memorizzare il codice di prenotazione per quel biglietto), in totale:
(2+2+2+2) *700 = 5.600 accessi giornalieri

Per la 2° operazione abbiamo un accesso in scrittura nell'entità Biglietto, un accesso in scrittura nell'associazione Occupato, un accesso in scrittura nell'associazione Visione, in totale:
(2+2+2)*700 = 4.200 accessi giornalieri

Per la 3° operazione, abbiamo un accesso in lettura nell'entita Dipendente (per cercare l'occorrenza di interesse), un accesso in scrittura nell'associazione Servizio (supponendo in media 6 turni di lavoro per dipendente) ed infine un accesso in scrittura nell'entita Turno di lavoro per memorizzare il turno di lavoro del dipendente scelto.
Quindi:
(1+2*6+2*6) *1*110 = 2750 accessi settimanali (circa 99 accessi giornalieri)

Per la 4° operazione, partendo dall'entità cinema recuperando le informazioni sugli orari di apertura, arriviamo ad accedere fino all'associazione Lavorano per ottenere le informazioni sui turni delle maschere e nell'associazione Proietta per i proiezionisti, quindi abbiamo.
Proseguendo poi per l'altro "ramo" che collega Cinema con Sala e Proiezione film, arriviamo all'entità' proiezione film per recuperare le proiezioni di interesse:

-) 1 accesso in lettura nell'entità Cinema;
-) 14 accessi medi in lettura (è il numero medio di dipendenti per cinema e ottenuti dividendo il numero di dipendenti per il numero) nell'associazione Lavorano e nell'entità dipendente;
-) I dipendenti per ogni cinema sono, proporzionalmente, 3/5 maschere e 2/5 proiezionisti.
Da questa ultima osservazione abbiamo:

- -) S1 = 14 *6*(3/5) accessi in lettura nell'associazione Servizio e nell'entità Turno di lavoro (il 6 come fattore moltiplicativo è il numero medio di turni di lavoro per dipendente che, in questo caso, sono le maschere);
- -) S2 = 14 * 20 (2/5) accessi in lettura nell'associazione Proietta (il 20 come fattore moltiplicativo è il numero medio di film proiettati da un proiezionista settimanalmente)

-) 5 accessi medi in lettura nell'associazione Costituito e nell'entità' Sala (numero medio di sale per cinema);
-) 150 accessi medi in lettura nell'associazione Proiettato in e nell'entita Proiezione film (numero medio di proiezioni al giorno fatte nelle sale di un cinema);

Sommando gli accessi e, moltiplicando tutti questi accessi per la frequenza attesa e il numero di cinema presenti, abbiamo:

(1+(1*14)*2+ 1*S1*2 + 1*S2 + 1*5*2 + 150*2)*7*1 = 3510 accessi settimanali (circa 126 accessi giornalieri)

L' 5° operazione richiede il seguente numero di accessi:
-)7 accessi in lettura nelle'entita Cinema
-) 35 accessi in lettura nell'entità sala e nell'associazione Costituito;
-) 1050 accessi in lettura nell'associazione Proiettato in e nell' entità Proiezione Film;
-) 105.000 accessi in lettura) nell'associazione Visione, nell'entità Biglietto, nell'associazione Biglietto-Prenotazione ed infine nell'entità Prenotazione (per contare le prenotazioni confermate, annullate e quelle non utilizzate);
Moltiplicando per la frequenza attesa:
(1*7+(1+1)*35 + 1.050*2 + 105.000*4) = 422.177 accessi mensili = 15.078 accessi giornalieri

Per la 6° operazione abbiamo un accesso in scrittura nell'entità Proiezione film, un accesso in scrittura nell'associazione Proiettato in (Per aggiungere la coppia Sala – Proiezione Film), un accesso in scrittura nell'associazione Proiettato (per aggiungere la coppia Proiezione film - Film):
(2+2+2) *20= 120 accessi giornalieri

Per la 7° operazione abbiamo un certo numero di accessi in lettura, poiché partendo dall'entità Prenotazione, si deve arrivare alle entità "più lontane" come le entità Cinema e Film, per recuperare le informazioni richieste dalle specifiche, per un totale di 10 accessi in lettura.
Ottenendo 10*652 = 6.520 accessi giornalieri

Per la 8° operazione, abbiamo un solo accesso in scrittura nell'entità Cliente:

2*350 = 700 accessi giornalieri

Per la 9° operazione è necessario un accesso in lettura nell'entità prenotazione (per cercare il biglietto di interesse) ed uno in scrittura (per l'aggiornamento dell'esito della prenotazione), quindi:
(1+2)*30= 90 accessi giornalieri

L'operazione 10 può essere trascurata in quanto viene eseguita in modalità' batch e non viene effettuata frequentemente.

Ristrutturazione dello schema E-R

1) La generalizzazione totale sull'entità Dipendente coinvolge l'operazione 3 e 4.

Tali operazioni non fanno particolari distinzioni tra sottoinsieme di occorrenze dell'entità dipendente e non ci sono attributi che caratterizzano in modo distinto le entità' figlie della generalizzazione.
La soluzione migliore è accorpare le entità figlie della generalizzazione nel genitore (Dipendente) aggiungendo un nuovo attributo "tipo" che discrimina se qualche occorrenza dell'entità Dipendente è un proiezionista o una maschera.

2)accorpando l'entità Prenotazione in Biglietto può essere comunque vantaggioso per le operazioni 5 e 7 riducendo così gli accessi.
Attuando tale ristrutturazione di tale schema parziale, otteniamo:

-)per la op. 5: 210.000 accessi mensili (7500 accessi giornalieri) in meno in quanto non occorre visitare l'associazione B-P e l'entità Prenotazione;

-) per l'op. 7: 2*652 = 1.304 accessi giornalieri in meno;

3) l'operazione 5 ci chiede quante prenotazioni sono state confermate, quante sono state annullate, e quante prenotazioni confermate non sono state utilizzate per accedere al cinema.

Andiamo a vedere se è possibile aggiungere un attributo per ogni tipo di prenotazione (se è confermata, annullata o non utilizzata) in aggiunta all'entità Proiezione film.

Le operazioni coinvolte sono la 1,5, 9 e 10.

Il calcolo delle operazioni senza questi dati ridondanti è stato precedentemente fatto.
Supponendo quindi che, all'interno dell'entità Sala, ci siano questi 3 attributi ridondanti, abbiamo per ogni operazione i seguenti costi:
-) Per l'operazione 1 abbiamo lo stesso numero di accessi che abbiamo già calcolato, in più dobbiamo però accedere in lettura in "Proiezione film", "Proiettato in", e poi accedere in lettura e in scrittura per arrivare nell'entità Sala (per incrementare il numero di prenotazioni confermate), quindi abbiamo
(1+1+1+2) *700 = 3.500 accessi giornalieri in più con il dato ridondante.

-) Per l'operazione 5 rispetto alla non ridondanza dei tre attributi, abbiamo:
(1*7+(1+1) *35) *1 = 77 accessi mensili = 3 accessi giornalieri

Dovuto al fatto che non è più necessario arrivare ad accedere in lettura fino all'entità Prenotazione.

-) per l'operazione 9 oltre quindi a modificare l'occorrenza corrispondente all'annullamento della prenotazione nell'entità Prenotazione occorre arrivare fino all'entità Sala per aggiornare il dato corrispondente, aggiungendo 6 accessi in lettura e 1 in scrittura, quindi:
(1*6+2) *30 = 240 accessi giornalieri;

-) l'operazione 10, anche qui viene trascurata;

Dal seguente calcolo, otteniamo, considerando le operazioni coinvolte in accessi giornalieri:

Con dato ridondante	Senza dato ridondante
Op. 1: 5.600	Op. 1: 5600 + 3.500 = 9.100
Op. 5: 15.078 -7.500 (considerando l'accorpamento Biglietto-Prenotazione) = 7578	Op. 5: 3

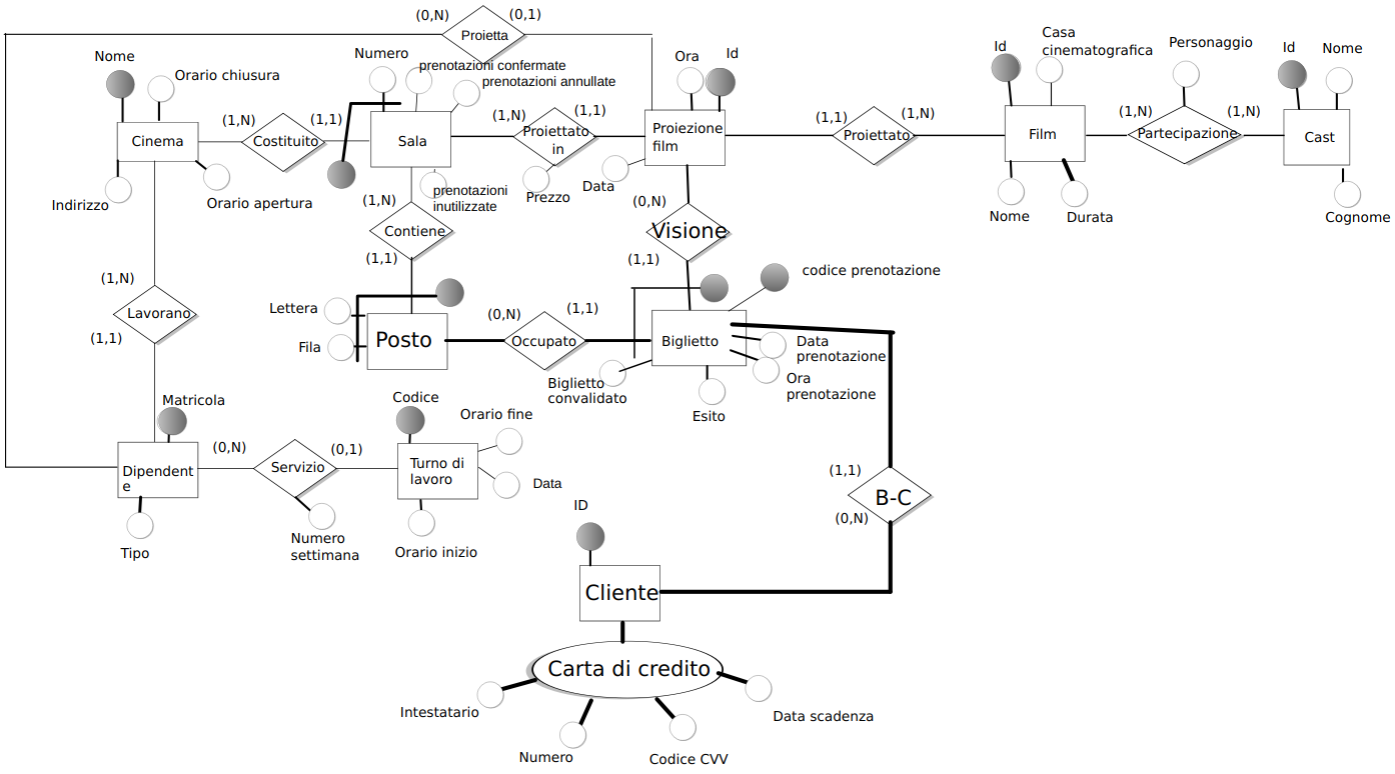
Op. 9: 90	Op. 1: 90+240 = 330
Totale= 13.268	Totale = 9.433

In conclusione, considerando anche un leggero spreco di memoria (3 attributi di tipo intero, occorrendo per ogni occorrenza 4 Byte) di 35*3*4 = 420 Bytes otteniamo ben 13.268 - 9433 = 3.835 accessi giornalieri in meno rispetto alla non ridondanza dei tre attributi nell'entità Sala.

3)La generalizzazione totale sull'entità Prenotazione è stata eliminata accorpando le entità figlie della generalizzazione nel genitore, aggiungendo un nuovo attributo nominato "Esito" per distinguere il tipo di occorrenza (annullata, confermata o inutilizzata).

Per diminuire il costo delle operazioni 5 e 7, viene accorpata l'entità Prenotazione nell'entità Biglietto.

Qui sotto viene riportata la seguente ristrutturazione dello schema E-R:



Trasformazione di attributi e identificatori

L'entità Biglietto ha come chiave primaria ben 4 identificatori esterni e l'attributo Codice_prenotazione.
Vengono quindi rimossi come chiave primaria i 4 identificatori esterni e rendendo come chiave prima solamente Codice_prenotazione così da garantire, in fase di progettazione fisica, le strutture ausiliarie per accedere ai dati siano di dimensioni ridotte, permettendo un risparmio di memoria nella realizzazione di legami logici tra le varie relazioni e facilitando le operazioni di join.

Traduzione di entità e associazioni

Cinema (Nome, Indirizzo, Orario apertura, Orario chiusura)

Sala(Numero,Cinema,Prenotazione_confermate,Prenotazioni_annullate,Prenotazioni_inutilizzate) con vincolo di integrità referenziale tra l'attributo Cinema della relazione Sala e l'attributo Nome della relazione Cinema

Posto (Numero, Lettera, Sala, Cinema) con vincolo di integrità referenziale tra l'attributi Sala e Cinema della relazione Posto e tra i corrispettivi attributi Numero e Cinema della relazione Sala

Film (ID, Nome, Durata, Casa cinematografica)

Cast (ID, Nome, Cognome)

Partecipazione (Attore, Film, Personaggio) con i seguenti vincoli di integrità referenziale:
-) l'attributo Attore della relazione Cast e l'attributo Codice della relazione Attore protagonista;
-) l'attributo Film della relazione Cast e l'attributo Id della relazione Film;

Dipendente (Matricola, Nome, Cognome, Cinema, Tipo) Con vincolo di integrità referenziale tra l'attributo Cinema della relazione Dipendente e l'attributo Nome della relazione Cinema

Turno di lavoro (Codice, Orario inizio, Orario fine, Data)

