Basi di Dati e Conoscenza
Progetto A.A. 2020/2021

SISTEMA DI GESTIONE DI TRASPORTO FERROVIARIO

0252789
Simone Bucci

Indice

1. Descrizione del Minimondo

Si vuole realizzare un sistema informativo per la gestione dell'operatività di un'azienda di trasporto pubblico ferroviario ad alta velocità. I treni gestiti dal servizio sono caratterizzati da una matricola (codice univoco numerico di quattro cifre). Ogni veicolo è anche associato ad una data di acquisto e ad uno storico di manutenzione. È inoltre di interesse mantenere informazioni legate alla marca e modello delle locomotrici e vagoni, tenendo traccia anche dei treni adibiti al trasporto merci o passeggeri. Di ciascun treno è di interesse conoscere anche da quanti vagoni questo è composto. Per i treni adibiti al trasporto passeggeri sono memorizzati il numero di carrozze di prima e seconda classe, il numero massimo di passeggeri che possono viaggiare in ciascun vagone (con riferimento ai singoli posti). I vagoni di un treno merci devono essere caratterizzati dalla portata massima (in tonnellate).

Ciascuna tratta ha un insieme di fermate identificate dal nome della stazione, dalla città e dalla provincia in cui si trova. Inoltre, per ciascuna tratta, vengono mantenuti i capilinea di partenza ed arrivo. Ciascuna fermata è associata all'orario di arrivo e partenza previsti, eccezion fatta per i capilinea in cui uno solo dei due orari è disponibile. Ogni tratta viene coperta da un numero predefinito di treni, la cui associazione viene gestita dai gestori del servizio. I gestori possono, su base periodica, modificare il numero di treni operanti su ciascuna tratta. Gli amministratori del servizio gestiscono anche i conducenti, identificati da un codice fiscale, un nome, un cognome, una data di nascita ed un luogo di nascita.

Nella gestione degli orari di lavoro, i gestori del servizio devono garantire che ciascun macchinista non effettui più di 5 turni settimanali, per un massimo di 4 ore di lavoro. A ciascun treno passeggeri deve anche essere associato un capotreno, soggetto agli stessi vincoli orari. La gestione dei turni avviene da parte dei gestori del servizio su base mensile. Qualora un conducente o un capotreno si ponga in malattia, i gestori del servizio devono poter indicare che il lavoratore non ha coperto il turno per malattia e identificare un nuovo lavoratore cui assegnare la sostituzione del turno.

Ogni lavoratore ha la possibilità di generare un report sui propri turni di lavoro, su base settimanale, riportante l'indicazione degli orari e dei treni in cui esso è coinvolto.

Si vuole anche realizzare, per quanto riguarda i treni passeggeri, una funzionalità di prenotazione di biglietto. All'atto di acquisto di un biglietto (identificato per tratta e cui viene associato un posto disponibile sul treno con l'aggiunta di un codice di prenotazione univoco) l'acquirente deve indicare il proprio nome, cognome data di nascita, codice fiscale e numero di carta di credito.

I gestori del servizio devono poter associare a ciascun vagone di un treno merci la tipologia della merce trasportata, della massa complessiva, della tratta su cui tale merce viene trasportata. Si vuole inoltre tenere traccia delle informazioni di fatturazione delle aziende che inviano e ricevono la merce. Per semplicità, si può assumere che in un vagone siano trasportate merci di una sola società, dirette ad una sola società.

I controllori devono poter verificare la validità di un biglietto, partendo dal codice di prenotazione. I controllori hanno accesso a tutte le informazioni del passeggero e possono contrassegnare la prenotazione come "valida ed utilizzata".

Gli addetti alla manutenzione possono inserire un report di manutenzione, indicante (in testo libero) quali riparazioni sono state effettuate in quale data. Queste informazioni sono associate alla singola locomotrice o al singolo vagone.

2. Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
3	Veicolo	Treno	Fanno riferimento allo stesso oggetto, il treno.
8	Carrozze	Vagoni	Fanno riferimento allo stesso oggetto, il vagone.
11	Tratta	Servizio Ferroviario	La tratta identifica il percorso (da stazione capolinea a stazione di arrivo) il servizio ferroviario copre una tratta in uno specifico giorno ed a specifici orari (orario arrivo e partenza per ciascuna fermata).
15	Gestori	Amministratori	Fanno riferimento alla stessa persona, l'amministratore di sistema.
16	Gestori	Amministratori	Fanno riferimento alla stessa persona, l'amministratore di sistema.
19	Gestori	Amministratori	Fanno riferimento alla stessa persona, l'amministratore di sistema.
20	Macchinista	Conducente	Fanno riferimento alla stessa persona, il conducente.
22	Gestori	Amministratori	Fanno riferimento alla stessa persona, l'amministratore di sistema.
23	Gestori	Amministratori	Fanno riferimento alla stessa persona, l'amministratore di sistema.
31	Acquirente	Passeggero	Fanno riferimento alla stessa persona, il passeggero.
33	Gestori	Amministratori	Fanno riferimento alla stessa persona, l'amministratore di sistema.
37	Società	Azienda	Fanno riferimento alla stessa entità, l'azienda.

Specifica disambiguata

Si vuole realizzare un sistema informativo per la gestione dell'operatività di un'azienda di trasporto pubblico ferroviario ad alta velocità. I treni gestiti dal servizio sono caratterizzati da una matricola (codice univoco numerico di quattro cifre). Ogni treno è anche associato ad una data di acquisto e ad uno storico di manutenzione. È inoltre di interesse mantenere informazioni legate alla marca e modello delle locomotrici e vagoni, tenendo traccia anche dei treni adibiti al trasporto merci o passeggeri. Di ciascun treno è di interesse conoscere anche da quanti vagoni questo è composto. Per i treni adibiti al trasporto passeggeri sono memorizzati il numero di vagoni di prima e seconda classe, il numero massimo di passeggeri che possono viaggiare in ciascun vagone (con riferimento ai singoli posti). I vagoni di un treno merci devono essere caratterizzati dalla portata massima (in tonnellate).
Ciascun servizio ferroviario copre una tratta e ha un insieme di fermate identificate dal nome della stazione, dalla città e dalla provincia in cui si trova. Inoltre, per ciascuna tratta, vengono mantenuti i capilinea di partenza ed arrivo. Ciascuna fermata è associata all'orario di arrivo e partenza previsti, eccezion fatta per i capilinea in cui uno solo dei due orari è disponibile. Ogni tratta viene coperta da un numero predefinito di treni, la cui associazione viene gestita dagli amministratori del servizio. Gli amministratori possono, su base periodica, modificare il numero di treni operanti su ciascuna tratta. Gli amministratori del servizio gestiscono anche i conducenti, identificati da un codice fiscale, un nome, un cognome, una data di nascita ed un luogo di nascita.
Nella gestione degli orari di lavoro, gli amministratori del servizio devono garantire che ciascun conducente non effettui più di 5 turni settimanali, per un massimo di 4 ore di lavoro. A ciascun treno passeggeri deve anche essere associato un capotreno, soggetto agli stessi vincoli orari. La gestione dei

turni avviene da parte degli amministratori del servizio su base mensile. Qualora un conducente o un capotreno si ponga in malattia, gli amministratori del servizio devono poter indicare che il lavoratore non ha coperto il turno per malattia e identificare un nuovo lavoratore cui assegnare la sostituzione del turno.
Ogni lavoratore ha la possibilità di generare un report sui propri turni di lavoro, su base settimanale, riportante l'indicazione degli orari e dei treni in cui esso è coinvolto.
Si vuole anche realizzare, per quanto riguarda i treni passeggeri, una funzionalità di prenotazione di biglietto. All'atto di acquisto di un biglietto (identificato per tratta e cui viene associato un posto disponibile sul treno con l'aggiunta di un codice di prenotazione univoco) il passeggero deve indicare il proprio nome, cognome, data di nascita, codice fiscale e numero di carta di credito.
I gestori del servizio devono poter associare a ciascun vagone di un treno merci la tipologia della merce trasportata, della massa complessiva, della tratta su cui tale merce viene trasportata. Si vuole inoltre tenere traccia delle informazioni di fatturazione delle aziende che inviano e ricevono la merce. Per semplicità, si può assumere che in un vagone siano trasportate merci di una sola azienda, dirette ad una sola azienda.
I controllori devono poter verificare la validità di un biglietto, partendo dal codice di prenotazione. I controllori hanno accesso a tutte le informazioni del passeggero e possono contrassegnare la prenotazione come "valida ed utilizzata".
Gli addetti alla manutenzione possono inserire un report di manutenzione, indicante (in testo libero) quali riparazioni sono state effettuate in quale data. Queste informazioni sono associate alla singola locomotrice o al singolo vagone.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Treno	Composto da una locomotiva e uno o più vagoni.	Veicolo	Vagoni, Locomotiva
Vagone	Componente di un treno	Carrozza	Treno
Locomotiva	Componente di un treno		Treno
Tratte	Ogni tratta può essere percorsa da uno più treni e può comprendere diverse fermate e due capilinea.		Treno, Fermata
Fermata	Stazione in cui si fermano i treni in una tratta		Tratta
Amministratore	Ente che gestisce il sistema	Gestore	
Conducente	Lavoratore a cui vengono assegnati dei turni per condurre un treno.	Macchinista	Treno

Controllore	Lavoratore che effettua la verifica e validazione del biglietto passeggero		
Capotreno	Lavoratore a cui vengono assegnati dei turni per un treno passeggeri.		Treno
Manutentore	Lavoratore che esegue attività di manutenzione sui treni e che vengono registrate sullo storico manutenzioni		
Lavoratore	Impiegato della società di gestione trasporto pubblico ferroviaria		Conducente, Capotreno
Azienda	Ente che spedisce o riceve merce	Società	
Acquirente	Persona che prenota un posto passeggero acquistando un biglietto	Passeggero	

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a treni
I treni gestiti dal servizio sono caratterizzati da una matricola (codice univoco numerico di quattro cifre). Ogni treno è anche associato ad una data di acquisto e ad uno storico di manutenzione. È inoltre di interesse mantenere informazioni legate alla marca e modello delle locomotrici e vagoni, tenendo traccia anche dei treni adibiti al trasporto merci o passeggeri. Di ciascun treno è di interesse conoscere anche da quanti vagoni questo è composto. Per i treni adibiti al trasporto passeggeri sono memorizzati il numero di vagoni di prima e seconda classe, il numero massimo di passeggeri che possono viaggiare in ciascun vagone (con riferimento ai singoli posti). I vagoni di un treno merci devono essere caratterizzati dalla portata massima (in tonnellate). Gli addetti alla manutenzione possono inserire un report di manutenzione, indicante (in testo libero) quali riparazioni sono state effettuate in quale data. Queste informazioni sono associate alla singola locomotrice o al singolo vagone.

Frasi relative a tratte
Ciascun servizio ferroviario copre una tratta e ha un insieme di fermate identificate dal nome della stazione, dalla città e dalla provincia in cui si trova. Inoltre, per ciascuna tratta, vengono mantenuti i capilinea di partenza ed arrivo. Ciascuna fermata è associata all'orario di arrivo e partenza previsti, eccezion fatta per i capilinea in cui uno solo dei due orari è disponibile. Ogni tratta viene coperta da un numero predefinito di treni, la cui associazione viene gestita dagli amministratori del servizio. Gli amministratori possono, su base periodica, modificare il numero di treni operanti su ciascuna tratta.

Frasi relative a lavoratori
Gli amministratori del servizio gestiscono anche i conducenti, identificati da un codice fiscale, un nome, un cognome, una data di nascita ed un luogo di nascita. Nella gestione degli orari di lavoro, gli amministratori del servizio devono garantire che ciascun conducente non effettui più di 5 turni settimanali, per un massimo di 4 ore di lavoro. A ciascun treno passeggeri deve anche essere associato un capotreno, soggetto agli stessi vincoli orari. La gestione dei turni avviene da parte egli amministratori del servizio su base mensile. Qualora un conducente o un capotreno si ponga in malattia, gli amministratori del servizio devono poter indicare che il lavoratore non ha coperto il turno per malattia e identificare un nuovo lavoratore cui assegnare la sostituzione del turno. Ogni lavoratore ha la possibilità di generare un report sui propri turni di lavoro, su base settimanale, riportante l'indicazione degli orari e dei treni in cui esso è coinvolto.

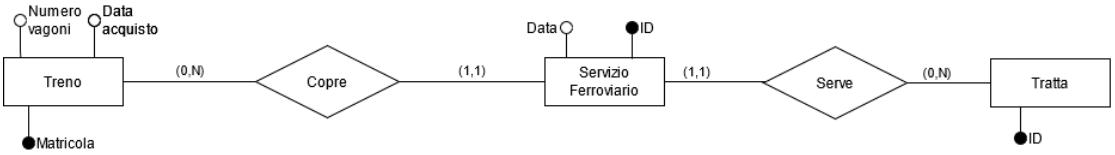
Frasi relative a passeggeri
Si vuole anche realizzare, per quanto riguarda i treni passeggeri, una funzionalità di prenotazione di biglietto. All'atto di acquisto di un biglietto (identificato per tratta e cui viene associato un posto disponibile sul treno con l'aggiunta di un codice di prenotazione univoco) l'acquirente deve indicare il proprio nome, cognome data di nascita, codice fiscale e numero di carta di credito. I controllori devono poter verificare la validità di un biglietto, partendo dal codice di prenotazione. I controllori hanno accesso a tutte le informazioni del passeggero e possono contrassegnare la prenotazione come "valida ed utilizzata".

Frasi relative a merci
I gestori del servizio devono poter associare a ciascun vagone di un treno merci la tipologia della merce trasportata, della massa complessiva, della tratta su cui tale merce viene trasportata. Si vuole inoltre tenere traccia delle informazioni di fatturazione delle aziende che inviano e ricevono la merce. Per semplicità, si può assumere che in un vagone siano trasportate merci di una sola azienda, dirette ad una sola azienda.

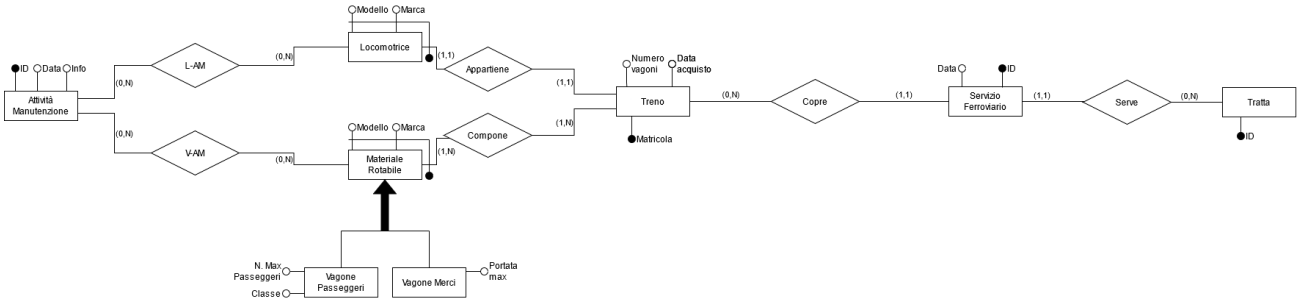
3. Progettazione concettuale

Costruzione dello schema E-R

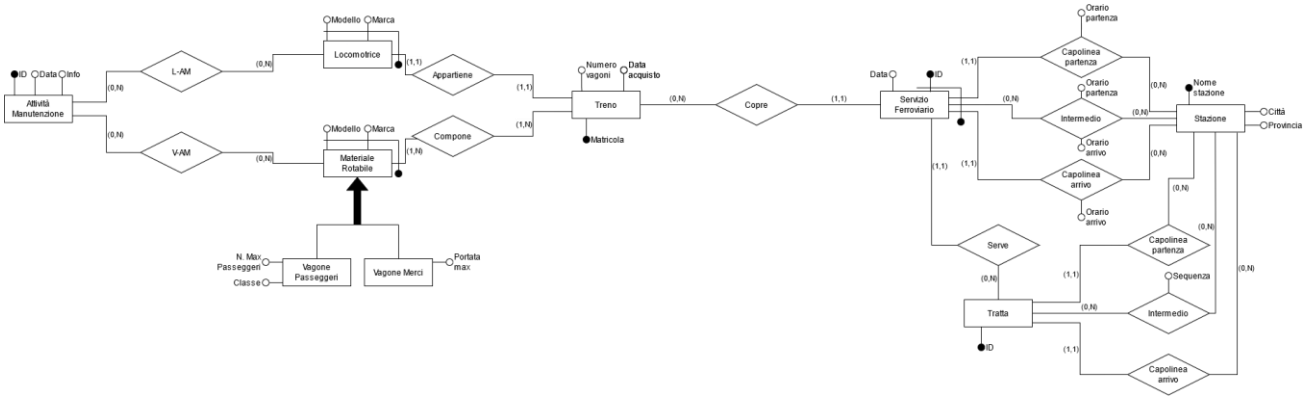
I concetti chiave sono il treno e la tratta che copre (il percorso tra due capolinea in una specifica ora del giorno). Per identificare l'occorrenza della copertura di una tratta nei diversi giorni dell'anno è stata aggiunta l'entità "Servizio Ferroviario" che potrà essere oggetto di prenotazione da parte dei passeggeri (servizio passeggeri) o da parte delle aziende (servizio merci).



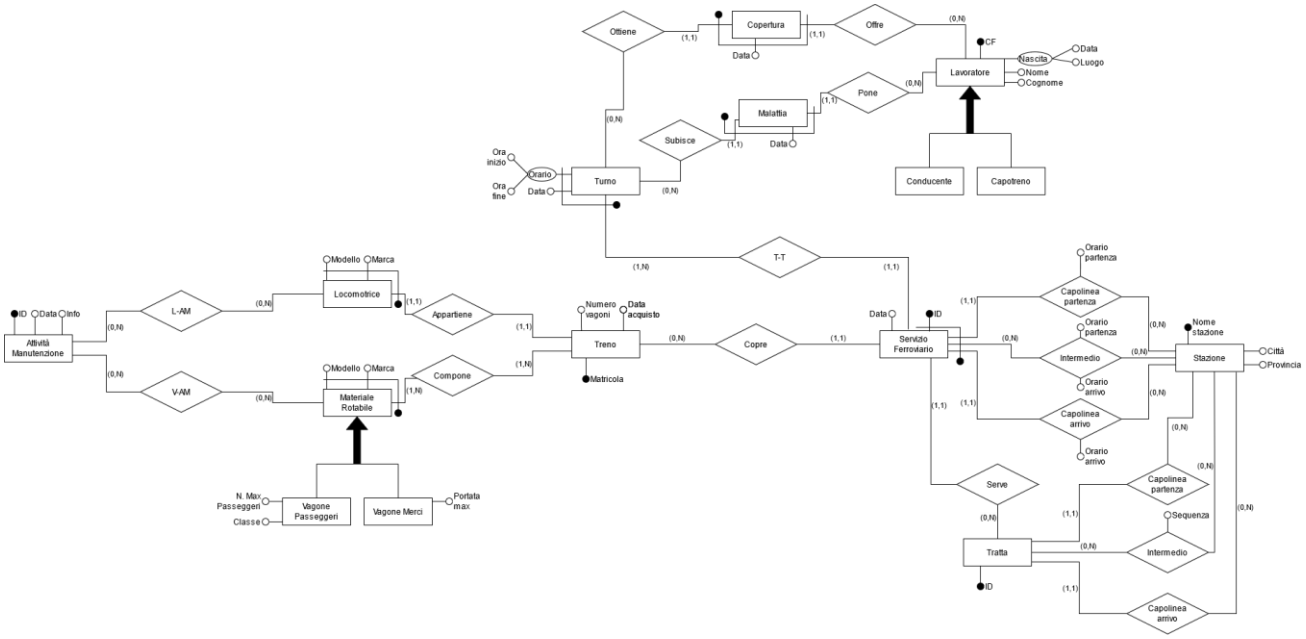
Il treno è composto da una locomotrice e più vagoni che possono essere di tipo passeggeri o merci. Per questo vengono create due entità "Locomotrice" e "Materiale Rotabile", in particolare per il materiale rotabile viene adottata una generalizzazione per differenziare i due tipi di vagone "Passeggeri" e "Merci".
Viene aggiunta inoltre l'entità "Attività Manutenzione" per lo storico degli interventi effettuati sulle singole locomotrici e vagoni.