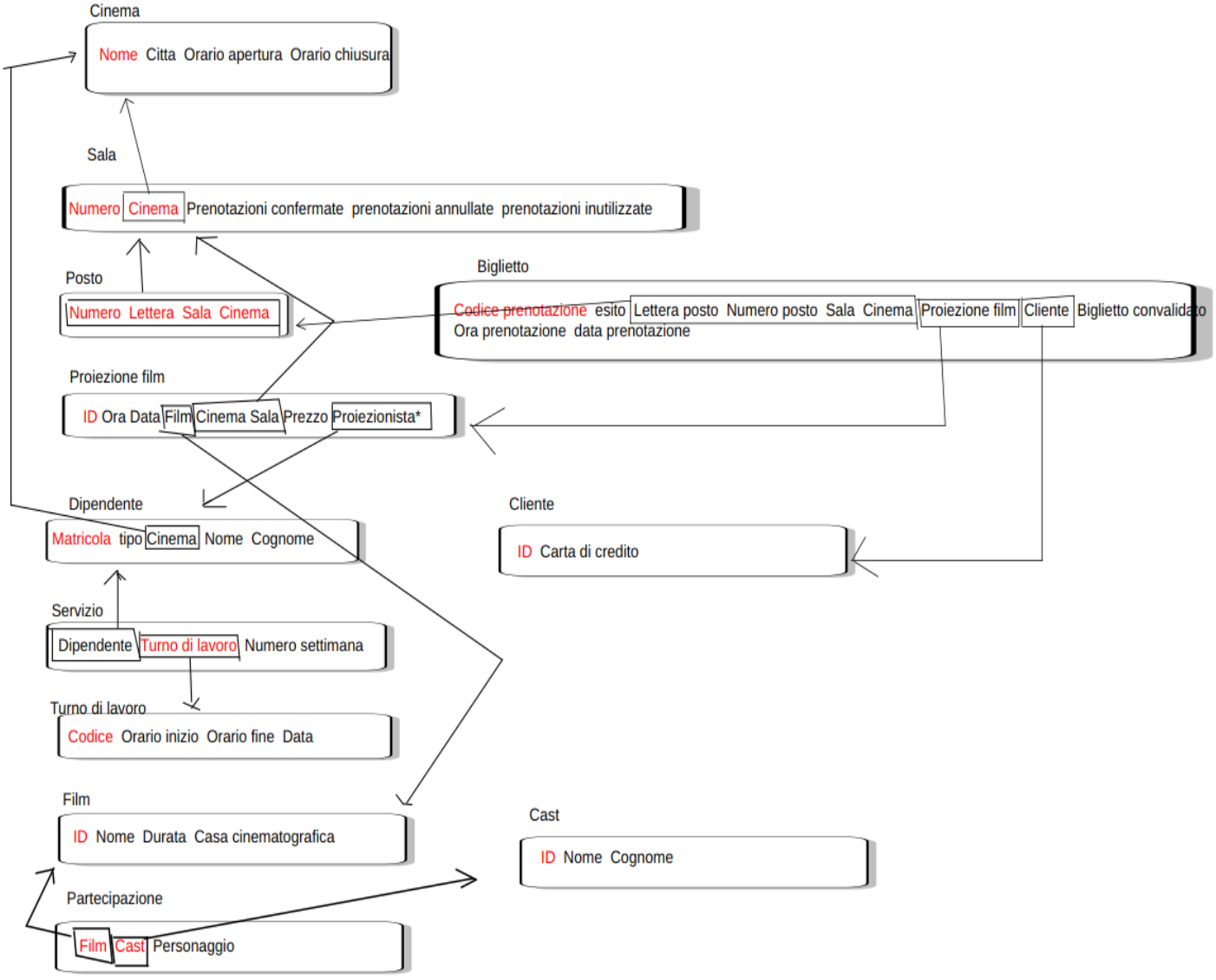
Servizio (Dipendente, Turno di lavoro, Numero settimana) con i seguenti vincoli di integrità referenziale:
-) l'attributo Dipendente della relazione Servizio e l'attributo Matricola della relazione Dipendente;
-) l'attributo Turno di lavoro della relazione Servizio e l'attributo Codice della relazione Turno di lavoro;

Proiezione film (ID, Data, Ora, Film, Cinema, Sala, Prezzo, Proiezionista*) con i seguenti vincoli di integrità referenziale:
-) l'attributo Cinema della relazione Proiezione film e l'attributo Cinema della relazione Sala;
-) l'attributo Sala della relazione Proiezione film e l'attributo Numero della relazione Sala;
-) l'attributo Matricola della relazione Proiezione film e l'attributo Proiezionista della relazione Dipendente;

Biglietto (Esito, Codice prenotazione, Lettera posto, Numero posto, Sala, Cinema, Proiezione film, Cliente, Biglietto convalidato, Ora prenotazione, Data prenotazione) con i seguenti vincoli di integrità referenziale:
-) l'attributo Lettera posto della relazione Biglietto e l'attributo Lettera fila della relazione Posto;

-) l'attributo Numero posto della relazione Biglietto e l'attributo Numero della relazione Posto;
-) l'attributo Sala della relazione Biglietto e l'attributo Numero sala della relazione Posto;
-) l'attributo Cinema della relazione Biglietto e l'attributo Cinema della relazione Posto;
-) l'attributo Proiezione film della relazione Biglietto e l'attributo Codice della relazione Proiezione film;
-) l'attributo Cliente della relazione Biglietto e l'attributo Id della relazione Cliente;

Cliente (Id, Carta di credito);



Normalizzazione del modello relazionale

Cliente

Id	Carta di credito

0266759	0222 02211 2121 02156; Michele fuzzi; 233; 08/28:

L'attributo composto carta di credito non è definito su un dominio con valori atomici, violando la prima forma normale.

È necessario ristrutturare la relazione come segue:
Cliente

Id	Numero	Intestatario	Codice CVV	Data scadenza
0266759	0222 02211 2121 02156	Michele Fuzzi	233	08/28

Notiamo però che, per ogni a id corrisponde una carta di credito, ma ogni numero di carta corrisponde un unico intestatario, codice cvv e data scadenza.

Id--> Numero
Numero -> Intestatario, Codice CVV, Data scadenza;
Quest'ultima dipendenza funzionale viola la terza forma normale, in quanto Numero nella relazione cliente non è chiave.

È necessario ristrutturare la relazione utente come segue:
Cliente

0266729	0222 02211 2121 02156

Carta di credito

Numero	Intestatario	Codice CVV	Data scadenza	Cliente
0222 02211 2121 02156	Michele Fuzzi	233	08/28	

Successivamente, notiamo dalle specifiche delle dipendenzi funzionali in Proiezione film, che sono le seguenti:

Codice_prenotazione -> Sala, Ora
Sala, Ora --> Prezzo

In particolare, l'ultima dipendenza viola anch'essa la 3NF, pertanto ristrutturiamo la relazione come segue:
Proiezione film

ID	Data	Ora	Film	Cinema	Sala	Proiezionista

1	15-02-2020	14:00:00	1	Augustus	1	NULL

Prezzo_biglietto

Sala_proiezione	Ora_proiezione	Prezzo
1	14:00:00	5.00

5. Progettazione fisica

Utenti e privilegi

Gli utenti che sono stati generati per poter iteragire con la base di dati sono I seguenti:
-) Amministratore;
-)Maschera;
-)Cliente;

Tali Utenti detengono privilegi di EXECUTE sulle stored procedures ed eseguendole, a seconda dei casi, vengono svolte operazioni di inserimento, lettura e/o aggiornamento sulle tabelle a cui si riferiscono.

Gli amministratori della catena detengono privilegi per le seguenti stored procedure:

-) aggiungi_turno_lavoro: vengono eseguite delle operazioni di lettura nella tabella Dipendente, operazioni di a aggiornamento nella tabella Proiezione_film ed infine, operazioni di inserimento nelle tabelle Turno_di_lavoro e Servizio;

-) report_prenotazioni: due operazioni di lettura nelle tabelle Utente e Sala;

-) aggiungi_proiezione: due operazioni di inserimento nelle tabelle Proiezione_film e Prezzo_biglietto;

-) crea_utente: un solo inserimento nella tabella utente;

-) aggiungi_cliente: un solo inserimento nella tabella Cliente;

-) report_dipendenti: Operazioni di lettura nelle tabelle Cinema, Dipendente, Proiezione_film, Film e Utente;

-) lista_dipendenti: operazioni di lettura nelle tabelle Utente, Dipendente e Servizio;

Le maschere che lavorano all'interno dei cinema, detenngono privilegi per le seguenti stored procedure:

-) Convalida_biglietto: operazioni di lettura nelle tabelle Utente e Biglietto e di aggiornamento nella tabella Biglietto;

-) stampa_biglietto: operazioni di lettura nelle tabelle Biglietto, Posto, Film e Proiezione_film;

I clienti dei cinema, detengono infine I medesimi privilegi per le seguenti stored procedure:

-) aggiungi_biglietto: operazioni di lettura nelle tabelle Cliente, Proiezione_film, Prezzo_biglietto, una operazione di inserimento nella tabella Biglietto e una operazione di aggiornamento nella tabella Sala;

-) aggiungi_carta_di_credito: operazione di lettura nella tabella Cliente e di inserimento nella tabella Carta_di_credito;

-) annulla_prenotazione: operazioni di lettura nella tabella Cliente, Proiezione_film e Biglietto, operazioni di

aggiornamento nelle tabelle Biglietto e Sala;

-)tramite le stored procedure lista_proiezioni e lista_posti disponibili, vengono eseguite operazioni di lettura nelle tabelle Utente, Proiezione_film, Film, Posto e Biglietto;

Strutture di memorizzazione

Tabella Cinema		
Attributo	**Tipo di dato**	**Attributi[2]**
Nome	Varchar(30)	PK, NN
Indirizzo	Varchar(45)	NN
Orario_apertura	Time	NN
Orario_chiusura	Time	NN

Tabella Sala		
Attributo	**Tipo di dato**	**Attributi**
Numero	Int	PK, NN, UN
Cinema	Varchar(30)	PK, NN
Prenotazioni_confermate	Int	NN
Prenotazioni_annullate	Int	NN
Prenotazioni_inutilizzate	Int	NN

Tabella Posto		
Attributo	**Tipo di dato**	**Attributi**
Lettera_fila	Char	PK, NN
Numero	Int	PK, NN, UN

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.

Numero_sala	Int	PK, NN, UN
Cinema	Varchar(30)	PK, NN

Tabella Film		
Attributo	**Tipo di dato**	**Attributi**
Id	Int	PK ,NN, AI
Nome	Varchar(45)	NN
Casa_cinematografica	Varchar(45)	NN
Durata	Time	NN

Tabella Cast		
Attributo	**Tipo di dato**	**Attributi**
ID	Int	PK, NN, AI
Nome	Varchar(35)	NN
Cognome	varchar(35)	NN

Tabella Partecipazione		
Attributo	**Tipo di dato**	**Attributi**
Attore	Int	PK, NN
Film	Int	PK, NN
Personaggio	Varchar(45)	NN

Tabella Dipendente		
Attributo	**Tipo di dato**	**Attributi**
Matricola	Int	PK, NN, AI
Nome	Varchar(35)	NN

Cognome	Varchar(35)	NN
Tipo	ENUM(“Maschera”, “Proiezionista”)	NN
Cinema	Varchar(30)	NN
Maschera	Varchar(45)	NN

Tabella Proiezione_film		
Attributo	**Tipo di dato**	**Attributi**
Data	Date	NN
Ora	Time	NN
Film	Int	NN
Cinema	Varchar(30)	NN
Numero_sala	Int	NN, UN
ID	Int	PK, NN, AI
Proiezionista	int	

Tabella Biglietto		
Attributo	**Tipo di dato**	**Attributi**
Esito	ENUM('Confermata', 'Annullata','Inutilizzata', 'In_prenotazione')	NN
Cinema	Varchar(30)	NN
Sala	Int	NN, UN
Lettera_posto	Char	NN
Numero_posto	Int	NN, UN
Codice_prenotazione	Int	PK,NN, AI
Proiezione_film	Int	NN

Cliente	Int	NN
Biglietto_convalidato	ENUM('si','no')	NN
Ora_prenotazione	Time	NN
Data_prenotazione	Date	NN

Tabella Prezzo_biglietto		
Attributo	**Tipo di dato**	**Attributi**
Sala_proiezione	Int	PK, NN, UN
Ora_proiezione	TIME	PK, NN
Prezzo	FLOAT	NN

Tabella Turno_di_lavoro		
Attributo	**Tipo di dato**	**Attributi**
Codice	Int	PK, NN, AI
Amministratore	VARCHAR(45)	NN
Orario_inizio	Time	NN
Orario_fine	Time	NN
Data	Date	NN

Tabella Servizio		
Attributo	**Tipo di dato**	**Attributi**

Dipendente	Int	NN
Turno_di_lavoro	Int	PK, NN
Numero settimana	Smallint	NN
Amministratore	Varchar(45)	NN

Tabella Cliente		
Attributo	**Tipo di dato**	**Attributi**
Id	Int	PK, NN, AI
Username	Varchar(45)	NN

Tabella Utente		
Attributo	**Tipo di dato**	**Attributi**
Username	Varchar(45)	PK, NN
Password	CHAR(32)	NN
Ruolo	ENUM('Amministratore','Cliente','Maschera')	NN

Tabella Carta_di_credito		
Attributo	**Tipo di dato**	**Attributi**
Numero	Int	PK, NN
Intestatario	Varchar (45)	NN
Codice_CVV	Int	UN, NN
Data_Scadenza	date	NN
Cliente	INT	NN

Indici

Gli indici creati hanno la peculiarità di ottenere delle ricerche efficienti e per un numero elevato di tuple all'interno di una tabella essi risultano indispensabili.
La creazione degli indici è stata tenuta conto anche dal fatto del numero strettamente crescente di informazioni nel tempo e dal carico delle operazioni che vengono fatte nel database.

Per ogni tabella e per ogni sua chiave primaria viene creato un indice primario.
Per i campi che vanno a formare la chiave esterna vengono generati degli indici, supportando il vincolo di integrità referenziale e, inoltre, sono utili soprattutto per un controllo veloce sulle chiavi esterne nelle tabelle secondarie senza accedere a tutti i campi di una tabella.

Nella tabella Film viene creato un indice FULL TEXT per il campo Nome, per velocizzare la ricerca nell'operazione 8.

Nella tabella Biglietto viene creato un indice UNIQUE per il campo Codice_prenotazione dovuto all'utilizzo di tale campo per alcune stored procedure presenti nel database.

Tabella Cliente	
Indice PRIMARY	**Tipo[3]:**
Colonna 1	PR
Indice fk_Cliente_1_idx	**Tipo:**
Colonna 5	IDX

Tabella Utente	
Indice PRIMARY	**Tipo:**
Colonna 1	PR

Tabella Dipendente	
Indice	**Tipo:**
Colonna 1	PR
Indice fk_Cinema_idx	**Tipo:**

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

Colonna 5	IDX
Indice fk_Maschera_utente_idx	**Tipo**:
Colonna 6	IDX

Tabella Turno di lavoro	
Indice PRIMARY	**Tipo:**
Colonna 1	PR

Tabella Biglietto	
Indice PRIMARY	**Tipo:**
Colonna 6	PR
Indice fk_posto_idx	**Tipo:**
Colonna 2 Colonna 3 Colonna 4 Colonna 5	IDX
Indice Proiezione film_idx	**Tipo:**
Colonna 7	UQ
Indice Cliente_idx	**Tipo**:
Colonna 8	IDX

Tabella Film	
Indice PRIMARY	**Tipo:**
Colonna 1	PR
Indice nome	**Tipo:**
Colonna 2	FULL TEXT

Tabella Carta_di_credito

Indice PRIMARY	**Tipo:**
Colonna 1	PR
fk_Carta di credito_1_idx	**Tipo:**
Colonna 5	IDX

Tabella Cinema	
Indice PRIMARY	**Tipo:**
Colonna 1	PR

Tabella Sala	
Indice PRIMARY	**Tipo:**
Colonna 1 Colonna 2	PR
Indice fk_Cinema_idx	**Tipo:**
Colonna 2	IDX

Tabella Posto	
Indice PRIMARY	**Tipo:**
Colonna 1 Colonna 2 Colonna 3 Colonna 4	PR
Indice fk_Posto_idx	**Tipo:**
Colonna 3 Colonna 4	IDX

Tabella Proiezione_film	
Indice PRIMARY	**Tipo:**
Colonna 6	PR
Indice DATA_ORA_idx	**Tipo:**

Colonna 1 Colonna 2	IDX
Indice fk_Film_idx	**Tipo:**
Colonna 3	IDX
Indice fk_SalaCinema_idx	**Tipo:**
Colonna 4 Colonna 5	IDX
Indice fk_Proiezionista_idx	**Tipo:**
Colonna 7	

Tabella Cast	
Indice PRIMARY	**Tipo:**
Colonna 1	PR

Tabella Partecipazione	
Indice PRIMARY	**Tipo:**
Colonna 1 Colonna 2	PR
Indice fk_Cast_idx	**Tipo:**
Colonna 2	IDX

Tabella Prezzo_biglietto	
Indice PRIMARY	**Tipo:**
Colonna 1 Colonna 2	PR

Tabella Servizio	
Indice PRIMARY	**Tipo:**
Colonna 2	PR
Indice fk_Utente_idx	**Tipo:**
Colonna 4	IDX
Indice fk_dipendente_idx	**Tipo:**

Colonna 1	IDX

Trigger

Per realizzare l'asserzione n.1 che è contenuta nella regola aziendale dello schema ER, viene istanziato il seguente trigger chiamato “UnSoloCodicePrenotazione”.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Biglietto_convalidato` BEFORE UPDATE ON `Biglietto` FOR EACH ROW
BEGIN
if (old.Biglietto_convalidato ="si" and new.Biglietto_convalidato = OLD.Biglietto_convalidato) then
signal sqlstate '45000' set message_text = 'Biglietto già convalidato';
	end if;
END
```

Gli amministratori della catena definendo I turni di lavoro, dovendo rispettare il vincolo di 8 ore massimo per turno, viene creato un trigger che permetta in caso di violazione di tale vincolo, di sollevare un’eccezione.