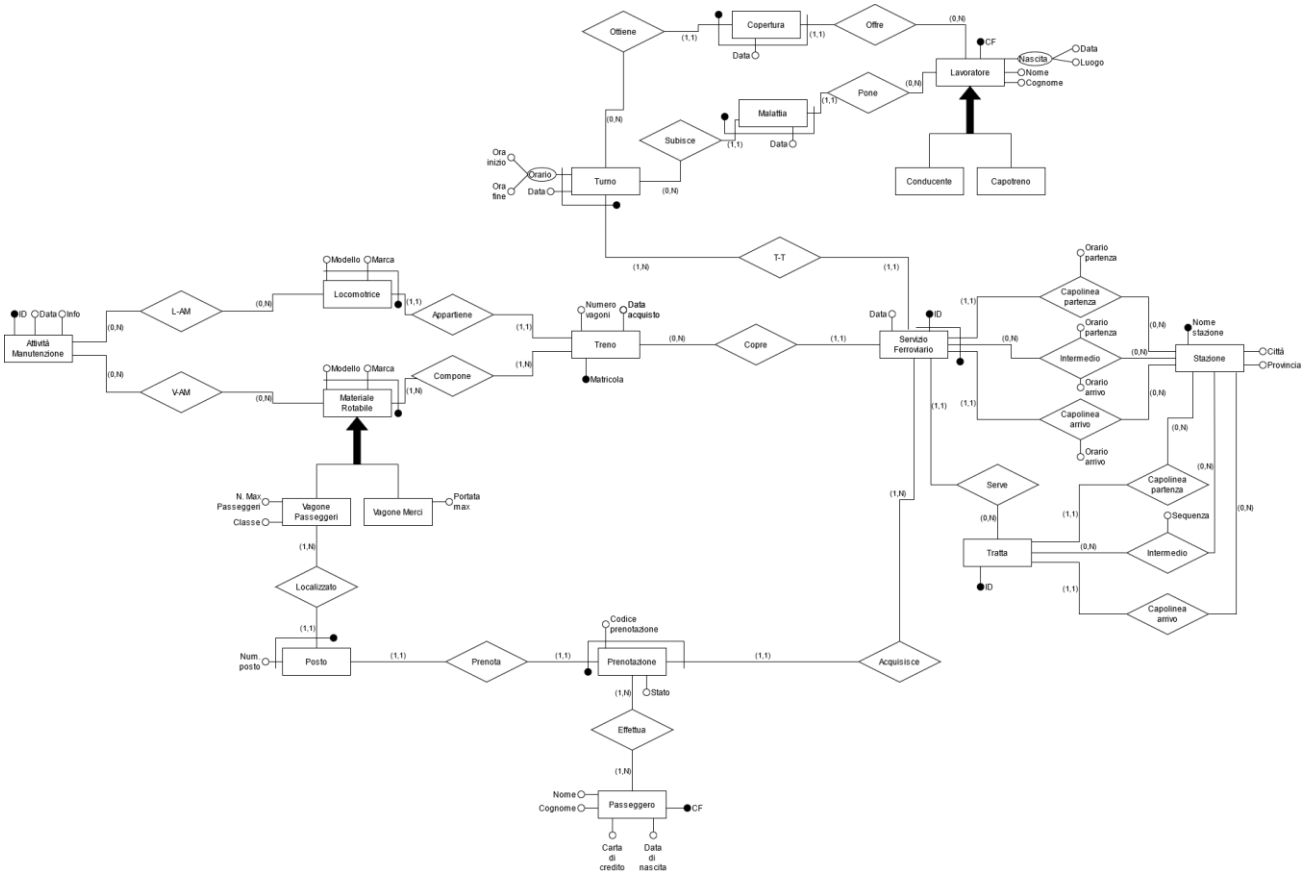
Ogni tratta è composta da un capolinea di partenza, uno di arrivo e da alcune fermate intermedie. Viene quindi creata una entità "Fermata" che verrà messa in relazione con "Tratta" e "Servizio Ferroviario" (per ottenere anche gli orari effettivi del viaggio) a seconda che sia uno dei due capolinea o una fermata intermedia.



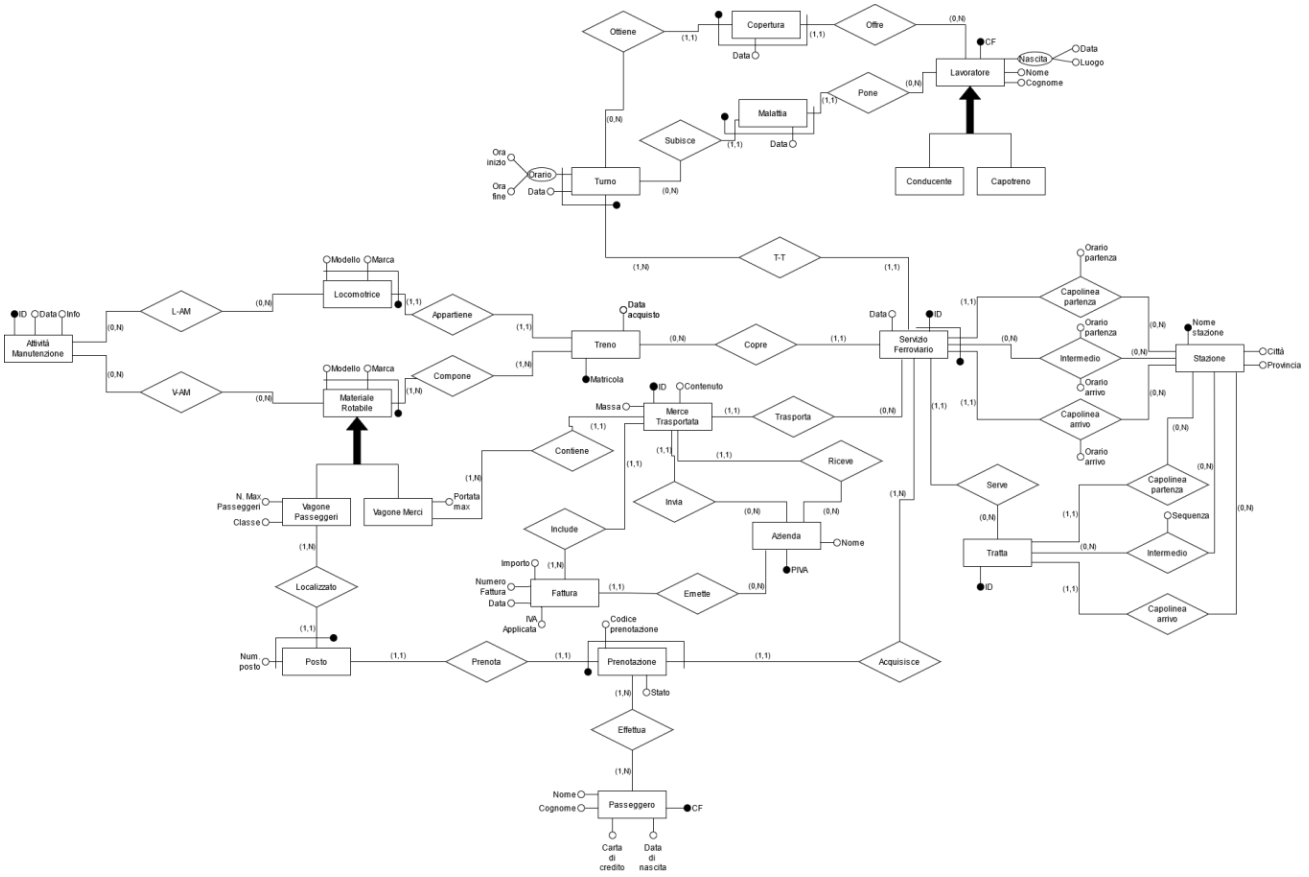
I lavoratori possono coprire o dichiararsi malati per determinati turni su determinati treni che coprono delle tratte. Sono state introdotte, quindi, due entità “Copertura” e “Malattia” messe in relazione con il lavoratore e con il turno. Ogni turno è in relazione con una o più tratte.



I passeggeri possono prenotare un posto su un vagone passeggeri di un treno che copre una determinata tratta. È stata creata quindi l’entità “Passeggero” che messa in relazione con “Prenotazione” indica la prenotazione di un determinato “Posto” su un determinato “Servizio Ferroviario” con le relative relazioni.
Le specifiche funzionali presuppongono, per semplicità, che un passeggero possa prenotare solo l’intera tratta e non percorsi parziali tra le fermate che compongono una tratta.

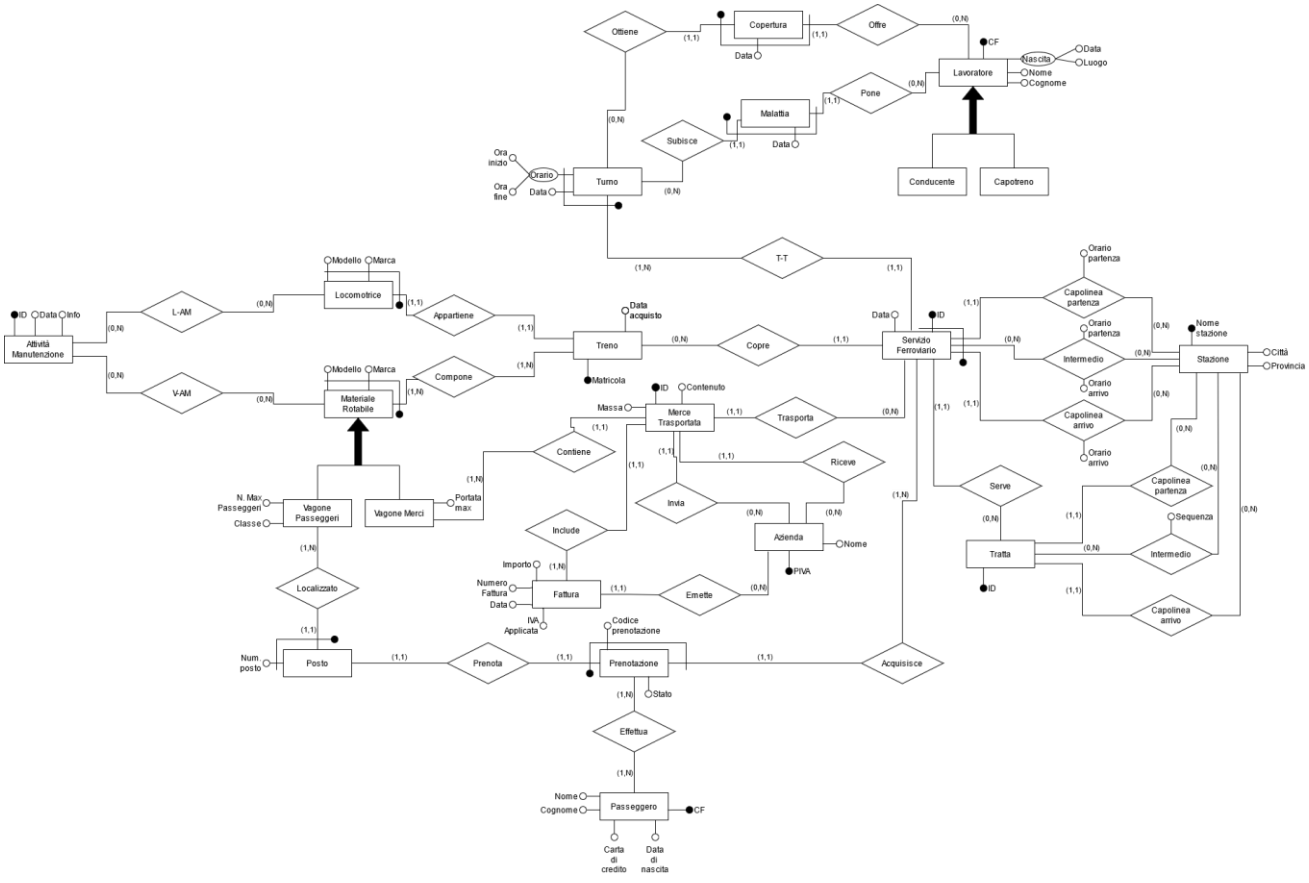


Le aziende possono far trasportare e ricevere merci attraverso i treni adibiti con vagoni merci su tratte specifiche. È stata creata infatti l'entità "Merce Trasportata" messa in relazione con il vagone merci e il servizio ferroviario su cui viaggia.
L'entità Merce Trasportata è inoltre in relazione con l'entità Fattura che riporta, oltre alle informazioni di fatturazione (data, importo, IVA applicata), anche le informazioni della ditta mittente e della ditta destinataria (tramite le due relazioni esistenti tra Fattura e l'entità Azienda).



Integrazione finale

Non sono stati trovati conflitti strutturali o sui nomi.



Regole aziendali

- I turni settimanali di ciascun conducente e capotreno non devono essere maggiori di 5
- Il numero dei turni settimanali si ottengono dal numero di "Copertura" con cui il lavoratore è in relazione
- Le ore di lavoro di ciascun conducente e capotreno devono essere massimo 4
- Le ore di lavoro di ciascun lavoratore si ottengono sottraendo l'ora di fine da quella di inizio di ogni turno e sommandole tra loro nelle date di una stessa settimana
- Il numero di vagoni di prima e di seconda classe si ottiene contando le occorrenze degli attributi corrispondenti sulle istanze dell'entità "Vagone Passeggeri"
- La disponibilità di un posto si verifica cercando una istanza di prenotazione con il numero di posto sulla tratta di interesse.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Treno	Veicolo composto da locomotrice e vagoni	Matricola, Data di acquisto	Matricola
Locomotrice	Componente trainante del treno	Modello, Marca	Modello, Marca, Treno
Materiale Rotabile	Componente che contiene passeggeri o merce del treno	Modello, Marca	Modello, Marca, Treno
Attività Manutenzione	Informazioni sull'avvenuta manutenzione di una locomotrice o vagone	ID, Data, Info	ID
Vagone Passeggeri	Tipologia di vagone	N.MaxPass. , Classe	Materiale Rotabile, Treno
Vagone Merci	Tipologia di vagone	Portata max	Materiale Rotabile, Treno
Posto	Posto di un vagone passeggeri	N. Posto	N. Posto, Materiale Rotabile, Treno
Prenotazione	Prenotazione di un posto	Codice prenotazione, Stato	Codice Prenotazione, Posto, Servizio Ferroviario
Passeggero	Persona che viaggia su un treno	Codice Fiscale, Nome, Cognome, Carta di credito, Data di nascita	Codice Fiscale
Merce Trasportata	Contenuto del vagone merci	ID, Contenuto, Massa	ID
Azienda	Azienda che usufruisce del trasporto per ricevere o inviare merci	Nome	PIVA
Servizio Ferroviario	Servizio di un treno che copre una tratta	ID	ID, Capolinea Partenza, Capolinea Arrivo, Intermedio, Fermata
Tratta	Percorso coperto da più treni che comprendono più fermate	ID	ID

Stazione	Stazione in cui i treni si possono fermare	Nome, Città, Provincia	Nome
Turno	Turno che svolge un lavoratore su un treno	Ora Inizio, Ora Fine, Data	Ora Inizio, Ora Fine, Data, Servizio Ferroviario
Copertura	Turno che un lavoratore copre in una determinata data	Data	Lavoratore, Turno, Data
Malattia	Turno in cui un lavoratore si dichiara in malattia in una determinata data	Data	Lavoratore, Turno, Data
Lavoratore	Persona che lavora per la società di gestione ferroviaria	CF, Data di nascita, Luogo di nascita, Nome, Cognome	CF
Fattura	Fattura emessa dall'azienda mittente della merce trasportata	Numero fattura, IVA applicata, Importo, Data	Numero fattura, Azienda

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo**1**	Volume atteso
Treno	E	500
Locomotrice	E	600
Materiale Rotabile	E	10000
Vagone Passeggeri	E	2000
Vagone Merci	E	8000
Attività Manutenzione	E	12000
L-AM	R	2000
V-AM	R	10000
Posto	E	100000
Prenotazione	E	1000000
Effettua	R	1000000
Passeggero	E	500000
Merce Trasportata	E	2000000
Azienda	E	10000
Servizio Ferroviario	E	4000
Tratta	E	100
Intermedio	R	16000
Capolinea Partenza	R	80
Intermedio	R	200
Capolinea Arrivo	R	80
Stazione	E	200
Turno	E	1000
Copertura	E	2000
Malattia	E	200
Lavoratore	E	3000
Conducente	E	1500
Capotreno	E	1500

Le relazioni con cardinalità (1,1) non vengono riportate poiché hanno cardinalità uguale alla entità referenziata.

I volumi attesi riportati ipotizzano una retention di un anno.

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
1	Assegnazione turni	1/mese
2	Modifica numero treni su una tratta	4/mese
3	Dichiarazione malattia	16/mese
4	Sostituzione lavoratore in malattia	16/mese
5	Aggiunta treno passeggero	1/anno
6	Aggiunta tratta	1/anno
7	Report lavoratore	4/mese

1 Indicare con E le entità, con R le relazioni

8	Prenotazione biglietto	80000/mese
9	Report manutenzione	10/mese
10	Verifica biglietto	80000/mese
11	Inserisci merce su un treno	160000/mese
12	Registrazione azienda	500/mese
13	Registrazione passeggero	40000/mese

Costo delle operazioni

Operazione 1

Concetto	Costrutto	Accessi	Tipo
Copertura	E	1000	S

Costo = 2*2000 = 4000

Operazione 2

Concetto	Costrutto	Accessi	Tipo
Serve	R	100	S

Costo = 2*4000*4/mese = 32000

Operazione 3

Concetto	Costrutto	Accessi	Tipo
Malattia	E	1	S

Costo = 2*1*16/mese = 32

Operazione 4

Concetto	Costrutto	Accessi	Tipo
Offre	R	3000	L
Copertura	E	1	S

Costo = (3000+2*1)*16/mese = 48032

Operazione 5

Concetto	Costrutto	Accessi	Tipo
Treno	E	1	S
Locomotrice	E	1	S
Vagone	E	25	S
Posto	E	30	S

Costo = (2*1+2*1+25*2+30*2)*1/anno = 114

Operazione 6

Concetto	Costrutto	Accessi	Tipo
Tratta	E	1	S

Costo = (2*1)*1/anno = 2

Operazione 7

Concetto	Costrutto	Accessi	Tipo
Turno	E	1	L

Costo = (1)*4/mese = 4

Operazione 8

Concetto	Costrutto	Accessi	Tipo
Prenotazione	E	1	S

Costo = (2*1)*80000/mese = 160000

Operazione 9

Concetto	Costrutto	Accessi	Tipo
Attività Manutenzione	E	1	S

Costo = (2*1)*10/mese = 20

Operazione 10

Concetto	Costrutto	Accessi	Tipo
Prenotazione	E	250000	L

Costo = (1000000)*80000/mese = 80000000000

Operazione 13

Concetto	Costrutto	Accessi	Tipo
Passeggero	E	1	S

Costo = (2*1)* 40000/mese = 80000

Ristrutturazione dello schema E-R

La generalizzazione che riguarda i due tipi di vagoni, passeggeri e merci è stata eliminata aggiungendo un attributo "Tipo" all'entità "Materiale Rotabile", è stata inoltre accorpata l'entità "Locomotrice". Quindi l'entità **Materiale Rotabile rappresenta tutte le tipologie di componenti che compongono un treno**: "Locomotore", "Vagone Passeggeri", "Vagone Merci" ed avranno quindi una gestione unificata dello storico manutenzioni.

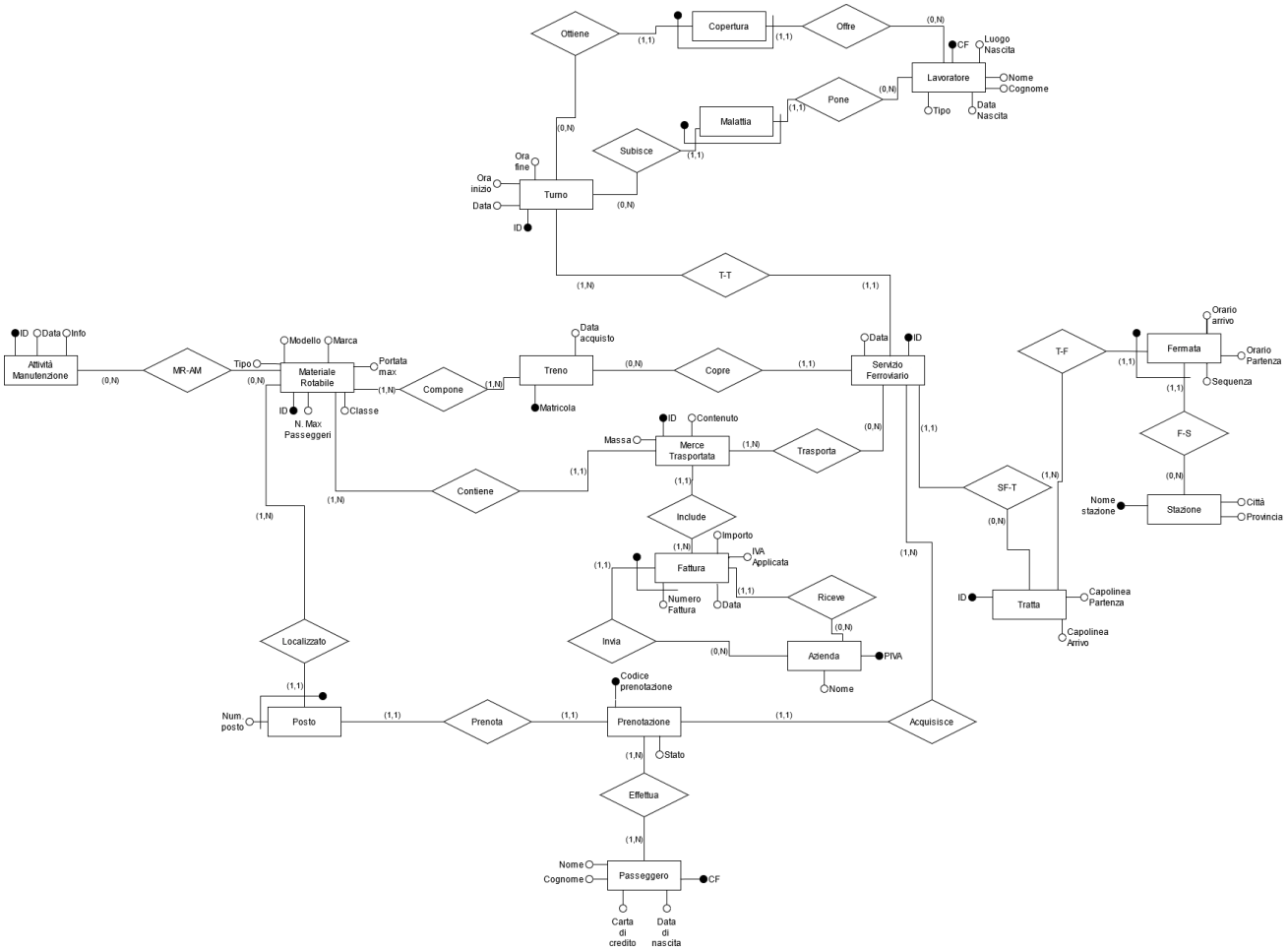
La generalizzazione che riguarda i due tipi di lavoratori è stata eliminata aggiungendo un attributo "Tipo" in sostituzione delle due entità "Conducente" e "Capotreno".