```
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`MassimoOreLavoro` BEFORE INSERT ON `Turno_di_lavoro` FOR EACH ROW
BEGIN

IF timediff(new.Orario_fine,new.Orario_inizio)  > "08:00:00"  then
Signal sqlstate '45000' set message_text ="eccessivo numero di ore nel turno di lavoro!";
END IF;
END
```

Eventi

Tale evento è stato istanziato in fase di configurazione del sistema.

```
set global event_scheduler = on;

CREATE EVENT IF NOT EXISTS `evento_prenotazione`
on schedule every 10 minute
on completion preserve
	comment 'elimina il biglietto nel caso in cui non si perfezioni entro 10 minuti la fase di prenotazione '
Do
	Begin
		delete from Biglietto
		where  Ora_prenotazione < (curtime() - interval 10 minute) and Esito = ' In_prenotazione';
	End
```

Stored Procedures e transazioni

Per le stored procedure "lista_posti_disponibili" e "report_dipendenti", vengono utilizzati i livelli di isolamento massimo serializable per evitare tutte le anomali possibili tenendo conto della frequenza attesa e possibili complicanze si avrebbero nel caso in cui simultaneamente due o più prenotazioni dello stesso posto da parte di due Clienti diversi e di inserimenti fantasma durante l'esecuzione del report dei dipendenti

Per le stored procedure "annulla_prenotazione", "report_prenotazioni", "aggiungi_turno_di_lavoro" è stato utilizzato il livello di isolamento repeatable read in quanto è stato ritenuto utile evitare tutte le possibili anomalie che potrebbero essere causate dall'esecuzione contemporanea di altre transazioni garantendo l'isolamento.

Per le stored procedure "aggiungi_biglietto", "lista_dipendenti", "aggiungi_carta_di_credito", "Convalida_biglietto", "lista_proiezioni", "stampa_biglietto" viene utilizzato il livello di isolamento read committed, poiché vengono evitata l'anomalia della lettura sporca.

Per la stored procedure "aggiungi_proiezione" viene invece utilizzato il livello d'isolamento minimo read uncommitted, avendo come contenuto della transazione solo due inserimenti.

```
CREATE PROCEDURE `Aggiungi_proiezione` (IN var_data_proiezione varchar(20), IN var_ora_proiezione varchar(20) ,IN var_film INT, IN var_cinema VARCHAR(30), IN var_sala INT, IN var_proiezionista INT)
BEGIN
declare var_data_date DATE;
    declare var_ora_time TIME;
    declare ID INT;
    declare var_prezzo FLOAT;

declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

    set transaction isolation level read uncommitted;
     start transaction;


    set var_data_date = str_to_date(var_data_proiezione, '%Y-%m-%d');
set var_ora_time = str_to_date(var_ora_proiezione, '%T');
    set ID = last_insert_id();

    if (curtime() > "20:00:00" and var_sala = 1) then
set var_prezzo = 7.50;
else
set var_prezzo = 5.00;
end if;
```

```
   INSERT INTO Proiezione_film  values (var_data_date, var_ora_time,var_film, var_cinema, var_sala,ID, var_proiezionista);
   INSERT INTO Prezzo_biglietto (Prezzo, Sala_proiezione, Ora_proiezione) values (var_prezzo, var_sala, var_ora_time);
commit;
END

CREATE PROCEDURE `crea_utente` (IN username VARCHAR(45), IN pass VARCHAR(45), IN ruolo varchar(45))
BEGIN
insert into Utente VALUES(username, MD5(pass), ruolo);
END


CREATE PROCEDURE `login` (in var_username varchar(45), in var_pass varchar(45), out var_role INT)
BEGIN
declare var_user_role ENUM('Amministratore','Cliente', 'Maschera');

   select `ruolo` from `Utente`
where `username` = var_username
      and `password` = var_pass
      into var_user_role;
-- See the corresponding enum in the client
if var_user_role = 'Amministratore' then
set var_role = 1;
elseif var_user_role = 'Cliente' then
set var_role = 2;
elseif var_user_role = 'Maschera' then
set var_role = 3;
else
set var_role = 4;
end if;
END

CREATE PROCEDURE `aggiungi_biglietto` (IN var_lettera_posto CHAR,IN var_numero_posto INT,IN var_username varchar(45), IN var_proiezione INT,OUT var_prezzo FLOAT, OUT var_codice_prenotazione INT)
BEGIN

   declare var_cliente int;
   declare var_cinema varchar(30);
   declare var_sala int;
   declare var_ora_proiezione time;
   declare var_prezzo float;


declare exit handler for sqlexception
   begin
```

```
    rollback;  -- rollback any changes made in the transaction
    resignal;  -- raise again the sql exception to the caller
  end;

  set transaction isolation level read committed;
  start transaction;

  select Id
  into var_cliente
  from Cliente
  where Username = var_username;

  select P.Ora, Cinema, Numero_sala
  from Proiezione_film  P
  where P.ID = var_proiezione into var_ora_proiezione, var_cinema, var_sala ;

  select Prezzo
  into var_prezzo
  from Prezzo_biglietto
  where Ora_proiezione = var_ora_proiezione and Sala_proiezione = var_sala;

  set var_codice_prenotazione = last_insert_id();

insert into `Biglietto`(Cinema, Sala, Lettera_posto, Numero_posto,Cliente,Proiezione_film,Ora_prenotazione,
Data_prenotazione) values
(var_cinema, var_sala, var_lettera_posto, var_numero_posto,var_cliente, var_proiezione, curtime(),curdate());

  update Sala
  set Prenotazioni_confermate = Prenotazioni_confermate +1
  where Numero = var_sala;

  commit;
END


CREATE PROCEDURE `aggiungi_turno_lavoro` (IN var_amministratore VARCHAR(45),IN var_data
varchar(20), IN var_ora_inizio varchar(20), IN var_ora_fine varchar(20), IN var_dipendente INT, OUT
flag_dip BOOLEAN)
BEGIN
declare var_ruolo  ENUM('Amministratore', 'Cliente','Maschera');
      declare var_orario_inizio_time TIME;
      declare var_codice_turno INT;
      declare var_orario_fine_time TIME;
      declare var_data_date DATE;
      declare var_settimana_smallint SMALLINT;
      declare tipo_dipendente ENUM("Maschera","Proiezionista");
      declare flag_dip boolean;
```

```
    declare exit handler for sqlexception
  begin
    rollback;  -- rollback any changes made in the transaction
    resignal;  -- raise again the sql exception to the caller
  end;

    set transaction isolation level serializable;
    start transaction;



    select Tipo
    from Dipendente
    where Matricola = var_dipendente into tipo_dipendente;



    set var_orario_inizio_time = str_to_date(var_ora_inizio, "%T");
    set var_orario_fine_time = str_to_date(var_ora_fine, "%T");
    set var_data_date = str_to_date(var_data, "%Y-%m-%d");
    set var_settimana_smallint = WEEK(var_data_date);

     if tipo_dipendente = 'Proiezionista' then
update Proiezione_film
       set Proiezionista = var_dipendente
       where Proiezione_film.Data = var_data_date and Proiezione_film.Ora between var_orario_inizio_time
and var_orario_fine_time;
end if;

set var_codice_turno = last_insert_id();

    INSERT INTO Turno_di_lavoro values (var_codice_turno, var_amministratore,
var_orario_inizio_time,var_orario_fine_time,var_data_date );
INSERT INTO Servizio values (var_dipendente, var_codice_turno, var_settimana_smallint ,
var_amministratore);
    commit;
END

CREATE PROCEDURE `lista_dipendenti` (IN var_amministratore varchar(45))
BEGIN
declare var_ruolo ENUM('Amministratore', 'Cliente','Maschera');
declare exit handler for sqlexception
  begin
    rollback;  -- rollback any changes made in the transaction
    resignal;  -- raise again the sql exception to the caller
  end;
```

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


select Ruolo
        into var_ruolo
        from Utente
        where Username = var_amministratore;


        if var_ruolo <> 'Amministratore' then
signal sqlstate '45002' set message_text =" l'utente non è un amministratore ";
        end if;


select *
    from Dipendente;

    select * from Servizio;
    commit;
END


CREATE PROCEDURE `report_dipendenti` (in var_amministratore VARCHAR(45))
BEGIN
declare var_ruolo  ENUM('Amministratore', 'Cliente','Maschera');




declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

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

    select Ruolo
into var_ruolo
from Utente
where Username = var_amministratore;


        if var_ruolo <> 'Amministratore' then
signal sqlstate '45002' set message_text =" l'utente non è un amministratore ";
        end if;
```

```
select C.nome
from Cinema C join Dipendente D on C.nome = D.Cinema, Proiezione_film P
where  D.matricola not in (select S.Dipendente
  from Servizio S join Turno_di_lavoro T on S.Turno_di_lavoro = T.Codice
                    where tipo ='Maschera' and T.Orario_inizio >= C.Orario_apertura and T.Orario_fine <=
C.Orario_chiusura )

group by C.nome
having count(D.Matricola)>=2;


select P.ID as Proiezione, F.nome as Film
from  Proiezione_film P join Film F on Film = F.Id
where  P.Proiezionista is null;

   commit;
END


CREATE PROCEDURE aggiungi_carta_di_credito (IN var_username VARCHAR(45),IN var_numero
BIGINT, IN var_intestatario VARCHAR(45), IN var_codice_cvv INT, IN var_data_scadenza
VARCHAR(20), IN var_codice_prenotazione INT)
BEGIN
declare var_codice_cliente int;
   declare var_data_date DATE;

declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
   end;

     set transaction isolation level read committed;
  start transaction;

select Id
   into var_codice_cliente
   from Cliente
   where Username = var_username;

   set var_data_date = str_to_date(var_data_scadenza, '%Y-%m-%d');

INSERT INTO Carta_di_credito values (var_numero, var_intestatario, var_codice_cvv,
var_data_date,var_codice_cliente);

   update Biglietto
```

```
    set Esito = "Confermata"
    where Codice_prenotazione = var_codice_prenotazione;

    commit;
END

CREATE PROCEDURE `lista_posti_disponibili` (IN var_proiezione INT)
BEGIN
declare var_sala int;

      set transaction read only;
    set transaction isolation level serializable;

    select Numero_sala
    from Proiezione_film
    where ID = var_proiezione into var_sala;

select Lettera_fila, Numero
    from Posto
    where (Lettera_fila, Numero) not in (select Lettera_posto, Numero_posto from Biglietto where
Proiezione_film = var_proiezione )
    and Numero_sala = var_sala;
    commit;
END

CREATE PROCEDURE `annulla_prenotazione` (IN var_prenotazione INT, IN var_username varchar(45))
BEGIN


declare var_cliente int;
    declare prenotazione int;
    declare var_ora_proiezione time;
    declare var_sala int;

    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

    set transaction isolation level repeatable read;
    start transaction;

    select Id
    from Cliente
    where Username = var_username into var_cliente;

    if( var_cliente not in (select Cliente from Biglietto)) then
```

```
signal sqlstate '45000' set message_text = 'Impossibile cancellare una prenotazione non effettuata';
end if;

    select P.Ora
    from Proiezione_film P join Biglietto B on P.ID = B.Proiezione_film
    where B.Codice_prenotazione = var_prenotazione into var_ora_proiezione;

    select Codice_prenotazione, Sala
    into prenotazione, var_sala
    from Biglietto
    where COdice_prenotazione = var_prenotazione;

    if var_prenotazione <> prenotazione    then
signal sqlstate '45000' set message_text = 'prenotazione non esistente';
end if;

    if timediff(curtime(), var_ora_proiezione) > '00:30:00' then
signal sqlstate '45000' set message_text = 'impossibile annullare la prenotazione : tempo scaduto';
end if;

    update Biglietto
    set Esito = 'Annullata'
    where Codice_prenotazione = var_prenotazione and Cliente = var_cliente;

    update Sala
    set Prenotazioni_confermate = Prenotazioni_confermate - 1 ,Prenotazioni_annullate =
Prenotazioni_annullate +1
    where Numero = var_sala;
    commit;
END


CREATE PROCEDURE `report_prenotazioni` (IN var_amministratore varchar(45))
BEGIN


declare var_ruolo ENUM('Amministratore','Maschera','Cliente');


declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

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

```
select Ruolo
    from Utente
    where Username = var_amministratore into var_ruolo;

    if var_ruolo <>'Amministratore' then
    signal sqlstate '45003' set message_text='The user is not an amministrator';
end if;

SELECT Cinema, Numero as Numero_sala, Prenotazioni_confermate, Prenotazioni_annullate,
Prenotazioni_inutilizzate
FROM Sala
order by Cinema;


commit;
END


CREATE PROCEDURE `aggiungi_cliente` (IN var_username VARCHAR(45), OUT var_cliente INT)
BEGIN
insert into Cliente (Username) values (var_username);
    set var_cliente = last_insert_id();
END


CREATE PROCEDURE `Convalida_biglietto` (IN var_maschera VARCHAR(45), IN
var_codice_prenotazione INT)
BEGIN
declare var_codice INT;
    declare var_ruolo ENUM('Amministratore','Cliente','Maschera');
    declare var_biglietto_convalidato ENUM('si','no');

    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

     set transaction isolation level repeatable read;
     start transaction;

    select Ruolo
    into var_Ruolo
    from Utente
    where username = var_maschera;

    if var_ruolo <> 'Maschera'      then
signal sqlstate '45003' set message_text = " l'utente non è una maschera";
```

```
end if;

    if var_codice_prenotazione not in (select Codice_prenotazione
from Biglietto) then
signal sqlstate '45000' set message_text = 'Codice di prenotazione non presente nel database';
    end if;

select Biglietto_convalidato
    from Biglietto
    where Codice_prenotazione = var_codice_prenotazione into var_biglietto_convalidato;



    if var_biglietto_convalidato = 'si' then
     signal sqlstate '45000' set message_text = 'la prenotazione è stata già utilizzata';
end if;