L'attributo "Numero vagoni" è stato eliminato per evitare ridondanze, in quanto determinabile dalla relazione che definisce la composizione del treno in uno o più vagoni.

L'attributo "Data" su "Copertura" e "Malattia" è stato eliminato per evitare ridondanze, in quanto determinabile dalla relazione con "Turno".

È stato introdotta l'entità "Fermata" che unifica e risolve le relazioni tra "Servizio Ferroviario" e "Stazione" e tra "Tratta" e "Stazione". Ogni tratta è quindi caratterizzata da più "Fermate" ciascuna identificata da un numero di sequenza della fermata nella tratta (oltre che ovviamente dagli orari di arrivo e partenza). La fermata con numero di sequenza 1 sarà quindi il capolinea di partenza, mentre quella con il numero di sequenza massimo sarà il capolinea di arrivo.

Per evitare ridondanze riguardo alle aziende che spediscono le merci con la relativa fattura sono state eliminate le relazioni tra "Azienda" e "Merce Trasportata", collegando le aziende direttamente a "Fattura" in modo tale da registrare chi emette e chi riceve la fattura con cui viene trasportata una determinata merce.

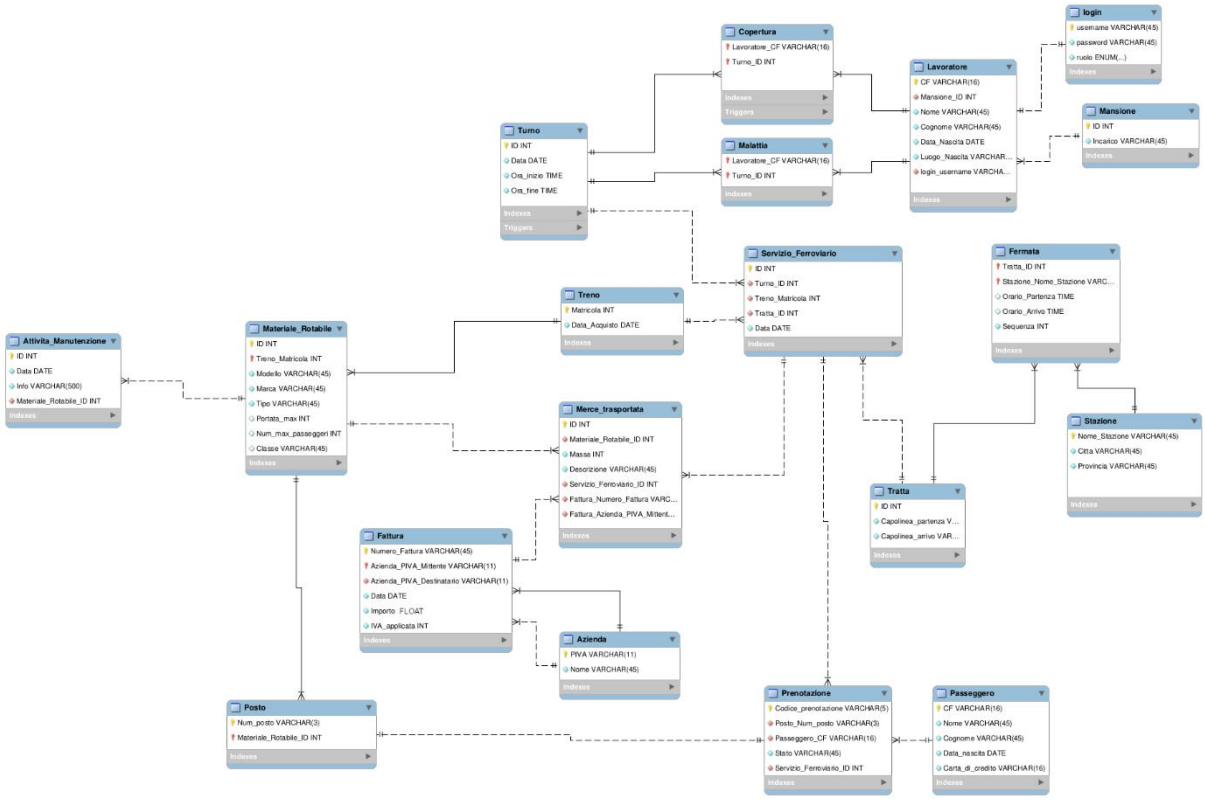


Trasformazione di attributi e identificatori

Gli attributi riguardanti la nascita del lavoratore (data e luogo) vengono separati, eliminando l'attributo composto. Stessa cosa avviene per l'attributo composto orario nell'entità Turno che viene decomposto negli attributi orario inizio e orario fine.
Viene eliminato l'identificatore esterno su turno e su materiale rotabile inserendo un attributo ID ad entrambe le entità per facilitarne l'interazione.

Traduzione di entità e associazioni

È stata inserita la tabella "login" per la gestione dei diversi utenti che potranno accedere alla base di dati con il ruolo assegnato. La tabella è identificata da un username e possiede inoltre un campo password ed un campo ruolo per indicare a quale ruolo (Amministratore, Lavoratore, Manutentore, Controllore) l'utente appartiene.



Normalizzazione del modello relazionale

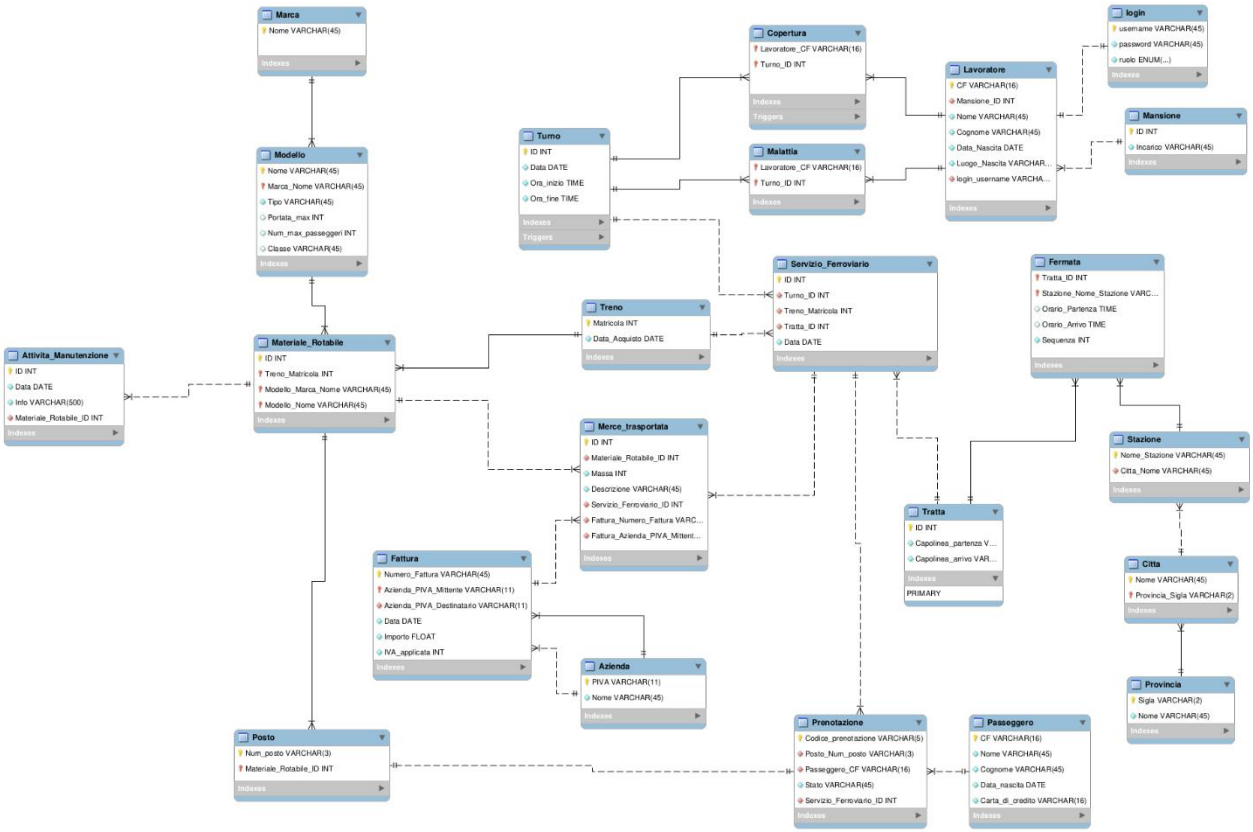
Lo schema si trova in 1NF perché tutti gli attributi sono definiti su un dominio indivisibile e non vi sono gruppi di attributi che si ripetono.

Lo schema si trova in 2NF perché tutti gli attributi non chiave dipendono dalla intera chiave.

Lo schema non si trova in 3NF perché tutti gli attributi non chiave non dipendono solamente dalla chiave. Infatti:

- L'attributo "Città" nella tabella "Stazione" è dipendente dall'attributo "Provincia".
- L'attributo "Modello" nella tabella "Materiale Rotabile" è dipendente dall'attributo "Marca" così come gli attributi "Portata max", "Num max passeggeri" e "Classe" sono dipendenti da "Modello".

Per rispettare la 3NF sono state quindi create quattro ulteriori tabelle: "Città", "Provincia", "Modello" e "Marca" e le relative relazioni.


Marca
Nome VARCHAR(45)
Indexes
Modello
Nome VARCHAR(45)
Marca_Nome VARCHAR(45)
Tipo VARCHAR(45)
Portata_max INT
Num_max_passeggeri INT
Classe VARCHAR(45)
Indexes
Attivita_Manutenzione
ID INT
Data DATE
Info VARCHAR(500)
Materiale_Rotabile_ID INT
Indexes
Materiale_Rotabile
ID INT
Treno_Matricola INT
Modello_Marca_Nome VARCHAR(45)
Modello_Nome VARCHAR(45)
Indexes
Posto
Num_posto VARCHAR(3)
Materiale_Rotabile_ID INT
Indexes
Treno
Matricola INT
Data_Acquisto DATE
Indexes
Turno
ID INT
Data DATE
Ora_inizio TIME
Ora_fine TIME
Indexes
Triggers
Copertura
Lavoratore_CF VARCHAR(16)
Turno_ID INT
Indexes
Triggers
Malattia
Lavoratore_CF VARCHAR(16)
Turno_ID INT
Indexes
Lavoratore
CF VARCHAR(16)
Mansione_ID INT
Nome VARCHAR(45)
Cognome VARCHAR(45)
Data_Nascita DATE
Luogo_Nascita VARCHAR...
login_username VARCHA...
Indexes
login
username VARCHAR(45)
password VARCHAR(45)
ruolo ENUM(...)
Indexes
Mansione
ID INT
Incarico VARCHAR(45)
Indexes
Servizio_Ferroviario
ID INT
Turno_ID INT
Treno_Matricola INT
Tratta_ID INT
Data DATE
Indexes
Merce_trasportata
ID INT
Materiale_Rotabile_ID INT
Massa INT
Descrizione VARCHAR(45)
Servizio_Ferroviario_ID INT
Fattura_Numero_Fattura VARC...
Fattura_Azienda_PIVA_Mittent...
Indexes
Fattura
Numero_Fattura VARCHAR(45)
Azienda_PIVA_Mittente VARCHAR(11)
Azienda_PIVA_Destinatario VARCHAR(11)
Data DATE
Importo FLOAT
IVA_applicata INT
Indexes
Azienda
PIVA VARCHAR(11)
Nome VARCHAR(45)
Indexes
Prenotazione
Codice_prenotazione VARCHAR(5)
Posto_Num_posto VARCHAR(3)
Passeggero_CF VARCHAR(16)
Stato VARCHAR(45)
Servizio_Ferroviario_ID INT
Indexes
Passeggero
CF VARCHAR(16)
Nome VARCHAR(45)
Cognome VARCHAR(45)
Data_nascita DATE
Carta_di_credito VARCHAR(16)
Indexes
Tratta
ID INT
Capolinea_partenza V...
Capolinea_arrivo VAR...
Indexes
PRIMARY
Fermata
Tratta_ID INT
Stazione_Nome_Stazione VARC...
Orario_Partenza TIME
Orario_Arrivo TIME
Sequenza INT
Indexes
Stazione
Nome_Stazione VARCHAR(45)
Citta_Nome VARCHAR(45)
Indexes
Citta
Nome VARCHAR(45)
Provincia_Sigla VARCHAR(2)
Indexes
Provincia
Sigla VARCHAR(2)
Nome VARCHAR(45)
Indexes

5. Progettazione fisica

Utenti e privilegi

I ruoli definiti e assegnati agli utenti ed i rispettivi privilegi sono:

- Amministratore: utente che gestisce la base di dati.
- Lavoratore: può accedere ai propri turni.
- Controllore: può verificare la validità di una prenotazione e contrassegnarla come utilizzata.
- Manutentore: può creare un resoconto di una attività di manutenzione.

A tale scopo segue l'elenco dei privilegi assegnati ai ruoli per ciascuna routine:

Permessi		
Routine	Ruolo	Tipo
aggiungi_lavoratore	Amministratore	EXECUTE
aggiungi_azienda	Amministratore	EXECUTE
aggiungi_passeggero	Amministratore	EXECUTE
aggiungi_prenotazione	Amministratore	EXECUTE
aggiungi_servizio	Amministratore	EXECUTE
aggiungi_spedizione	Amministratore	EXECUTE
aggiungi_turno	Amministratore	EXECUTE
lavoratori_disponibili	Amministratore	EXECUTE
richiesta_malattia	Amministratore	EXECUTE
assegna_turno	Amministratore	EXECUTE
configurazione_treno	Amministratore	EXECUTE
composizione_treno	Amministratore	EXECUTE
modifica_treno_servizio	Amministratore	EXECUTE
modifica_tratta_servizio	Amministratore	EXECUTE
fermate_tratta	Amministratore	EXECUTE
tratte_giornaliere	Amministratore	EXECUTE
report_lavoratore	Lavoratore	EXECUTE
verifica_prenotazione	Controllore	EXECUTE
convalida_prenotazione	Controllore	EXECUTE
report_manutenzione	Manutentore	EXECUTE

Strutture di memorizzazione

Tabella Treno		
Attributo	Tipo di dato	Attributi[2]
Matricola	VARCHAR(4)	PK
Data_Acquisto	DATE	NN

Tabella Materiale_Rotabile		
Attributo	Tipo di dato	Attributi[2]
ID	INT	PK
Treno_Matricola	INT	PK

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.

Modello_Nome	VARCHAR(45)	PK

Tabella Modello		
Attributo	Tipo di dato	Attributi2
Nome	VARCHAR(45)	PK
Marca_Nome	VARCHAR(45)	PK
Tipo	VARCHAR(45)	NN
Num_max_passeggeri	INT	
Portata_max	INT	
Classe	VARCHAR(45)	

Tabella Marca		
Attributo	Tipo di dato	Attributi2
Nome	VARCHAR(45)	PK

Tabella Attivita_Manutenzione		
Attributo	Tipo di dato	Attributi2
ID	INT	PK, AI
Data	DATE	NN
Info	VARCHAR(45)	NN
Materiale_Rotabile_ID	INT	NN

Tabella Posto		
Attributo	Tipo di dato	Attributi2
Num_posto	VARCHAR(3)	PK
Materiale_Rotabile_ID	INT	PK

Tabella Prenotazione		
Attributo	Tipo di dato	Attributi2
Codice_prenotazione	INT	PK
Posto_Num_posto	VARCHAR(3)	NN
Passeggero_CF	VARCHAR(16)	NN
Servizio_Ferroviario_ID	INT	NN
Stato	VARCHAR(45)	NN

Tabella Passeggero		
Attributo	Tipo di dato	Attributi2
CF	VARCHAR(16)	PK
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
Data_nascita	DATE	NN
Carta_di_credito	VARCHAR(16)	NN

Tabella Merce_trasportata		
Attributo	Tipo di dato	Attributi2
ID	INT	PK
Servizio_Ferroviario_ID	INT	NN
Materiale_Rotabile_ID	INT	NN

Massa	INT	NN
Descrizione	VARCHAR(120)	NN
Azienda_PIVA_Mittente	VARCHAR(11)	NN
Fattura_Numero_Fattura	VARCHAR(45)	NN

Tabella Fattura		
Attributo	Tipo di dato	Attributi2
Numero_Fattura	VARCHAR(45)	PK
Azienda_PIVA_Mittente	VARCHAR(11)	PK
Azienda_PIVA_Destinatario	VARCHAR(11)	NN
Data	DATE	NN
Importo	FLOAT	NN
IVA_applicata	INT	NN

Tabella Azienda		
Attributo	Tipo di dato	Attributi**2**
PIVA	VARCHAR(11)	PK
Nome	VARCHAR(45)	NN

Tabella Turno		
Attributo	Tipo di dato	Attributi**2**
ID	INT	PK
Data	DATE	NN
Ora_inizio	TIME	NN
Ora_fine	TIME	NN

Tabella Copertura		
Attributo	Tipo di dato	Attributi**2**
Lavoratore_CF	VARCHAR(16)	PK
Turno_ID	INT	PK

Tabella Malattia		
Attributo	Tipo di dato	Attributi**2**
Lavoratore_CF	VARCHAR(16)	PK
Turno_ID	INT	PK

Tabella Lavoratore		
Attributo	Tipo di dato	Attributi**2**
CF	VARCHAR(16)	PK
Mansione_ID	INT	NN
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
Data_nascita	DATE	NN
Luogo_Nascita	VARCHAR(45)	NN

Tabella Mansione		
Attributo	Tipo di dato	Attributi**2**
ID	INT	PK

Incarico	VARCHAR(45)	NN

Tabella Servizio_Ferroviario		
Attributo	Tipo di dato	Attributi2
ID	INT	PK
Tratta_ID	INT	NN
Treno_Matricola	INT	NN
Turno_ID	INT	NN
Data	DATE	NN

Tabella Tratta		
Attributo	Tipo di dato	Attributi2
ID	INT	PK
Capolinea_partenza	VARCHAR(45)	NN
Capolinea_arrivo	VARCHAR(45)	NN

Tabella Stazione		
Attributo	Tipo di dato	Attributi2
Nome_Stazione	VARCHAR(45)	PK
Citta_Nome	VARCHAR(45)	NN

Tabella Citta		
Attributo	Tipo di dato	Attributi2
Nome	VARCHAR(45)	PK
Provincia_sigla	VARCHAR(2)	PK

Tabella Provincia		
Attributo	Tipo di dato	Attributi2
Sigla	VARCHAR(2)	PK
Nome	VARCHAR(45)	NN

Tabella Fermata		
Attributo	Tipo di dato	Attributi2
Tratta_ID	INT	PK
Stazione_Nome_Stazione	VARCHAR(45)	PK
Stazione_Citta_Nome	VARCHAR(45)	NN
Sequenza	INT	NN
Orario_partenza	TIME	
Orario_arrivo	TIME	

Tabella login		
Attributo	Tipo di dato	Attributi2
Username	VARCHAR(45)	PK
Password	VARCHAR(2)	NN
Ruolo	ENUM(‘amministratore’,’lavoratore’,’controllore’,’manutentore’)	NN

Indici

Tabella Modello	
Indice Modello_marca	Tipo[3]:
fk_Modello_Marca1_idx	IDX

Tabella Materiale_Rotabile	
Indice Vagone	Tipo[3]:
fk_Vagone_Treno1_idx	IDX
fk_Materiale_Rotabile_Modello1_idx	IDX