    Update Biglietto
    set Biglietto_convalidato ='si'
    where codice_prenotazione= var_codice_prenotazione ;
    commit;
END

CREATE PROCEDURE `lista_proiezioni` (IN var_username varchar(45)  )
BEGIN
declare var_ruolo  ENUM("Amministratore","Maschera","Cliente");
declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

    set transaction read only;
    set transaction isolation level read committed;

    select Ruolo
    from Utente
    where Username = var_username into var_ruolo;

    if var_ruolo <> 'Cliente' then
signal sqlstate '45000' set message_text = "L'utente non è un cliente";
end if;

select F.nome as film,  F.Durata, P.ID as codice_proiezione, P.Data, P.Ora, P.Cinema, P.Numero_sala
    from Proiezione_film P join Film F on P. Film = F.ID;
    -- where P.data between curdate() and curdate() + interval 5 day;
```

```
END

CREATE PROCEDURE `stampa_biglietto` (IN var_codice_prenotazione INT)
BEGIN

   declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
   end;

   set transaction read only;
   set transaction isolation level read committed;




SELECT distinct B.Cinema, B.Sala, B.lettera_posto, B.numero_posto, P.`Data`, P.`Ora`, F.Nome
   FROM Biglietto B, Posto T, Proiezione_film P, Film F
WHERE B.Lettera_posto = T.Lettera_fila and B.Numero_posto = T.Numero
and B.Proiezione_film = P.ID
      and P.film = F.ID
      and B.Codice_prenotazione = var_codice_prenotazione ;
      commit;
END
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
-- MySQL Workbench Forward Engineering

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

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

-- -----------------------------------------------------
-- Schema mydb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `mydb` ;
USE `mydb` ;

-- -----------------------------------------------------
-- Table `mydb`.`Cinema`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Cinema` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Cinema` (
  `Nome` VARCHAR(30) NOT NULL,
  `Indirizzo` VARCHAR(45) NOT NULL,
  `Orario_apertura` TIME NOT NULL,
  `Orario_chiusura` TIME NOT NULL,
  PRIMARY KEY (`Nome`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`Sala`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Sala` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Sala` (
  `Numero` INT UNSIGNED NOT NULL,
  `Cinema` VARCHAR(30) NOT NULL,
  `Prenotazioni_confermate` INT NOT NULL DEFAULT 0,
```

```
  `Prenotazioni_annullate` INT NOT NULL DEFAULT 0,
  `Prenotazioni_inutilizzate` INT NOT NULL DEFAULT 0,
  PRIMARY KEY (`Numero`, `Cinema`),
  CONSTRAINT `fk_Cinema`
    FOREIGN KEY (`Cinema`)
    REFERENCES `mydb`.`Cinema` (`Nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Cinema_idx` ON `mydb`.`Sala` (`Cinema` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`Film`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Film` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Film` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Durata` TIME NOT NULL,
  `Casa_cinematografica` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`))
ENGINE = InnoDB;

CREATE FULLTEXT INDEX `nome` ON `mydb`.`Film` (`Nome`) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`Cast`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Cast` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Cast` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(35) NOT NULL,
  `Cognome` VARCHAR(35) NOT NULL,
  PRIMARY KEY (`ID`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `mydb`.`Partecipazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Partecipazione` ;

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

```
  `Film` INT NOT NULL,
  `Attore` INT NOT NULL,
  `Personaggio` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Film`, `Attore`),
  CONSTRAINT `fk_Cast`
    FOREIGN KEY (`Attore`)
    REFERENCES `mydb`.`Cast` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Cast_2`
    FOREIGN KEY (`Film`)
    REFERENCES `mydb`.`Film` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Cast_idx` ON `mydb`.`Partecipazione` (`Attore` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`Utente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Utente` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Utente` (
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `ruolo` ENUM('Amministratore', 'Cliente', 'Maschera') NOT NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `mydb`.`Dipendente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Dipendente` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Dipendente` (
  `Matricola` INT NOT NULL AUTO_INCREMENT,
  `Cinema` VARCHAR(30) NOT NULL COMMENT '                                        ',
  `Tipo` ENUM('Proiezionista', 'Maschera') NOT NULL,
  `Nome` VARCHAR(35) NOT NULL,
  `Cognome` VARCHAR(35) NOT NULL,
  `Maschera` VARCHAR(45) NULL DEFAULT NULL,
  PRIMARY KEY (`Matricola`),
  CONSTRAINT `fk_Cinema_dipendente`
    FOREIGN KEY (`Cinema`)
    REFERENCES `mydb`.`Cinema` (`Nome`)
```

```
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Maschera_utente`
    FOREIGN KEY (`Maschera`)
    REFERENCES `mydb`.`Utente` (`username`)
    ON DELETE SET NULL
    ON UPDATE SET NULL)
ENGINE = InnoDB
PACK_KEYS = DEFAULT;

CREATE INDEX `fk_Cinema_idx` ON `mydb`.`Dipendente` (`Cinema` ASC) VISIBLE;

CREATE INDEX `fk_Maschera_utente_idx` ON `mydb`.`Dipendente` (`Maschera` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`Turno_di_lavoro`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Turno_di_lavoro` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Turno_di_lavoro` (
  `Codice` INT NOT NULL AUTO_INCREMENT COMMENT '            ',
  `Amministratore` VARCHAR(45) NOT NULL,
  `Orario_inizio` TIME NOT NULL,
  `Orario_fine` TIME NOT NULL,
  `Data` DATE NOT NULL,
  PRIMARY KEY (`Codice`),
  CONSTRAINT `fk_Turno_di_lavoro_1`
    FOREIGN KEY (`Amministratore`)
    REFERENCES `mydb`.`Utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Turno_di_lavoro_1_idx` ON `mydb`.`Turno_di_lavoro` (`Amministratore` ASC)
VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`Servizio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Servizio` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Servizio` (
  `Dipendente` INT NOT NULL,
  `Turno_di_lavoro` INT NOT NULL AUTO_INCREMENT,
  `Numero_settimana` SMALLINT NOT NULL,
  `Amministratore` VARCHAR(45) NOT NULL,
```

```
  PRIMARY KEY (`Turno_di_lavoro`),
  CONSTRAINT `fk_Dipendente`
    FOREIGN KEY (`Dipendente`)
    REFERENCES `mydb`.`Dipendente` (`Matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Turno_di_lavoro`
    FOREIGN KEY (`Turno_di_lavoro`)
    REFERENCES `mydb`.`Turno_di_lavoro` (`Codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Utente`
    FOREIGN KEY (`Amministratore`)
    REFERENCES `mydb`.`Utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Utente_idx` ON `mydb`.`Servizio` (`Amministratore` ASC) VISIBLE;

CREATE INDEX `fk_dipendente_idx` ON `mydb`.`Servizio` (`Dipendente` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`Cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Cliente` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Cliente` (
  `Id` INT NOT NULL AUTO_INCREMENT,
  `Username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id`),
  CONSTRAINT `fk_Cliente_1`
    FOREIGN KEY (`Username`)
    REFERENCES `mydb`.`Utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Cliente_1_idx` ON `mydb`.`Cliente` (`Username` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`Carta_di_credito`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Carta_di_credito` ;

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

```
  `Numero` BIGINT NOT NULL COMMENT '              ',
  `Intestatario` VARCHAR(45) NOT NULL,
  `Codice_CVV` INT UNSIGNED NOT NULL,
  `Data_scadenza` DATE NOT NULL,
  `Cliente` INT NOT NULL,
  PRIMARY KEY (`Numero`),
  CONSTRAINT `fk_Carta di credito_1`
    FOREIGN KEY (`Cliente`)
    REFERENCES `mydb`.`Cliente` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Carta di credito_1_idx` ON `mydb`.`Carta_di_credito` (`Cliente` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`Proiezione_film`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Proiezione_film` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Proiezione_film` (
  `Data` DATE NOT NULL,
  `Ora` TIME NOT NULL,
  `Film` INT NOT NULL,
  `Cinema` VARCHAR(30) NOT NULL,
  `Numero_sala` INT UNSIGNED NOT NULL,
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Proiezionista` INT NULL,
  PRIMARY KEY (`ID`),
  CONSTRAINT `fk_Film`
    FOREIGN KEY (`Film`)
    REFERENCES `mydb`.`Film` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Cinema_proiezione`
    FOREIGN KEY (`Cinema` , `Numero_sala`)
    REFERENCES `mydb`.`Sala` (`Cinema` , `Numero`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Proiezionista`
    FOREIGN KEY (`Proiezionista`)
    REFERENCES `mydb`.`Dipendente` (`Matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Film_idx` ON `mydb`.`Proiezione_film` (`Film` ASC) VISIBLE;
```

```
CREATE INDEX `fk_SalaCinema_idx` ON `mydb`.`Proiezione_film` (`Cinema` ASC, `Numero_sala` ASC) VISIBLE;

CREATE INDEX `DATA_ORA_idx` ON `mydb`.`Proiezione_film` (`Data` ASC, `Ora` ASC) VISIBLE;

CREATE INDEX `fk_Proiezionista_idx` ON `mydb`.`Proiezione_film` (`Proiezionista` ASC) VISIBLE;

CREATE UNIQUE INDEX `Sala_ora_idx` ON `mydb`.`Proiezione_film` (`Numero_sala` ASC, `Ora` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`Posto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Posto` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Posto` (
  `Cinema` VARCHAR(30) NOT NULL,
  `Numero_sala` INT UNSIGNED NOT NULL,
  `Lettera_fila` CHAR NOT NULL,
  `Numero` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`Cinema`, `Lettera_fila`, `Numero_sala`, `Numero`),
  CONSTRAINT `fk_Posto`
    FOREIGN KEY (`Cinema` , `Numero_sala`)
    REFERENCES `mydb`.`Sala` (`Cinema` , `Numero`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Posto_idx` ON `mydb`.`Posto` (`Cinema` ASC, `Numero_sala` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `mydb`.`Biglietto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Biglietto` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Biglietto` (
  `Cinema` VARCHAR(30) NOT NULL,
  `Sala` INT UNSIGNED NOT NULL,
  `Lettera_posto` CHAR NOT NULL,
  `Numero_posto` INT UNSIGNED NOT NULL,
  `Cliente` INT NOT NULL,
  `Codice_prenotazione` INT NOT NULL AUTO_INCREMENT,
  `Proiezione_film` INT NOT NULL,
  `Esito` ENUM('Confermata', 'Annullata', 'Inutilizzata') NOT NULL DEFAULT 'Inutilizzata',
  `Biglietto_convalidato` ENUM('si', 'no') NOT NULL DEFAULT 'no',
```

```
  `Ora_prenotazione` TIME NOT NULL,
  `Data_prenotazione` DATE NOT NULL,
  PRIMARY KEY (`Codice_prenotazione`),
  CONSTRAINT `Cliente`
    FOREIGN KEY (`Cliente`)
    REFERENCES `mydb`.`Cliente` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Proiezione film`
    FOREIGN KEY (`Proiezione_film`)
    REFERENCES `mydb`.`Proiezione_film` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Posto`
    FOREIGN KEY (`Cinema` , `Lettera_posto` , `Sala` , `Numero_posto`)
    REFERENCES `mydb`.`Posto` (`Cinema` , `Lettera_fila` , `Numero_sala` , `Numero`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
AUTO_INCREMENT = 1;

CREATE INDEX `Cliente_idx` ON `mydb`.`Biglietto` (`Cliente` ASC) VISIBLE;

CREATE INDEX `Proiezione film_idx` ON `mydb`.`Biglietto` (`Proiezione_film` ASC) VISIBLE;

CREATE INDEX `fk_posto_idx` ON `mydb`.`Biglietto` (`Cinema` ASC, `Lettera_posto` ASC, `Sala` ASC,
`Numero_posto` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `mydb`.`Prezzo_biglietto`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `mydb`.`Prezzo_biglietto` ;

CREATE TABLE IF NOT EXISTS `mydb`.`Prezzo_biglietto` (
  `Prezzo` FLOAT NOT NULL,
  `Sala_proiezione` INT UNSIGNED NOT NULL,
  `Ora_proiezione` TIME NOT NULL,
  PRIMARY KEY (`Sala_proiezione`, `Ora_proiezione`),
  CONSTRAINT `fk_Prezzo_biglietto_1`
    FOREIGN KEY (`Sala_proiezione` , `Ora_proiezione`)
    REFERENCES `mydb`.`Proiezione_film` (`Numero_sala` , `Ora`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

USE `mydb` ;
```

```
-- -----------------------------------------------------
-- procedure aggiungi_biglietto
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`aggiungi_biglietto`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `aggiungi_biglietto` (IN var_lettera_posto CHAR,IN var_numero_posto INT,IN var_username varchar(45), IN var_proiezione INT,OUT var_prezzo FLOAT, OUT var_codice_prenotazione INT)
BEGIN

    declare var_cliente int;
    declare var_cinema varchar(30);
    declare var_sala int;
    declare var_ora_proiezione time;
    declare var_prezzo float;

          declare exit handler for sqlexception
    begin
       rollback;  -- rollback any changes made in the transaction
       resignal;  -- raise again the sql exception to the caller
    end;

    set transaction isolation level serializable;
    start transaction;

    select Id
    into var_cliente
    from Cliente
    where Username = var_username;

    select P.Ora, Cinema, Numero_sala
    from Proiezione_film  P
    where P.ID = var_proiezione into var_ora_proiezione, var_cinema, var_sala ;

    select Prezzo
    into var_prezzo
    from Prezzo_biglietto
    where Ora_proiezione = var_ora_proiezione and Sala_proiezione = var_sala;

    set var_codice_prenotazione = last_insert_id();

          insert into `Biglietto`(Cinema, Sala, Lettera_posto, Numero_posto,Cliente,Proiezione_film,Ora_prenotazione, Data_prenotazione) values
                (var_cinema, var_sala, var_lettera_posto, var_numero_posto,var_cliente, var_proiezione,
```

```
curtime(),curdate());

    update Sala
    set Prenotazioni_confermate = Prenotazioni_confermate +1
    where Numero = var_sala;

    commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure aggiungi_turno_lavoro
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`aggiungi_turno_lavoro`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `aggiungi_turno_lavoro` (IN var_amministratore VARCHAR(45),IN var_data
varchar(20), IN var_ora_inizio varchar(20), IN var_ora_fine varchar(20), IN var_dipendente INT)
            BEGIN
                        declare var_ruolo  ENUM('Amministratore', 'Cliente','Maschera');
        declare var_orario_inizio_time TIME;
        declare var_codice_turno INT;
        declare var_orario_fine_time TIME;
        declare var_data_date DATE;
        declare var_settimana_smallint SMALLINT;
        declare tipo_dipendente ENUM("Maschera","Proiezionista");


        declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

        set transaction isolation level repeatable read;
                        start transaction;



        select Tipo
        from Dipendente
        where Matricola = var_dipendente into tipo_dipendente;
```

```
	set var_orario_inizio_time = str_to_date(var_ora_inizio, "%T");
	set var_orario_fine_time = str_to_date(var_ora_fine, "%T");
	set var_data_date = str_to_date(var_data, "%Y-%m-%d");
	set var_settimana_smallint = WEEK(var_data_date);

	 if tipo_dipendente = 'Proiezionista' then
				update Proiezione_film
	   set Proiezionista = var_dipendente
	   where Proiezione_film.Data = var_data_date and Proiezione_film.Ora between var_orario_inizio_time
and var_orario_fine_time;
			end if;

			set var_codice_turno = last_insert_id();

	INSERT INTO Turno_di_lavoro values (var_codice_turno, var_amministratore,
var_orario_inizio_time,var_orario_fine_time,var_data_date );
			INSERT INTO Servizio values (var_dipendente, var_codice_turno, var_settimana_smallint ,
var_amministratore);
	commit;
	    END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure aggiungi_carta_di_credito
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`aggiungi_carta_di_credito`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE aggiungi_carta_di_credito (IN var_username VARCHAR(45),IN var_numero
BIGINT, IN var_intestatario VARCHAR(45), IN var_codice_cvv INT, IN var_data_scadenza
VARCHAR(20), IN var_codice_prenotazione INT)
BEGIN
	declare var_codice_cliente int;
   declare var_data_date DATE;

	declare exit handler for sqlexception
   begin
     rollback;  -- rollback any changes made in the transaction
     resignal;  -- raise again the sql exception to the caller
   end;

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

```
        select Id
    into var_codice_cliente
    from Cliente
    where Username = var_username;

    set var_data_date = str_to_date(var_data_scadenza, '%Y-%m-%d');

        INSERT INTO Carta_di_credito values (var_numero, var_intestatario, var_codice_cvv,
var_data_date,var_codice_cliente);

    update Biglietto
    set Esito = "Confermata"
    where Codice_prenotazione = var_codice_prenotazione;

    commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure report_prenotazioni
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`report_prenotazioni`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `report_prenotazioni` (IN var_amministratore varchar(45))
BEGIN

declare var_ruolo ENUM('Amministratore','Maschera','Cliente');

        declare exit handler for sqlexception
    begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
    end;

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

        select Ruolo
    from Utente
    where Username = var_amministratore into var_ruolo;

    if var_ruolo <>'Amministratore' then
```

```
   signal sqlstate '45003' set message_text='The user is not an amministrator';
          end if;

SELECT Cinema, Numero as Numero_sala, Prenotazioni_confermate, Prenotazioni_annullate,
Prenotazioni_inutilizzate
FROM Sala
order by Cinema;

commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure report_dipendenti
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`report_dipendenti`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `report_dipendenti` (in var_amministratore VARCHAR(45))
BEGIN
          declare var_ruolo  ENUM('Amministratore', 'Cliente','Maschera');


          declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
   end;

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

   select Ruolo
          into var_ruolo
          from Utente
          where Username = var_amministratore;


      if var_ruolo <> 'Amministratore' then
                          signal sqlstate '45002' set message_text =" l'utente non è un amministratore ";
      end if;


          select C.nome
```

```
        from Cinema C join Dipendente D on C.nome = D.Cinema, Proiezione_film P
        where  D.matricola not in (select S.Dipendente
                                   from Servizio S join Turno_di_lavoro T on
S.Turno_di_lavoro = T.Codice
                    where tipo ='Maschera' and T.Orario_inizio >= C.Orario_apertura and T.Orario_fine <=
C.Orario_chiusura )

group by C.nome
having count(D.Matricola)>=2;

select P.ID as Proiezione, F.nome as Film
        from  Proiezione_film P join Film F on Film = F.Id
        where  P.Proiezionista is null;

   commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure Aggiungi_proiezione
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`Aggiungi_proiezione`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Aggiungi_proiezione` (IN var_data_proiezione varchar(20), IN var_ora_proiezione
varchar(20) ,IN var_film INT, IN var_cinema VARCHAR(30), IN var_sala INT, IN var_proiezionista INT)
BEGIN
        declare var_data_date DATE;
   declare var_ora_time TIME;
   declare ID INT;
   declare var_prezzo FLOAT;

   declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
   end;
   set transaction isolation level read uncommitted;
    start transaction;

   set var_data_date = str_to_date(var_data_proiezione, '%Y-%m-%d');
        set var_ora_time = str_to_date(var_ora_proiezione, '%T');
   set ID = last_insert_id();
```

```
   if (curtime() > "20:00:00" and var_sala = 1) then
                    set var_prezzo = 7.50;
          else
                    set var_prezzo = 5.00;
          end if;


   INSERT INTO Proiezione_film  values (var_data_date, var_ora_time,var_film, var_cinema, var_sala,ID,
var_proiezionista);
   INSERT INTO Prezzo_biglietto (Prezzo, Sala_proiezione, Ora_proiezione) values (var_prezzo, var_sala,
var_ora_time);
   commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure stampa_biglietto
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`stampa_biglietto`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `stampa_biglietto` (IN var_codice_prenotazione INT)
BEGIN

   declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
   end;

   set transaction read only;
   set transaction isolation level read committed;



SELECT distinct B.Cinema, B.Sala, B.lettera_posto, B.numero_posto, P.`Data`, P.`Ora`, F.Nome
   FROM Biglietto B, Posto T, Proiezione_film P, Film F
          WHERE B.Lettera_posto = T.Lettera_fila and B.Numero_posto = T.Numero
                    and B.Proiezione_film = P.ID
      and P.film = F.ID
      and B.Codice_prenotazione = var_codice_prenotazione ;
      commit;
END$$
```

```
DELIMITER ;

-- -----------------------------------------------------
-- procedure Convalida_biglietto
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`Convalida_biglietto`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `Convalida_biglietto` (IN var_maschera VARCHAR(45), IN var_codice_prenotazione INT)
BEGIN
        declare var_codice INT;
   declare var_ruolo ENUM('Amministratore','Cliente','Maschera');
   declare var_biglietto_convalidato ENUM('si','no');

   declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
   end;

    set transaction isolation level read committed;
    start transaction;

   select Ruolo
   into var_Ruolo
   from Utente
   where username = var_maschera;

   if var_ruolo <> 'Maschera'      then
                    signal sqlstate '45003' set message_text = " l'utente non è una maschera";
         end if;

   if var_codice_prenotazione not in (select Codice_prenotazione
                                                                                    from Biglietto) then
                    signal sqlstate '45000' set message_text = 'Codice di prenotazione non presente nel database';
   end if;

         select Biglietto_convalidato
   from Biglietto
   where Codice_prenotazione = var_codice_prenotazione into var_biglietto_convalidato;


   if var_biglietto_convalidato = 'si' then
    signal sqlstate '45000' set message_text = 'la prenotazione è stata già utilizzata';
```

```
            end if;

    Update Biglietto
    set Biglietto_convalidato ='si'
    where codice_prenotazione= var_codice_prenotazione ;
    commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure login
-- -----------------------------------------------------

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

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `login` (in var_username varchar(45), in var_pass varchar(45), out var_role INT)
BEGIN
            declare var_user_role ENUM('Amministratore','Cliente', 'Maschera');

    select `ruolo` from `Utente`
                        where `username` = var_username
        and `password` = var_pass
        into var_user_role;
             -- See the corresponding enum in the client
                        if var_user_role = 'Amministratore' then
                                    set var_role = 1;
                        elseif var_user_role = 'Cliente' then
                                    set var_role = 2;
                        elseif var_user_role = 'Maschera' then
                                    set var_role = 3;
                        else
                                    set var_role = 4;
                        end if;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure crea_utente
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`crea_utente`;
```

```
DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `crea_utente` (IN username VARCHAR(45), IN pass VARCHAR(45), IN ruolo varchar(45))
BEGIN
	insert into Utente VALUES(username, MD5(pass), ruolo);
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure aggiungi_cliente
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`aggiungi_cliente`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `aggiungi_cliente` (IN var_username VARCHAR(45), OUT var_cliente INT)
BEGIN
	insert into Cliente (Username) values (var_username);
   set var_cliente = last_insert_id();
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure annulla_prenotazione
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`annulla_prenotazione`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `annulla_prenotazione` (IN var_prenotazione INT, IN var_username varchar(45))
BEGIN

	declare var_cliente int;
   declare prenotazione int;
   declare var_ora_proiezione time;
   declare var_sala int;

   declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
```

```
    end;

    set transaction isolation level repeatable read;
    start transaction;

    select Id
    from Cliente
    where Username = var_username into var_cliente;

    if( var_cliente not in (select Cliente from Biglietto)) then
                    signal sqlstate '45000' set message_text = 'Impossibile cancellare una prenotazione non
effettuata';
            end if;

    select P.Ora
    from Proiezione_film P join Biglietto B on P.ID = B.Proiezione_film
    where B.Codice_prenotazione = var_prenotazione into var_ora_proiezione;

    select Codice_prenotazione, Sala
    into prenotazione, var_sala
    from Biglietto
    where COdice_prenotazione = var_prenotazione;

    if var_prenotazione <> prenotazione    then
                    signal sqlstate '45000' set message_text = 'prenotazione non esistente';
            end if;

    if timediff(curtime(), var_ora_proiezione) > '00:30:00' then
                    signal sqlstate '45000' set message_text = 'impossibile annullare la prenotazione : tempo
scaduto';
            end if;

    update Biglietto
    set Esito = 'Annullata'
    where Codice_prenotazione = var_prenotazione and Cliente = var_cliente;

    update Sala
    set Prenotazioni_confermate = Prenotazioni_confermate - 1 ,Prenotazioni_annullate =
Prenotazioni_annullate +1
    where Numero = var_sala;
    commit;
END$$

DELIMITER ;

-- ----------------------------------------------------
-- procedure lista_proiezioni
-- ----------------------------------------------------
```

```
USE `mydb`;
DROP procedure IF EXISTS `mydb`.`lista_proiezioni`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `lista_proiezioni` (IN var_username varchar(45)  )
BEGIN
	declare var_ruolo  ENUM("Amministratore","Maschera","Cliente");
	declare exit handler for sqlexception
   begin
      rollback;  -- rollback any changes made in the transaction
      resignal;  -- raise again the sql exception to the caller
   end;

   set transaction read only;
   set transaction isolation level read committed;

   select Ruolo
   from Utente
   where Username = var_username into var_ruolo;

   if var_ruolo <> 'Cliente' then
		signal sqlstate '45000' set message_text = "L'utente non è un cliente";
	end if;

	select F.nome as film ,  F.Durata, P.ID as codice_proiezione, P.Data, P.Ora, P.Cinema,
P.Numero_sala
   from Proiezione_film P join Film F on P. Film = F.ID;
   -- where P.data between curdate() and curdate() + interval 5 day;


END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure lista_posti_disponibili
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`lista_posti_disponibili`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `lista_posti_disponibili` (IN var_proiezione INT)
BEGIN
	declare var_sala int;
```

```
    declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

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

    select Numero_sala
    from Proiezione_film
    where ID = var_proiezione into var_sala;

            select Lettera_fila, Numero
    from Posto
    where (Lettera_fila, Numero) not in (select Lettera_posto, Numero_posto from Biglietto where
Proiezione_film = var_proiezione and Esito = "Confermata" or Esito = "Inutilizzata" )
    and Numero_sala = var_sala;
    commit;
END$$

DELIMITER ;

-- -----------------------------------------------------
-- procedure lista_dipendenti
-- -----------------------------------------------------

USE `mydb`;
DROP procedure IF EXISTS `mydb`.`lista_dipendenti`;

DELIMITER $$
USE `mydb`$$
CREATE PROCEDURE `lista_dipendenti` (IN var_amministratore varchar(45))
BEGIN
            declare var_ruolo ENUM('Amministratore', 'Cliente','Maschera');
            declare exit handler for sqlexception
    begin
        rollback;  -- rollback any changes made in the transaction
        resignal;  -- raise again the sql exception to the caller
    end;

    set transaction read only;
    set transaction isolation level read committed;

                        select Ruolo
        into var_ruolo
        from Utente
        where Username = var_amministratore;
```

```
    if var_ruolo <> 'Amministratore' then
                    signal sqlstate '45002' set message_text =" l'utente non è un amministratore ";
    end if;


        select *
  from Dipendente;

  select * from Servizio;
  commit;
END$$

DELIMITER ;
SET SQL_MODE = '';
DROP USER IF EXISTS login;
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_
DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
CREATE USER 'login' IDENTIFIED BY 'login';

GRANT EXECUTE ON procedure `mydb`.`login` TO 'login';
GRANT EXECUTE ON procedure `mydb`.`aggiungi_turno_lavoro` TO 'login';
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 `mydb`.`aggiungi_turno_lavoro` TO 'Amministratore';
GRANT EXECUTE ON procedure `mydb`.`report_prenotazioni` TO 'Amministratore';
GRANT EXECUTE ON procedure `mydb`.`Aggiungi_proiezione` TO 'Amministratore';
GRANT EXECUTE ON procedure `mydb`.`crea_utente` TO 'Amministratore';
GRANT EXECUTE ON procedure `mydb`.`aggiungi_cliente` TO 'Amministratore';
GRANT EXECUTE ON procedure `mydb`.`report_dipendenti` TO 'Amministratore';
GRANT EXECUTE ON procedure `mydb`.`lista_dipendenti` TO 'Amministratore';
SET SQL_MODE = '';
DROP USER IF EXISTS Cliente;
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 'Cliente' IDENTIFIED BY 'Cliente';

GRANT EXECUTE ON procedure `mydb`.`aggiungi_carta_di_credito` TO 'Cliente';
GRANT EXECUTE ON procedure `mydb`.`aggiungi_biglietto` TO 'Cliente';
GRANT EXECUTE ON procedure `mydb`.`annulla_prenotazione` TO 'Cliente';
```

```
GRANT EXECUTE ON procedure `mydb`.`lista_proiezioni` TO 'Cliente';
GRANT EXECUTE ON procedure `mydb`.`lista_posti_disponibili` TO 'Cliente';
SET SQL_MODE = '';
DROP USER IF EXISTS Maschera;
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 'Maschera' IDENTIFIED BY 'Maschera';

GRANT EXECUTE ON procedure `mydb`.`Convalida_biglietto` TO 'Maschera';
GRANT EXECUTE ON procedure `mydb`.`stampa_biglietto` TO 'Maschera';

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

-- ----------------------------------------------------
-- Data for table `mydb`.`Cinema`
-- ----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Cinema` (`Nome`, `Indirizzo`, `Orario_apertura`, `Orario_chiusura`) VALUES
('Augustus', 'via trevi (RM)', '09:00:00', '24:00:00');
INSERT INTO `mydb`.`Cinema` (`Nome`, `Indirizzo`, `Orario_apertura`, `Orario_chiusura`) VALUES
('Battisti', 'via alemana (RM)', '09:00:00', '24:00:00');
INSERT INTO `mydb`.`Cinema` (`Nome`, `Indirizzo`, `Orario_apertura`, `Orario_chiusura`) VALUES
('Nerone', 'via ottaviani moro (RM)', '09:00:00', '13:00:00');

COMMIT;


-- ----------------------------------------------------
-- Data for table `mydb`.`Sala`
-- ----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`,
`Prenotazioni_inutilizzate`) VALUES (1, 'Augustus', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`,
`Prenotazioni_inutilizzate`) VALUES (2, 'Augustus', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`,
`Prenotazioni_inutilizzate`) VALUES (3, 'Augustus', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`,
`Prenotazioni_inutilizzate`) VALUES (4, 'Augustus', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`,
`Prenotazioni_inutilizzate`) VALUES (5, 'Augustus', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`,
`Prenotazioni_inutilizzate`) VALUES (1, 'Battisti', DEFAULT, DEFAULT, DEFAULT);
```

```
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`, `Prenotazioni_inutilizzate`) VALUES (2, 'Battisti', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`, `Prenotazioni_inutilizzate`) VALUES (3, 'Battisti', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`, `Prenotazioni_inutilizzate`) VALUES (4, 'Battisti', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`, `Prenotazioni_inutilizzate`) VALUES (5, 'Battisti', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`, `Prenotazioni_inutilizzate`) VALUES (1, 'Nerone', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`, `Prenotazioni_inutilizzate`) VALUES (2, 'Nerone', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`, `Prenotazioni_inutilizzate`) VALUES (3, 'Nerone', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`, `Prenotazioni_inutilizzate`) VALUES (4, 'Nerone', DEFAULT, DEFAULT, DEFAULT);
INSERT INTO `mydb`.`Sala` (`Numero`, `Cinema`, `Prenotazioni_confermate`, `Prenotazioni_annullate`, `Prenotazioni_inutilizzate`) VALUES (5, 'Nerone', DEFAULT, DEFAULT, DEFAULT);

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`Film`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (1, 'saw', '01:20:00', 'Paella');
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (2, 'saw1', '01:20:00', 'Paella');
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (3, 'saw12', '01:20:00', 'Paella');
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (4, 'saw123', '01:20:00', 'Paella');
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (5, 'saw1234', '01:20:00', 'Paella');
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (6, 'saw12345', '01:20:00', 'Paella');
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (7, 'saw123456', '01:20:00', 'Paella');
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (8, 'vacanze in italia', '01:20:00', 'Paella');
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (9, 'vacanze in italia1', '01:20:00', 'Paella');
INSERT INTO `mydb`.`Film` (`ID`, `Nome`, `Durata`, `Casa_cinematografica`) VALUES (10, 'vacanze in italia12', '01:20:00', 'Paella');
```

```
COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`Cast`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Cast` (`ID`, `Nome`, `Cognome`) VALUES (1, 'jonny', 'pippo');
INSERT INTO `mydb`.`Cast` (`ID`, `Nome`, `Cognome`) VALUES (2, 'jonny', 'pluto');
INSERT INTO `mydb`.`Cast` (`ID`, `Nome`, `Cognome`) VALUES (3, 'jonny', 'plutorito');
INSERT INTO `mydb`.`Cast` (`ID`, `Nome`, `Cognome`) VALUES (4, 'jonny', 'plu');
INSERT INTO `mydb`.`Cast` (`ID`, `Nome`, `Cognome`) VALUES (5, 'alessandra', 'amoroso');
INSERT INTO `mydb`.`Cast` (`ID`, `Nome`, `Cognome`) VALUES (6, 'checco', 'zalone');
INSERT INTO `mydb`.`Cast` (`ID`, `Nome`, `Cognome`) VALUES (7, 'tom', 'cruise');
INSERT INTO `mydb`.`Cast` (`ID`, `Nome`, `Cognome`) VALUES (8, 'tom', 'zudel');

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`Partecipazione`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Partecipazione` (`Film`, `Attore`, `Personaggio`) VALUES (1, 1, 'bambola
assassina');
INSERT INTO `mydb`.`Partecipazione` (`Film`, `Attore`, `Personaggio`) VALUES (2, 1, 'bambola
assassina');
INSERT INTO `mydb`.`Partecipazione` (`Film`, `Attore`, `Personaggio`) VALUES (3, 1, 'bambola
assassina');
INSERT INTO `mydb`.`Partecipazione` (`Film`, `Attore`, `Personaggio`) VALUES (4, 1, 'bambola
assassina');
INSERT INTO `mydb`.`Partecipazione` (`Film`, `Attore`, `Personaggio`) VALUES (5, 1, 'bambola
assassina');

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`Utente`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Utente` (`username`, `password`, `ruolo`) VALUES ('francy',
'006d2143154327a64d86a264aea225f3', 'Cliente');
INSERT INTO `mydb`.`Utente` (`username`, `password`, `ruolo`) VALUES ('Luca',
'006d2143154327a64d86a264aea225f3', 'Amministratore');
```

```
INSERT INTO `mydb`.`Utente` (`username`, `password`, `ruolo`) VALUES ('roberto', 'roberto', 'Amministratore');
INSERT INTO `mydb`.`Utente` (`username`, `password`, `ruolo`) VALUES ('ugo', 'u', 'Maschera');
INSERT INTO `mydb`.`Utente` (`username`, `password`, `ruolo`) VALUES ('angelo', 'f', 'Cliente');

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`Turno_di_lavoro`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Turno_di_lavoro` (`Codice`, `Amministratore`, `Orario_inizio`, `Orario_fine`, `Data`) VALUES (2, 'roberto', '9:00:00', '13:00:00', '2020-10-10');

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`Cliente`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Cliente` (`Id`, `Username`) VALUES (1, 'francy');
INSERT INTO `mydb`.`Cliente` (`Id`, `Username`) VALUES (2, 'angelo');

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`Proiezione_film`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Proiezione_film` (`Data`, `Ora`, `Film`, `Cinema`, `Numero_sala`, `ID`, `Proiezionista`) VALUES ('2020-10-10', '10:00:00', 2, 'Augustus', 1, 1, NULL);

COMMIT;


-- -----------------------------------------------------
-- Data for table `mydb`.`Posto`
-- -----------------------------------------------------
START TRANSACTION;
USE `mydb`;
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Augustus', 1, 'A', 1);
```

```
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Augustus', 1, 'A', 2);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Augustus', 1, 'A', 3);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Augustus', 1, 'A', 4);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Augustus', 1, 'A', 5);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Augustus', 2, 'A', 1);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Augustus', 2, 'A', 2);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Augustus', 2, 'A', 3);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Battisti', 2, 'A', 2);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Battisti', 2, 'A', 3);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Nerone', 2, 'A', 2);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Nerone', 2, 'A', 5);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Nerone', 2, 'A', 7);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Nerone', 2, 'A', 9);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Nerone', 2, 'A', 22);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Nerone', 2, 'A', 23);
INSERT INTO `mydb`.`Posto` (`Cinema`, `Numero_sala`, `Lettera_fila`, `Numero`) VALUES ('Nerone', 2, 'A', 223);

insert into Dipendente (Cinema, Tipo, Nome, Cognome) values ("Augustus", "Proiezionista", "Ciro","Esposito");
insert into Dipendente (Cinema, Tipo, Nome, Cognome) values ("Augustus", "Proiezionista", "sandro","Esposito");
insert into Dipendente (Cinema, Tipo, Nome, Cognome) values ("Battisti", "Proiezionista", "Ciro","betti");
insert into Dipendente (Cinema, Tipo, Nome, Cognome) values ("Battisti", "Proiezionista", "ferdinando","olto");
insert into Dipendente (Cinema, Tipo, Nome, Cognome) values ("Nerone", "Proiezionista", "luca","furbo");
insert into Dipendente (Cinema, Tipo, Nome, Cognome, Maschera) values ("Nerone", "Maschera", "ugo","selo", "ugo");
insert into Carta_di_credito values (222245451212666, "saggio Fardella", 121, "2022-12-01",2);
insert into Carta_di_credito values (746845622122121, "francesco motiè", 127, "2022-12-01",1);
INSERT INTO Proiezione_film (`Data`, `Ora`, `Film`, `Cinema`, `Numero_sala`) values ("2020-10-10","20:00:00",1,"Augustus", 1);
```

```
INSERT INTO Proiezione_film (`Data`, `Ora`, `Film`, `Cinema`, `Numero_sala`) values ("2020-10-10","20:00:00",2,"Augustus", 2);

INSERT INTO Proiezione_film (`Data`, `Ora`, `Film`, `Cinema`, `Numero_sala`) values ("2020-10-10","14:00:00",2,"Augustus", 1);

INSERT INTO Proiezione_film (`Data`, `Ora`, `Film`, `Cinema`, `Numero_sala`) values ("2020-09-06","10:00:00",3,"Augustus", 3);

insert into Biglietto (Cinema, Sala, Lettera_posto, Numero_posto, Cliente, Proiezione_film,Ora_prenotazione, Data_prenotazione)
values ("Augustus", 1, "A", 1,1, 1,curtime(), curdate());

insert into Biglietto (Cinema, Sala, Lettera_posto, Numero_posto, Cliente, Proiezione_film, Ora_prenotazione, Data_prenotazione)
values ("Augustus", 1, "A", 2,2, 2,curtime(), curdate());

insert into Biglietto (Cinema, Sala, Lettera_posto, Numero_posto, Cliente, Proiezione_film, Ora_prenotazione, Data_prenotazione)
values ("Augustus", 1, "A", 3,2, 1,curtime(), curdate());

insert into Biglietto (Cinema, Sala, Lettera_posto, Numero_posto, Cliente, Proiezione_film, Ora_prenotazione, Data_prenotazione)
values ("Augustus", 1, "A", 4,1, 1,curtime(), curdate());

insert into Biglietto (Cinema, Sala, Lettera_posto, Numero_posto, Cliente, Proiezione_film, Ora_prenotazione, Data_prenotazione)
values ("Augustus", 1, "A", 5,1, 1,curtime(), curdate());

insert into Biglietto (Cinema, Sala, Lettera_posto, Numero_posto, Cliente, Proiezione_film, Ora_prenotazione, Data_prenotazione)
values ("Augustus", 2, "A", 1,1, 1,curtime(), curdate());

insert into Biglietto (Cinema, Sala, Lettera_posto, Numero_posto, Cliente, Proiezione_film, Ora_prenotazione, Data_prenotazione)
values ("Augustus", 2, "A", 2,1, 1,curtime(), curdate());

update Sala
set Prenotazioni_confermate = 5
where Numero = 1 and Cinema = "Augustus";

update Sala
set Prenotazioni_confermate = 2
where Numero = 2 and Cinema = "Augustus";


COMMIT;
```

```
USE `mydb`;

DELIMITER $$

USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`MassimoOreLavoro` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`MassimoOreLavoro` BEFORE INSERT ON
`Turno_di_lavoro` FOR EACH ROW
BEGIN

        IF timediff(new.Orario_fine,new.Orario_inizio)  > "08:00:00"  then
                Signal sqlstate '45000' set message_text ="eccessivo numero di ore nel turno di lavoro!";
        END IF;
END$$


USE `mydb`$$
DROP TRIGGER IF EXISTS `mydb`.`Biglietto_convalidato` $$
USE `mydb`$$
CREATE DEFINER = CURRENT_USER TRIGGER `mydb`.`Biglietto_convalidato` BEFORE UPDATE
ON `Biglietto` FOR EACH ROW
BEGIN
        if (old.Biglietto_convalidato ="si" and new.Biglietto_convalidato = OLD.Biglietto_convalidato) then
                        signal sqlstate '45000' set message_text = 'Biglietto già convalidato';
    end if;
END$$


DELIMITER ;
```