Tabella Attivita_Manutenzione	
Indice Manutenzione	Tipo[3]:
fk_Attivita_Manutenzione_Materiale_Rotabile_idx	IDX

Tabella Posto	
Indice Posto	Tipo[3]:
fk_Posto_Materiale_Rotabile1_idx	IDX

Tabella Prenotazione	
Indice Prenotazione	Tipo[3]:
fk_Prenotazione_Posto1_idx	IDX
fk_Prenotazione_Passeggero1_idx	IDX
fk_Prenotazione_Servizio_Ferroviario1_idx	IDX

Tabella Merce_Trasportata	
Indice Merce	Tipo[3]:
fk_Merce_trasportata_Materiale_Rotabile1_idx	IDX
fk_Merce_trasportata_Servizio_Ferroviario1_idx	IDX
fk_Merce_trasportata_Azienda1_idx	IDX
fk_Merce_trasportata_Azienda2_idx	IDX

Tabella Copertura	
Indice Copertura	Tipo[3]:
fk_Copertura_Lavoratore1_idx	IDX
fk_Copertura_Turno1_idx	IDX

Tabella Malattia	
Indice Malattia	Tipo[3]:
fk_Malattia_Lavoratore1_idx	IDX
fk_Malattia_Turno1_idx	IDX

Tabella Lavoratore	
Indice Lavoratore	Tipo[3]:
fk_Lavoratore_Mansione1_idx	IDX

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

Tabella Servizio_Ferroviario	
Indice Servizio	Tipo3:
fk_Tratta_Turno1_idx	IDX
fk_Tratta_Treno1_idx	IDX
fk_Itinerario_Tratta1_idx	IDX

Tabella Fermata	
Indice Fermata	Tipo3:
fk_Fermata_Stazione1_idx	IDX

Tabella Stazione	
Indice Stazione	Tipo3:
fk_Citta_Citta1_idx	IDX

Tabella Citta	
Indice Citta	Tipo3:
fk_Citta_Provincia1_idx	IDX

Trigger

È stato implementato il trigger **Turno_BEFORE_INSERT** per controllare la correttezza dell'orario e per far sì che un turno non superi le 4 ore come da regola aziendale.

```
CREATE DEFINER = CURRENT_USER TRIGGER `AziendaFerroviaria`.`Turno_BEFORE_INSERT`
BEFORE INSERT ON `Turno` FOR EACH ROW
BEGIN

IF(new.Ora_inizio > new.Ora_fine) THEN
SIGNAL SQLSTATE '45000'
 SET MESSAGE_TEXT = 'Orari non validi';
ELSEIF(TIMEDIFF(new.Ora_fine, new.Ora_inizio) > "04:00:00") THEN
 SIGNAL SQLSTATE '45000'
 SET MESSAGE_TEXT = 'Il turno deve essere di un massimo di 4 ore';
END IF;
END
```

È stato implementato il trigger **Copertura_BEFORE_INSERT** per far sì che non vengano assegnati più di 5 turni alla settimana ad un lavoratore.

```
CREATE DEFINER = CURRENT_USER TRIGGER
`AziendaFerroviaria`.`Copertura_BEFORE_INSERT` BEFORE INSERT ON `Copertura` FOR
EACH ROW

BEGIN
DECLARE n INT;
DECLARE d DATE;

select data INTO d from Turno where ID=new.Turno_ID;

SELECT count(*) INTO n from Copertura, Turno where Turno_ID=ID and
Lavoratore_CF=new.Lavoratore_CF and week(d)=week(Data) and year(d) = year(Data);
```

```
    IF(n >= 5) THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Massimo numero di turni raggiunto';
    END IF;
END
```

Eventi

Non sono stati usati eventi nella applicazione.

Viste

Convoglio

Vista che mostra tutti i treni con la loro composizione.

```
CREATE VIEW Convoglio as SELECT Treno_Matricola, Modello_Marca_Nome, Modello_Nome,
Tipo, Portata_max, Num_max_passeggeri,Classe
FROM Materiale_Rotabile, Modello
where Modello_Nome=Nome and Modello_Marca_Nome=Marca_Nome
order by Treno_Matricola;
```

composizione_treno

Vista che mostra la composizione dei treni, con le varie specifiche dei vagoni e il loro numero.

```
CREATE VIEW `composizione_treno` AS
select treno_matricola, tipo, classe, count(*) N_Vagoni
from Convoglio
where (Classe in ('PRIMA','SECONDA') or Tipo='LOCOMOTORE' or Tipo='MERCI')
group by Treno_Matricola, Tipo, Classe
order by Treno_Matricola;
```

Stored Procedures e transazioni

aggiungi_azienda

Procedura che aggiunge un'azienda nella base di dati inserendone i dati nella tabella "Azienda".

```
CREATE PROCEDURE `aggiungi_azienda` (IN var_piva VARCHAR(11), IN var_nome
VARCHAR(45))
BEGIN
        INSERT INTO Azienda
        (`PIVA`,
        `Nome`)
        VALUES
        (var_piva,
        var_nome);
END
```

aggiungi_lavoratore

Procedura che aggiunge un lavoratore nella base di dati inserendone i dati nella tabella "Lavoratore" per i dettagli sul lavoratore e "login" per la gestione degli accessi.

```
CREATE PROCEDURE `aggiungi_lavoratore` (IN var_cf VARCHAR(16), IN var_mansione
INT, IN var_nome VARCHAR(45), IN var_cognome VARCHAR(45), IN var_data_nascita
DATE, IN var_luogo_nascita VARCHAR(45), IN var_username VARCHAR(45), IN var_pass
VARCHAR(45))
BEGIN
```

```
        start transaction;
        INSERT INTO login(`username`,`password`,`ruolo`)
        VALUES(var_username, var_pass, '2');

        INSERT INTO Lavoratore(`CF`, `Mansione_ID`, `Nome`, `Cognome`,
`Data_nascita`, `Luogo_nascita`,`login_username`)
        VALUES(var_cf, var_mansione, var_nome, var_cognome, var_data_nascita,
var_luogo_nascita, var_username);
        commit;
END
```

aggiungi_passeggero

Procedura che aggiunge un passeggero nella base di dati inserendone i dati nella tabella “Passeggero”.

```
CREATE PROCEDURE `aggiungi_passeggero` (IN var_cf VARCHAR(16), IN var_nome
VARCHAR(45), IN var_cognome VARCHAR(45), IN var_data_nascita DATE, IN var_carta
VARCHAR(16))
BEGIN
start transaction;
    INSERT INTO `AziendaFerroviaria`.`Passeggero`
    (`CF`,
    `Nome`,
    `Cognome`,
    `Data_nascita`,
    `Carta_di_credito`)
    VALUES
    (var_cf,
    var_nome,
    var_cognome,
    var_data_nascita,
    var_carta);
commit;
END
```

aggiungi_prenotazione

Procedura che aggiunge una prenotazione controllando la disponibilità del posto su quel servizio ferroviario e inserendo i dati dell’acquirente e del servizio prenotato nella tabella “Prenotazione” nel caso in cui sia libero.

```
CREATE PROCEDURE `aggiungi_prenotazione` (IN var_codice_prenotazione VARCHAR(5),
IN var_posto_num VARCHAR(4), IN var_passeggero_cf VARCHAR(16), IN var_servizio_id
INT)

BEGIN
DECLARE EXIT HANDLER FOR sqlexception
 BEGIN
 rollback;
 resignal;
 END;

SET TRANSACTION ISOLATION LEVEL serializable;
start transaction;

 if exists (Select * from Prenotazione where Posto_num_posto = var_posto_num and
Servizio_Ferroviario_ID = var_servizio_id) then
SIGNAL SQLSTATE '45000'
```

```
 SET MESSAGE_TEXT = 'Il posto selezionato è occupato.';
 end if;

    INSERT IGNORE INTO `AziendaFerroviaria`.`Prenotazione`
    (`Codice_prenotazione`,
    `Posto_Num_posto`,
    `Passeggero_CF`,
    `Stato`,
    `Servizio_Ferroviario_ID`)
    VALUES
    (var_codice_prenotazione,
    var_posto_num,
    var_passeggero_cf,
    'VALIDO',
    var_servizio_id);
commit;
END
```

aggiungi_servizio

Procedura che aggiunge un servizio nella base di dati inserendone i dati nella tabella “Servizio Ferroviario”.

```
CREATE PROCEDURE `aggiungi_servizio` (IN var_id INT, IN var_turno INT, IN
var_treno VARCHAR(4), IN var_tratta INT, IN var_data DATE)
BEGIN
DECLARE EXIT HANDLER FOR sqlexception
 BEGIN
 rollback;
 resignal;
 END;

SET TRANSACTION ISOLATION LEVEL repeatable read;
start transaction;
INSERT INTO `AziendaFerroviaria`.`Servizio_Ferroviario`
(`ID`,
`Turno_ID`,
`Treno_Matricola`,
`Tratta_ID`,
`Data`)
VALUES
(var_id,
var_turno,
var_treno,
var_tratta,
var_data);
commit;
END
```

aggiungi_spedizione

Procedura che aggiunge una spedizione nella base di dati inserendone i dati insieme alle informazioni del mittente e del destinatario nella tabella “Merce_Trasportata” solo dopo aver verificato che il vagone sul servizio ferroviario specificato sia di tipo merci e non sia già occupato.

```
CREATE PROCEDURE `aggiungi_spedizione` (IN var_merce_id INT, IN var_mittente
VARCHAR(11), IN var_servizio INT, IN var_vagone_id INT, IN var_massa INT, IN
var_descrizione VARCHAR(120), IN var_fattura VARCHAR(45))
BEGIN
```

```
DECLARE EXIT HANDLER FOR sqlexception
 BEGIN
 rollback;
 resignal;
 END;

SET TRANSACTION ISOLATION LEVEL serializable;
start transaction;
    start transaction;

if exists (Select * from Merce_trasportata where Materiale_Rotabile_ID =
var_vagone_id and
Servizio_Ferroviario_ID = var_servizio) then
SIGNAL SQLSTATE '45000'
 SET MESSAGE_TEXT = 'Il vagone selezionato è occupato.';
 end if;

if exists (Select * from Materiale_Rotabile,Modello where ID = var_vagone_id and
Modello_Nome = Nome and Tipo != 'MERCI') then
SIGNAL SQLSTATE '45000'
 SET MESSAGE_TEXT = 'Il tipo di vagone selezionato non è corretto.';
 end if;

 if exists (Select * from Materiale_Rotabile,Modello where ID = var_vagone_id and
Modello_Nome = Nome and Portata_max < var_massa) then
SIGNAL SQLSTATE '45000'
 SET MESSAGE_TEXT = 'Il limite di massa per il vagone è stato superato.';
 end if;

    INSERT IGNORE INTO `AziendaFerroviaria`.`Merce_trasportata`
    (`ID`,
    `Servizio_Ferroviario_ID`,
    `Materiale_Rotabile_ID`,
    `Massa`,
    `Descrizione`,
    `Fattura_Numero_Fattura`,
    `Fattura_Azienda_PIVA_Mittente`)
    VALUES
    (var_merce_id,
    var_servizio,
    var_vagone_id,
    var_massa,
    var_descrizione,
    var_fattura,
    var_mittente);

    commit;
END
```