Codice del Front-End

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



#define BUFF_SIZE 4096

struct configuration {
char *host;
char *db_username;
```

```
char *db_password;
unsigned int port;
char *database;


char username[128];
char password[128];
};


// The final config struct will point into this
static char config[BUFF_SIZE];


/**
 * JSON type identifier. Basic types are:
 *      o Object
 *      o Array
 *      o String
 *      o Other primitive: number, boolean (true/false) or null
 */
typedef enum {
JSMN_UNDEFINED = 0,
JSMN_OBJECT = 1,
JSMN_ARRAY = 2,
JSMN_STRING = 3,
JSMN_PRIMITIVE = 4
} jsmntype_t;


enum jsmnerr {
/* Not enough tokens were provided */
JSMN_ERROR_NOMEM = -1,
/* Invalid character inside JSON string */
JSMN_ERROR_INVAL = -2,
/* The string is not a full JSON packet, more bytes expected */
JSMN_ERROR_PART = -3
};


/**
 * JSON token description.
 * type         type (object, array, string etc.)
 * start  start position in JSON data string
 * end          end position in JSON data string
 */
typedef struct {
jsmntype_t type;
```

```
int start;
int end;
int size;
#ifdef JSMN_PARENT_LINKS
int parent;
#endif
} jsmntok_t;


/**
 * JSON parser. Contains an array of token blocks available. Also stores
 * the string being parsed now and current position in that string
 */
typedef struct {
unsigned int pos; /* offset in the JSON string */
unsigned int toknext; /* next token to allocate */
int toksuper; /* superior token node, e.g parent object or array */
} jsmn_parser;


/**
 * Allocates a fresh unused token from the token pool.
 */
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens, size_t num_tokens) {
jsmntok_t *tok;
if (parser->toknext >= num_tokens) {
return NULL;
}
tok = &tokens[parser->toknext++];
tok->start = tok->end = -1;
tok->size = 0;
#ifdef JSMN_PARENT_LINKS
tok->parent = -1;
#endif
return tok;
}


/**
 * Fills token type and boundaries.
 */
static void jsmn_fill_token(jsmntok_t *token, jsmntype_t type,
                            int start, int end) {
token->type = type;
token->start = start;
token->end = end;
token->size = 0;
}
```

```
/**
 * Fills next available token with JSON primitive.
 */
static int jsmn_parse_primitive(jsmn_parser *parser, const char *js,
size_t len, jsmntok_t *tokens, size_t num_tokens) {
jsmntok_t *token;
int start;


start = parser->pos;


for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
switch (js[parser->pos]) {
#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 = fopen(filename, "rb");
if(f == NULL) {

fprintf(stderr, "Unable to open file %s\n", filename);
exit(1);
}

fseek(f, 0, SEEK_END);
size_t fsize = ftell(f);
fseek(f, 0, SEEK_SET);  //same as rewind(f);


if(fsize >= BUFF_SIZE) {
fprintf(stderr, "Configuration file too large\n");
abort();
}

fread(config, fsize, 1, f);
fclose(f);

config[fsize] = 0;
```

```
return fsize;
}


int parse_config(char *path, struct configuration *conf)
{
int i;
int r;
jsmn_parser p;
jsmntok_t t[128]; /* We expect no more than 128 tokens */


load_file(path);


jsmn_init(&p);
r = jsmn_parse(&p, config, strlen(config), t, sizeof(t)/sizeof(t[0]));
if (r < 0) {
printf("Failed to parse JSON: %d\n", r);
return 0;
}


/* Assume the top-level element is an object */
if (r < 1 || t[0].type != JSMN_OBJECT) {
printf("Object expected\n");
return 0;
}


/* Loop over all keys of the root object */
for (i = 1; i < r; i++) {
if (jsoneq(config, &t[i], "host") == 0) {
/* We may use 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;
}

#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <termios.h>
#include <sys/ioctl.h>
#include <pthread.h>
#include <signal.h>
#include <stdbool.h>


#include "defines.h"


// Per la gestione dei segnali
static volatile sig_atomic_t signo;
typedef struct sigaction sigaction_t;
static void handler(int s);


char *getInput(unsigned int lung, char *stringa, bool hide)
{
char c;
unsigned int i;


// Dichiara le variabili necessarie ad un possibile mascheramento dell'input
sigaction_t sa, savealrm, saveint, savehup, savequit, saveterm;
sigaction_t savetstp, savettin, savettou;
struct termios term, oterm;


if(hide) {
// Svuota il buffer
(void) fflush(stdout);


// Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente senza output sulla shell
sigemptyset(&sa.sa_mask);
```

```
sa.sa_flags = SA_INTERRUPT; // Per non resettare le system call
sa.sa_handler = handler;
(void) sigaction(SIGALRM, &sa, &savealrm);
(void) sigaction(SIGINT, &sa, &saveint);
(void) sigaction(SIGHUP, &sa, &savehup);
(void) sigaction(SIGQUIT, &sa, &savequit);
(void) sigaction(SIGTERM, &sa, &saveterm);
(void) sigaction(SIGTSTP, &sa, &savetstp);
(void) sigaction(SIGTTIN, &sa, &savettin);
(void) sigaction(SIGTTOU, &sa, &savettou);

// Disattiva l'output su schermo
if (tcgetattr(fileno(stdin), &oterm) == 0) {
(void) memcpy(&term, &oterm, sizeof(struct termios));
term.c_lflag &= ~(ECHO|ECHONL);
(void) tcsetattr(fileno(stdin), TCSAFLUSH, &term);
} else {
(void) memset(&term, 0, sizeof(struct termios));
(void) memset(&oterm, 0, sizeof(struct termios));
}
}

// Acquisisce da tastiera al più lung - 1 caratteri
for(i = 0; i < lung; i++) {
(void) fread(&c, sizeof(char), 1, stdin);
if(c == '\n') {
stringa[i] = '\0';
break;
} else
stringa[i] = c;


// Gestisce gli asterischi
if(hide) {
if(c == '\b') // Backspace
(void) write(fileno(stdout), &c, sizeof(char));
else
(void) write(fileno(stdout), "*", sizeof(char));
}
}

// Controlla che il terminatore di stringa sia stato inserito
if(i == lung - 1)
stringa[i] = '\0';


// Se sono stati digitati più caratteri, svuota il buffer della tastiera
if(strlen(stringa) >= lung) {
```

```
// Svuota il buffer della tastiera
do {
c = getchar();
} while (c != '\n');
}


if(hide) {
//L'a capo dopo l'input
(void) write(fileno(stdout), "\n", 1);


// Ripristina le impostazioni precedenti dello schermo
(void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);


// Ripristina la gestione dei segnali
(void) sigaction(SIGALRM, &savealrm, NULL);
(void) sigaction(SIGINT, &saveint, NULL);
(void) sigaction(SIGHUP, &savehup, NULL);
(void) sigaction(SIGQUIT, &savequit, NULL);
(void) sigaction(SIGTERM, &saveterm, NULL);
(void) sigaction(SIGTSTP, &savetstp, NULL);
(void) sigaction(SIGTTIN, &savettin, NULL);
(void) sigaction(SIGTTOU, &savettou, NULL);


// Se era stato ricevuto un segnale viene rilanciato al processo stesso
if(signo)
(void) raise(signo);
}

return stringa;
}


// Per la gestione dei segnali
static void handler(int s) {
signo = s;
}




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

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

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

// Decide quale delle due lettere mostrare come predefinite
char s, n;
if(predef) {
s = toupper(yes);
n = no;
} else {
s = yes;
n = toupper(no);
}


// Richiesta della risposta
while(true) {
// Mostra la domanda
printf("%s [%c/%c]: ", domanda, s, n);


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

// Controlla quale risposta è stata data
if(c == '\0') { // getInput() non può restituire '\n'!
return predef;
} else if(c == yes) {
return true;
} else if(c == no) {
return false;
} else if(c == toupper(yes)) {
if(predef || insensitive) return true;
} else if(c == toupper(yes)) {
if(!predef || insensitive) return false;
}
}
}


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


// Genera la stringa delle possibilità
char *possib = malloc(2 * num * sizeof(char));
int i, j = 0;
for(i = 0; i < num; i++) {
possib[j++] = choices[i];
```

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


// Chiede la risposta
while(true) {
// Mostra la domanda
printf("%s [%s]: ", domanda, possib);


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


// Controlla se è un carattere valido
for(i = 0; i < num; i++) {
if(c == choices[i])
return c;
}
}
}

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


#include "defines.h"


void print_stmt_error (MYSQL_STMT *stmt, char *message)
{
fprintf (stderr, "%s\n", message);
if (stmt != NULL) {
fprintf (stderr, "Error %u (%s): %s\n",
mysql_stmt_errno (stmt),
mysql_stmt_sqlstate(stmt),
mysql_stmt_error (stmt));
}
}




void print_error(MYSQL *conn, char *message)
{
fprintf (stderr, "%s\n", message);
```

```
if (conn != NULL) {
#if MYSQL_VERSION_ID >= 40101
fprintf (stderr, "Error %u (%s): %s\n",
mysql_errno (conn), mysql_sqlstate(conn), mysql_error (conn));
#else
fprintf (stderr, "Error %u: %s\n",
mysql_errno (conn), mysql_error (conn));
#endif
}
}


bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn)
{
bool update_length = true;


*stmt = mysql_stmt_init(conn);
if (*stmt == NULL)
{
print_error(conn, "Could not initialize statement handler");
return false;
}


if (mysql_stmt_prepare (*stmt, statement, strlen(statement)) != 0) {
print_stmt_error(*stmt, "Could not prepare statement");
return false;
}


mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);


return true;
}


void finish_with_error(MYSQL *conn, char *message)
{
print_error(conn, message);
mysql_close(conn);
exit(EXIT_FAILURE);
}


void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool close_stmt)
{
```

```
print_stmt_error(stmt, message);
if(close_stmt)    mysql_stmt_close(stmt);
mysql_close(conn);
exit(EXIT_FAILURE);
}


static void print_dashes(MYSQL_RES *res_set)
{
MYSQL_FIELD *field;
unsigned int i, j;


mysql_field_seek(res_set, 0);
putchar('+');
for (i = 0; i < mysql_num_fields(res_set); i++) {
field = mysql_fetch_field(res_set);
for (j = 0; j < field->max_length + 2; j++)
putchar('-');
putchar('+');
}
putchar('\n');
}


static void dump_result_set_header(MYSQL_RES *res_set)
{
MYSQL_FIELD *field;
unsigned long col_len;
unsigned int i;


/* determine column display widths -- requires result set to be */
/* generated with mysql_store_result(), not mysql_use_result() */


mysql_field_seek (res_set, 0);


for (i = 0; i < mysql_num_fields (res_set); i++) {
field = mysql_fetch_field (res_set);
col_len = strlen(field->name);


if (col_len < field->max_length)
col_len = field->max_length;
if (col_len < 4 && !IS_NOT_NULL(field->flags))
col_len = 4; /* 4 = length of the word "NULL" */
```

```
field->max_length = col_len; /* reset column info */
}

print_dashes(res_set);
putchar('|');
mysql_field_seek (res_set, 0);
for (i = 0; i < mysql_num_fields(res_set); i++) {
field = mysql_fetch_field(res_set);
printf(" %-*s |", (int)field->max_length, field->name);
}
putchar('\n');


print_dashes(res_set);
}


void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title)
{
int i;
int status;
int num_fields;       /* number of columns in result */
MYSQL_FIELD *fields;  /* for result set metadata */
MYSQL_BIND *rs_bind;  /* for output buffers */
MYSQL_RES *rs_metadata;
MYSQL_TIME *date;
size_t attr_size;


/* Prefetch the whole result set. This in conjunction with
* STMT_ATTR_UPDATE_MAX_LENGTH set in `setup_prepared_stmt`
* updates the result set metadata which are fetched in this
* function, to allow to compute the actual max length of
* the columns.
*/
if (mysql_stmt_store_result(stmt)) {
fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
exit(0);
}


/* the column count is > 0 if there is a result set */
/* 0 if the result is only the final status packet */
num_fields = mysql_stmt_field_count(stmt);


if (num_fields > 0) {
```

```
/* there is a result set to fetch */
printf("%s\n", title);


if((rs_metadata = mysql_stmt_result_metadata(stmt)) == NULL) {
finish_with_stmt_error(conn, stmt, "Unable to retrieve result metadata\n", true);
}


dump_result_set_header(rs_metadata);

fields = mysql_fetch_fields(rs_metadata);


rs_bind = (MYSQL_BIND *)malloc(sizeof (MYSQL_BIND) * num_fields);
if (!rs_bind) {
finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
}
memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);


/* set up and bind result set output buffers */
for (i = 0; i < num_fields; ++i) {


// Properly size the parameter buffer
switch(fields[i].type) {
case MYSQL_TYPE_DATE:
case MYSQL_TYPE_TIMESTAMP:
case MYSQL_TYPE_DATETIME:
case MYSQL_TYPE_TIME:
attr_size = sizeof(MYSQL_TIME);
break;
case MYSQL_TYPE_FLOAT:
attr_size = sizeof(float);
break;
case MYSQL_TYPE_DOUBLE:
attr_size = sizeof(double);
break;
case MYSQL_TYPE_TINY:
attr_size = sizeof(signed char);
break;
case MYSQL_TYPE_SHORT:
case MYSQL_TYPE_YEAR:
attr_size = sizeof(short int);
break;
case MYSQL_TYPE_LONG:
case MYSQL_TYPE_INT24:
```

```
attr_size = sizeof(int);
break;
case MYSQL_TYPE_LONGLONG:
attr_size = sizeof(int);
break;
default:
attr_size = fields[i].max_length;
break;
}

// Setup the binding for the current parameter
rs_bind[i].buffer_type = fields[i].type;
rs_bind[i].buffer = malloc(attr_size + 1);
rs_bind[i].buffer_length = attr_size + 1;


if(rs_bind[i].buffer == NULL) {
finish_with_stmt_error(conn, stmt, "Cannot allocate output buffers\n", true);
}
}


if(mysql_stmt_bind_result(stmt, rs_bind)) {
finish_with_stmt_error(conn, stmt, "Unable to bind output parameters\n", true);
}


/* fetch and display result set rows */
while (true) {
status = mysql_stmt_fetch(stmt);


if (status == 1 || status == MYSQL_NO_DATA)
break;


putchar('|');


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


if (rs_bind[i].is_null_value) {
printf (" %-*s |", (int)fields[i].max_length, "NULL");
continue;
}
```

```
switch (rs_bind[i].buffer_type) {

case MYSQL_TYPE_VAR_STRING:
case MYSQL_TYPE_DATETIME:
printf(" %-*s |", (int)fields[i].max_length, (char*)rs_bind[i].buffer);
break;


case MYSQL_TYPE_TIME:
date = (MYSQL_TIME *)rs_bind[i].buffer;
printf(" %d:%02d:%02d |", date->hour, date->minute, date->second);
break;

case MYSQL_TYPE_DATE:
case MYSQL_TYPE_TIMESTAMP:
date = (MYSQL_TIME *)rs_bind[i].buffer;
printf(" %d-%02d-%02d |", date->year, date->month, date->day);
break;

case MYSQL_TYPE_STRING:
printf(" %-*s |", (int)fields[i].max_length, (char *)rs_bind[i].buffer);
break;

case MYSQL_TYPE_FLOAT:
case MYSQL_TYPE_DOUBLE:
printf(" %.02f |", *(float *)rs_bind[i].buffer);
break;

case MYSQL_TYPE_LONG:
case MYSQL_TYPE_SHORT:
case MYSQL_TYPE_TINY:
printf(" %-*d |", (int)fields[i].max_length, *(int *)rs_bind[i].buffer);
break;

case MYSQL_TYPE_NEWDECIMAL:
printf(" %-*.02lf |", (int)fields[i].max_length, *(float*) rs_bind[i].buffer);
break;

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

```
mysql_free_result(rs_metadata); /* free metadata */


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


static void print_ticket(int prenotazione, MYSQL *conn){
MYSQL_BIND param[2];
MYSQL_STMT *prepared_stmt;




// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call stampa_biglietto(?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize code insertion statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));


param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &prenotazione;
param[0].buffer_length = sizeof(prenotazione);


//bind the parameters
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for code insertion\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while printing the ticket.");
}
// Dump the result set
dump_result_set(conn, prepared_stmt, "\nticket");
```

```
mysql_stmt_close(prepared_stmt);
}



static void booking_validation(MYSQL *conn){
MYSQL_BIND param[2];
MYSQL_STMT *prepared_stmt;
bool esito;


// Input for the registration routine
char codice_prenotazione[46];
int codice_prenotazione_int;


// Get the required information
printf("Booking code:");
getInput(46,codice_prenotazione,false);






// Apply proper type conversions
codice_prenotazione_int = atoi(codice_prenotazione);


// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call convalida_biglietto(?, ?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize code insertion statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));


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


param[1].buffer_type =MYSQL_TYPE_LONG;
param[1].buffer = &codice_prenotazione_int;
param[1].buffer_length = sizeof(codice_prenotazione_int);
```

```
//bind the parameters
if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for code insertion\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while validated the booking code.");
}
else {
printf("booking code correctly validated...\n");
}


mysql_stmt_close(prepared_stmt);


esito = yesOrNo("Vuoi stampare il biglietto convalidato?", 's', 'n', false, true);
if(esito)
print_ticket(codice_prenotazione_int,conn);
else
return;
}







void run_as_mask(MYSQL *conn){
char options[3] = {'1','2'};
char op;


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


if(!parse_config("/home/roberto/Scrivania/db/maschera.json", &conf)) {
fprintf(stderr, "Unable to load mask configuration\n");
exit(EXIT_FAILURE);
}


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

```
exit(EXIT_FAILURE);
}
while(true){
printf("\033[2J\033[H");
printf("*** What should I do for you? ***\n\n");
printf("1) convalidate booking code\n");
printf("2) Quit\n");


op = multiChoice("Select an option", options, 2);


switch(op){
case '1':
booking_validation(conn);
break;
case '2':
return;




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







}


getchar();
}


}

static void add_ticket(MYSQL *conn){
MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[6];
int status;
```

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

// Prepare parameters
memset(param, 0, sizeof(param));



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


if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for add ticket insertion\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve projections list\n", true);
}


// We have multiple result sets here!
do {
// Skip OUT variables (although they are not present in the procedure...)
if(conn->server_status & SERVER_PS_OUT_PARAMS) {
goto next;
}


dump_result_set(conn, prepared_stmt, "");



// more results? -1 = no, >0 = error, 0 = yes (keep looking)
   next:
status = mysql_stmt_next_result(prepared_stmt);
if (status > 0)
finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
} while (status == 0);
```

```
mysql_stmt_close(prepared_stmt);


// Input for the registration routine
char lettera_posto;
char numero_posto[5];
char proiezione[46];
int numero_posto_int;
float prezzo_float;
int proiezione_int;
int codice_prenotazione;

printf("\n\n");


// Get the required information
printf("Projection ID: ");
getInput(46,proiezione,false);




//convert values
proiezione_int = atoi(proiezione);


if(!setup_prepared_stmt(&prepared_stmt, "call lista_posti_disponibili(?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize add_ticket statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));


param[0].buffer_type =MYSQL_TYPE_LONG;
param[0].buffer = &proiezione_int;
param[0].buffer_length = sizeof(proiezione_int);


if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for add seat insertion\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while adding the seat.");
```

```
}


// We have multiple result sets here!
do {
// Skip OUT variables (although they are not present in the procedure...)
if(conn->server_status & SERVER_PS_OUT_PARAMS) {
goto next2;
}


dump_result_set(conn, prepared_stmt, "");




// more results? -1 = no, >0 = error, 0 = yes (keep looking)
   next2:
status = mysql_stmt_next_result(prepared_stmt);
if (status > 0)
finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
} while (status == 0);



mysql_stmt_close(prepared_stmt);




// Get the required information
printf("Seat letter: ");
getInput(1, &lettera_posto,false);
printf("Seat number: ");
getInput(2, numero_posto,false);




//convert values
numero_posto_int = atoi(numero_posto);



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

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


// Prepare parameters
memset(param, 0, sizeof(param));


param[0].buffer_type =MYSQL_TYPE_STRING;
param[0].buffer = &lettera_posto;
param[0]. buffer_length = sizeof(char);


param[1].buffer_type =MYSQL_TYPE_LONG;
param[1].buffer = &numero_posto_int;
param[1]. buffer_length = sizeof(numero_posto_int);


param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = conf.username;
param[2].buffer_length = strlen(conf.username);


param[3].buffer_type =MYSQL_TYPE_LONG;
param[3].buffer = &proiezione_int;
param[3].buffer_length = sizeof(proiezione_int);


param[4].buffer_type =MYSQL_TYPE_FLOAT; //OUT
param[4].buffer = &prezzo_float;
param[4].buffer_length = sizeof(prezzo_float);


param[5].buffer_type =MYSQL_TYPE_LONG; //OUT
param[5].buffer = &codice_prenotazione;
param[5].buffer_length = sizeof(codice_prenotazione);







if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for add ticket insertion\n", true);
}


// Run procedure
```

```
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while adding the ticket.");
} else {
printf("\tticket correctly added...\n");
}


// Get back the ticket's price
// Prepare parameters
memset(param, 0, sizeof(param));


param[0].buffer_type = MYSQL_TYPE_FLOAT; // OUT
param[0].buffer = &prezzo_float;
param[0].buffer_length = sizeof(prezzo_float);


param[1].buffer_type = MYSQL_TYPE_LONG; // OUT
param[1].buffer = &codice_prenotazione;
param[1].buffer_length = sizeof(codice_prenotazione);

if(mysql_stmt_bind_result(prepared_stmt, param)) {
finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
}

// Retrieve output parameter
if(mysql_stmt_fetch(prepared_stmt)) {
finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
}
printf("\tticket price:  %4.2f\n", prezzo_float);
printf("\tid ticket: %d", codice_prenotazione);


mysql_stmt_close(prepared_stmt);


//start routine "add_credit_card"


// Input for the registration routine
char intestatario_carta[46];
char codice_cvv[46];
char numero_carta[46];
char data_scadenza_carta[20];
int codice_cvv_int;
long long int numero_carta_long;
```

```
printf("\n\ninserire i dati della carta di credito.");

// Input for the registration routine
printf("\nNumber of credit card: ");
getInput(46,numero_carta,false);
printf("\nCVV: ");
getInput(46,codice_cvv,false);
printf("\naccountholder: ");
getInput(46,intestatario_carta, false);
printf("expiration date: ");
getInput(20,data_scadenza_carta,false);


//convert values
numero_carta_long =  strtoll (numero_carta, NULL, 10);
codice_cvv_int = atoi(codice_cvv);


if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_carta_di_credito(?,?,?,?,?,?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize add_ticket statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));


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


param[1].buffer_type = MYSQL_TYPE_LONGLONG;
param[1].buffer = &numero_carta_long;
param[1].buffer_length = sizeof(numero_carta_long);


param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = intestatario_carta;
param[2].buffer_length = strlen(intestatario_carta);


param[3].buffer_type = MYSQL_TYPE_LONG;
param[3].buffer = &codice_cvv_int;
param[3].buffer_length = sizeof(codice_cvv_int);
```

```
param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
param[4].buffer = data_scadenza_carta;
param[4].buffer_length = strlen(data_scadenza_carta);


param[5].buffer_type = MYSQL_TYPE_LONG;
param[5].buffer = &codice_prenotazione;
param[5].buffer_length = sizeof(codice_prenotazione);




if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for add credit card insertion\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while adding the credit card.");
} else {
printf("\tcredit card correctly added...\n");
}


printf("\t\t***The booking Id is: %d \t\t\tConserve this for enter to the Cinema, thank you***!", codice_prenotazione);


mysql_stmt_close(prepared_stmt);







}


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


// Input for the registration routine
char codice_prenotazione[46];
```

```
int codice_prenotazione_int;


// Get the required information
printf("booking id: ");
getInput(46,codice_prenotazione,false);


//convert values
codice_prenotazione_int = atoi(codice_prenotazione);


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


// Prepare parameters
memset(param, 0, sizeof(param));


param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &codice_prenotazione_int;
param[0].buffer_length = sizeof(codice_prenotazione_int);


param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = conf.username;
param[1].buffer_length = strlen(conf.username);


if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for delete booking\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while deleting the booking.");
} else {
printf("booking is deleted correctly...\n");
}


mysql_stmt_close(prepared_stmt);
```

```
}




void run_as_customer(MYSQL *conn)
{
char options[3] = {'1','2','3'};
char op;

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


if(!parse_config("/home/roberto/Scrivania/db/cliente.json", &conf)) {
fprintf(stderr, "Unable to load professor configuration\n");
exit(EXIT_FAILURE);
}


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


while(true) {
printf("\033[2J\033[H");
printf("*** What should I do for you? ***\n\n");
printf("1) buy_ticket\n");
printf("2) delete_booking\n");
printf("3) Quit\n");


op = multiChoice("Select an option", options, 3);


switch(op) {
case '1':
add_ticket(conn);
break;

case '2':
delete_booking(conn);
break;
case '3':
return;
```

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


getchar();
}
}

static void show_employee(MYSQL*conn){
MYSQL_STMT *prepared_stmt;
int status;
MYSQL_BIND param[1];


if(!setup_prepared_stmt(&prepared_stmt, "call lista_dipendenti(?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize work shift statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));


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


if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for employee list\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while showing the employee.");
}


// We have multiple result sets here!
do {

// Skip OUT variables (although they are not present in the procedure...)
if(conn->server_status & SERVER_PS_OUT_PARAMS) {
goto next;
```

```
}



dump_result_set(conn, prepared_stmt, "");

// more results? -1 = no, >0 = error, 0 = yes (keep looking)
   next:
status = mysql_stmt_next_result(prepared_stmt);
if (status > 0)
finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
} while (status == 0);
mysql_stmt_close(prepared_stmt);
}




static void add_work_shift(MYSQL *conn){
MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[6];


show_employee(conn); //call the store procedure "lista_dipendenti"
printf("\n\n");


// Input for the registration routine


int matricola_int;
char matricola[46];
char Orario_inizio[21];
char Orario_fine[21];
char Data[21];



        printf("enter the matricola: ");
        getInput(46, matricola, false);
        printf("Enter the time in hh:mm:ss format : ");
        getInput(21,Orario_inizio,false);
        printf("Enter the end time in hh:mm:ss format : ");
        getInput(21,Orario_fine,false);
        printf("date (in this form: 'year-month-day'):");
        getInput(21,Data,false);
```

```
        //appropriate conversions
        matricola_int = atoi(matricola);



        if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_turno_lavoro(?,?,?,?,?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize work shift statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));


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




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


param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = Orario_inizio;
param[2].buffer_length = strlen(Orario_inizio);


param[3].buffer_type = MYSQL_TYPE_VAR_STRING;
param[3].buffer = Orario_fine;
param[3].buffer_length = strlen(Orario_fine);


param[4].buffer_type = MYSQL_TYPE_LONG;
param[4].buffer = &matricola_int;
param[4].buffer_length = sizeof(matricola_int);




if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for work shift insertion\n", true);
}
```

```
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while adding the work shift.");
} else {
printf("Work shift correctly added...\n");
}


mysql_stmt_close(prepared_stmt);

}


static void employee_report(MYSQL *conn){
MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[1];
int status;







// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call report_dipendenti(?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize employee reports statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));

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


if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for employee reports\n", true);
}



// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while executing the routine.");
```

```
}


puts("\n\nnomi dei Cinema che in qualche fascia oraria sono sprovvisti di almeno due maschere e le informazioni delle proiezioni sprovvisti di proiezionista:");

// We have multiple result sets here!
do {

// Skip OUT variables (although they are not present in the procedure...)
if(conn->server_status & SERVER_PS_OUT_PARAMS) {
goto next;
}



dump_result_set(conn, prepared_stmt, "");

// more results? -1 = no, >0 = error, 0 = yes (keep looking)
   next:
status = mysql_stmt_next_result(prepared_stmt);
if (status > 0)
finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
} while (status == 0);


mysql_stmt_close(prepared_stmt);




}


static void bookings_report(MYSQL *conn) {
MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[1];


          if(!setup_prepared_stmt(&prepared_stmt, "call report_prenotazioni(?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize bookings report statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));

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

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


if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for bookings report\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while showing the bookings.");
}


// Dump the result set
dump_result_set(conn, prepared_stmt, "\nList of bookings");


mysql_stmt_close(prepared_stmt);


}




static void add_projection(MYSQL *conn){
MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[6];


// Input for the registration routine


char data_proiezione[20];
char ora_proiezione[20];
char cinema[31];
char film[46];
char sala[2];
int sala_int;
int film_int;
char proiezionista[46];
bool is_null = true;


// Get the required information
        printf("\nCinema name: ");
```

```
        getInput(31, cinema, false);
        printf("hall number: ");
        getInput(2,sala,false);
        printf("film id: ");
        getInput(46, film,false);
        printf("film screening time: ");
        getInput(20,ora_proiezione,false);
        printf("film screening date (in this form: 'year month day'): ");
        getInput(20,data_proiezione,false);


        //appropriate conversions
        sala_int = atoi(sala);
        film_int = atoi(film);




        if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_proiezione(?,?,?,?,?,?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize add projection  statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));


param[0].buffer_type = MYSQL_TYPE_VAR_STRING;
param[0].buffer = &data_proiezione;
param[0].buffer_length = strlen(data_proiezione);


param[1].buffer_type = MYSQL_TYPE_VAR_STRING;
param[1].buffer = &ora_proiezione;
param[1].buffer_length = strlen(ora_proiezione);


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




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

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


param[4].buffer_type = MYSQL_TYPE_LONG;
param[4].buffer = &sala_int;
param[4].buffer_length = sizeof(sala_int);


param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
param[5].buffer = proiezionista;
param[5].buffer_length = strlen(proiezionista);
param[5].is_null = &is_null;




if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for projection insertion\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while adding the projection.");
} else {
printf(" Projection correctly added...\n");
}


mysql_stmt_close(prepared_stmt);


}


static void create_user(MYSQL *conn){
MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[3];
char options[3] = {'1','2', '3'};
char r;


// Input for the registration routine
char username[46];
char password[46];
char ruolo[46];
```

```
// Get the required information
printf("\nUsername: ");
getInput(46, username, false);
printf("password: ");
getInput(46, password, true);
printf("Assign a possible role:\n");
printf("\t1) Cliente\n");
printf("\t2) Maschera\n");
printf("\t3) Amministratore\n");
r = multiChoice("Select role", options, 3);


// Convert role into enum value
switch(r) {
case '1':
strcpy(ruolo, "cliente");
break;
case '2':
strcpy(ruolo, "maschera");
break;
case '3':
strcpy(ruolo, "amministratore");
break;
default:
fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
abort();
}


// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call crea_utente(?, ?, ?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize user insertion statement\n", false);
}


// Prepare parameters
memset(param, 0, sizeof(param));


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


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

```
param[2].buffer_type = MYSQL_TYPE_VAR_STRING;
param[2].buffer = ruolo;
param[2].buffer_length = strlen(ruolo);


if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for user insertion\n", true);
}


// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
print_stmt_error (prepared_stmt, "An error occurred while adding the user.");
} else {
printf("User correctly added...\n");
}


mysql_stmt_close(prepared_stmt);


}


static void add_customer(MYSQL *conn){


MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[2];


// Input for the registration routine
char username[46];
int id;


// Get the required information
printf("customer username: ");
getInput(46, username, false);


// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_cliente(?, ?)", conn)) {
finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize customer insertion statement\n", false);
}
```

```
// Prepare parameters
memset(param, 0, sizeof(param));


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


param[1].buffer_type = MYSQL_TYPE_LONG; // OUT
param[1].buffer = &id;
param[1].buffer_length = sizeof(id);


if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for customer insertion\n", true);
}


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


// Get back the ID of the newly-added customer
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
param[0].buffer = &id;
param[0].buffer_length = sizeof(id);

if(mysql_stmt_bind_result(prepared_stmt, param)) {
finish_with_stmt_error(conn, prepared_stmt, "Could not retrieve output parameter", true);
}

// Retrieve output parameter
if(mysql_stmt_fetch(prepared_stmt)) {
finish_with_stmt_error(conn, prepared_stmt, "Could not buffer results", true);
}


printf("customer correctly added with ID %d...\n", id);


   out:
mysql_stmt_close(prepared_stmt);
```

```
}


void run_as_administrator(MYSQL *conn)
{
char options[7] = {'1','2', '3', '4', '5','6','7'};
char op;

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


if(!parse_config("/home/roberto/Scrivania/db/amministratore.json", &conf)) {
fprintf(stderr, "Unable to load administrator configuration\n");
exit(EXIT_FAILURE);
}


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


while(true) {
printf("\033[2J\033[H");
printf("*** What should I do for you? ***\n\n");
printf("1) add_work_shift\n");
printf("2) employee_reports\n");
printf("3) bookings_report\n");
printf("4) add_projection\n");
printf("5) create_user\n");
printf("6) add_customer\n");
printf("7) Quit\n");


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


switch(op) {

case '1':
add_work_shift(conn);
break;

case '2':
employee_report(conn);
break;
```

```
case '3':
bookings_report(conn);
break;

case '4':
add_projection(conn);
break;

case '5':
create_user(conn);
break;

case '6':
add_customer(conn);
break;

case '7':
return;

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


getchar();
}
}


typedef enum {
ADMINISTRATOR = 1,
CUSTOMER ,
MASK ,
FAILED_LOGIN
} role_t;




struct configuration conf;


static MYSQL *conn;
```

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

MYSQL_BIND param[3]; // Used both for input and output
int role = 0;


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


// Prepare parameters
memset(param, 0, sizeof(param));

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


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


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


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


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


// Prepare output parameters
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
```

```
param[0].buffer = &role;
param[0].buffer_length = sizeof(role);

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

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


mysql_stmt_close(login_procedure);
return role;


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


int main(void) {
role_t role;


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


conn = mysql_init (NULL);
if (conn == NULL) {
fprintf (stderr, "mysql_init() failed (probably out of memory)\n");
exit(EXIT_FAILURE);
}


if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database, conf.port,
NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
fprintf (stderr, "mysql_real_connect() failed\n");
mysql_close (conn);
exit(EXIT_FAILURE);
```

```
}


printf("Username: ");
getInput(128, conf.username, false);
printf("Password: ");
getInput(128, conf.password, true);


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


switch(role) {
case MASK:
run_as_mask(conn);
break;

case CUSTOMER:
run_as_customer(conn);
break;

case ADMINISTRATOR:
run_as_administrator(conn);
break;

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


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


printf("Bye!\n");


mysql_close (conn);
return 0;
}
```