aggiungi_turno

Procedura che aggiunge un turno nella base di dati inserendone i dati nella tabella “Turno”.

```
CREATE PROCEDURE `aggiungi_turno` (IN var_id INT, IN var_data DATE, IN var_inizio
TIME, IN var_fine TIME)

BEGIN
DECLARE EXIT HANDLER FOR sqlexception
 BEGIN
        rollback;
```

```
    resignal;
 END;

 start transaction;
 INSERT INTO `AziendaFerroviaria`.`Turno`
    (`ID`,
    `Data`,
    `Ora_inizio`,
    `Ora_fine`)
    VALUES
    (var_id,
    var_data,
    var_inizio,
    var_fine);

 commit;
END
```

assegna_turno

Procedura che assegna ad un lavoratore un turno controllandone prima la disponibilità verificando che per quel turno non sia in malattia, inserisce i dati nella tabella “Copertura”.

```
CREATE PROCEDURE `assegna_turno` (IN var_lavoratore VARCHAR(16), IN var_turno
INT)

BEGIN
DECLARE EXIT HANDLER FOR sqlexception
 BEGIN
rollback;
 resignal;

 END;
 set transaction isolation level serializable;
 start transaction;

 if exists (Select * from Malattia where Lavoratore_CF = var_lavoratore and
Turno_ID = var_turno) then
SIGNAL SQLSTATE '45000'
 SET MESSAGE_TEXT = 'Lavoratore in malattia';
 end if;

 INSERT IGNORE INTO Copertura VALUES (var_lavoratore, var_turno);

 commit;
END
```

composizione_treno

Procedura che restituisce da quale materiale rotabile è composto un treno riportandone il tipo, la classe (se disponibile) e la quantità, i dati vengono presi dalla vista “composizione_treno”.

```
CREATE PROCEDURE `composizione_treno` (IN var_treno INT)
BEGIN


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

```
start transaction;

    select treno_matricola Matricola, Tipo, Classe, N_Vagoni Quantita
    from composizione_treno
    where Treno_matricola = var_treno;

commit;
END
```

configurazione_treno

Procedura che restituisce la configurazione di un treno riportando nello specifico tutti i materiali da cui è composto con le loro specifiche.

```
CREATE PROCEDURE `configurazione_treno` (IN var_treno INT)
BEGIN

set transaction read only;
set transaction isolation level read committed;
start transaction;
    SELECT Treno_Matricola Matricola, Modello_Marca_Nome Marca, Modello_Nome
Modello, Tipo, Portata_max, Num_max_passeggeri Pass_max,Classe
    FROM Materiale_Rotabile, Modello
    where Modello_Nome=Nome and Modello_Marca_Nome=Marca_Nome and Treno_Matricola
= var_treno;
commit;
END
```

convalida_prenotazione

Procedura che permette di convalidare una prenotazione aggiornando lo stato di quest'ultima da "Valida" ad "Utilizzata".

```
CREATE PROCEDURE `convalida_prenotazione` (IN var_codice_prenotazione VARCHAR(5))
BEGIN
DECLARE EXIT HANDLER FOR sqlexception
BEGIN
 rollback;
 resignal;
 END;

set transaction isolation level serializable;
    UPDATE `AziendaFerroviaria`.`Prenotazione`
    SET
    `Stato` = 'Utilizzata'
    WHERE `Codice_prenotazione` = var_codice_prenotazione;
commit;
END
```

lavoratori_disponibili

Procedura che restituisce le informazioni di tutti i lavorato disponibili dati un giorno e gli orario di inizio e fine turno.

```
CREATE PROCEDURE `lavoratori_disponibili` (IN var_data DATE, IN var_inizio
TIME, IN var_fine TIME)
BEGIN
```

```
set transaction read only;
set transaction isolation level serializable;
 start transaction;

 select CF, Nome, Cognome
from Lavoratore
where CF not in (
select Lavoratore_CF
from Copertura
join Turno on (Turno_ID = ID)
where Data = var_data and ((var_inizio > Ora_inizio and var_inizio <
Ora_fine) or (var_fine > Ora_inizio and var_fine < Ora_fine)
or (var_inizio <= Ora_inizio and var_fine >= Ora_fine))
 union
select Lavoratore_CF
 from Malattia
 join Turno on (Turno_ID = ID)
 where Data = var_data and ((var_inizio > Ora_inizio and var_inizio <
Ora_fine) or (var_fine > Ora_inizio and var_fine < Ora_fine)
or (var_inizio <= Ora_inizio and var_fine >= Ora_fine)
 ));

 commit;
END
```

login

Procedura che permette l'accesso alla base di dati con il proprio ruolo assegnato.

```
CREATE PROCEDURE `login` (in var_username varchar(45), in var_pass varchar(45),
out var_role INT)
BEGIN
    declare var_user_role ENUM('amministratore', 'lavoratore',
'controllore','manutentore');

    select `ruolo` from `login`
        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 = 'lavoratore' then
            set var_role = 2;
        elseif var_user_role = 'controllore' then
            set var_role = 3;
        elseif var_user_role = 'manutentore' then
            set var_role = 4;
        else
            set var_role = 5;
        end if;
END
```

modifica_tratta_servizio

Procedura che permette di modificare la tratta di un particolare servizio ferroviario.

```
CREATE PROCEDURE `modifica_tratta_servizio` (IN var_servizio INT, IN var_tratta
INT)
BEGIN
DECLARE EXIT HANDLER FOR sqlexception
BEGIN
 rollback;
 resignal;
 END;

set transaction isolation level serializable;

start transaction;
UPDATE `AziendaFerroviaria`.`Servizio_Ferroviario`
set Tratta_ID = var_tratta
where ID = var_servizio;
commit;
END
```

modifica_treno_servizio

Procedura che permette di modificare il treno che esegue uno specifico servizio ferroviario.

```
CREATE PROCEDURE `modifica_treno_servizio` (IN var_servizio INT, IN var_treno
VARCHAR(4))
BEGIN
DECLARE EXIT HANDLER FOR sqlexception
 BEGIN
rollback;
 resignal;
 END;

set transaction isolation level serializable;

start transaction;
UPDATE `AziendaFerroviaria`.`Servizio_Ferroviario`
set Treno_Matricola = var_treno
where ID = var_servizio;
commit;
END
```

report_lavoratore

Procedura che permette di ottenere i turni con le relative informazioni per uno specifico lavoratore.

```
CREATE PROCEDURE `report_lavoratore` (IN var_lavoratore VARCHAR(16))
BEGIN
set transaction read only;
set transaction isolation level read committed;
 start transaction;
SELECT Lavoratore_CF CF, Turno.Data, Ora_inizio, Ora_fine,
Servizio_Ferroviario.ID Codice_Servizio FROM
Copertura,Turno,Servizio_Ferroviario,Tratta
where Lavoratore_CF=var_lavoratore and Copertura.Turno_ID=Turno.ID and
Servizio_Ferroviario.Turno_ID=Turno.ID and Turno.Data >= current_date
and Servizio_Ferroviario.Tratta_ID=Tratta.ID;
 commit;
END
```

report_manutentore

Procedura che permette di creare un report di manutenzione andando ad inserire i relativi dati all'interno della tabella report_manutenzione.

```
CREATE PROCEDURE `report_manutenzione` (IN var_id INT, IN var_data DATE, IN
var_info LONGTEXT, IN var_materiale_rotabile_id INT)
BEGIN

set transaction isolation level read committed;
start transaction;
    INSERT INTO `AziendaFerroviaria`.`Attivita_Manutenzione`
    (`Data`,
    `Info`,
    `Materiale_Rotabile_ID`)
    VALUES
    (var_data,
    var_info,
    var_materiale_rotabile_id);

    set var_id = last_insert_id();
commit;
END
```

richiesta_malattia

Procedura che permette di inserire uno specifico lavoratore in malattia rimuovendolo dal turno per cui si pone in malattia e inserendolo nella tabella "Malattia" in modo che non venga riassegnato per quel turno.

```
CREATE PROCEDURE `richiesta_malattia` (IN var_lavoratore VARCHAR(16), IN
var_turno VARCHAR(10))
BEGIN
set transaction isolation level repeatable read;
 start transaction;

insert into Malattia
 select Lavoratore_CF, Turno_ID
 from Copertura
 where Lavoratore_CF = var_lavoratore and Turno_ID=
var_turno;

 delete from Copertura
where Lavoratore_CF = var_lavoratore and Turno_ID = var_turno;

 commit;
END
```

verifica_prenotazione

Procedura che verifica lo stato di una prenotazione.

```
CREATE PROCEDURE `verifica_prenotazione` (IN var_codice_prenotazione VARCHAR(5))

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

```
    SELECT Stato FROM Prenotazione
    WHERE Codice_prenotazione = var_codice_prenotazione;

    commit;
END
```

fermate_tratta

Procedura che restituisce tutte le fermate con i relativi orari di partenza ed arrivo di una tratta.

```
CREATE PROCEDURE `fermate_tratta` (IN var_tratta INT)
BEGIN
set transaction read only;
set transaction isolation level serializable;
 start transaction;

 SELECT Stazione_Nome_Stazione Stazione,Orario_Partenza Partenza, Orario_Arrivo
Arrivo from Fermata where Tratta_ID = var_tratta order by Sequenza;
commit;
END
```

tratte_giornaliere

Procedura che restituisce tutte le tratte con i relativi capolinea di partenza ed arrivo.

```
CREATE PROCEDURE `tratte_giornaliere` ()
BEGIN
set transaction read only;
set transaction isolation level serializable;
 start transaction;

 SELECT Tratta_ID Tratta, Capolinea_partenza Da, Capolinea_arrivo A,
Orario_Partenza Delle  FROM Fermata,Tratta where Tratta_ID=ID and Sequenza=1
order by Orario_Partenza;
commit;
END
```

6. Appendice: Implementazione

Codice SQL per istanziare 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,ERR
OR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema AziendaFerroviaria
-- -----------------------------------------------------

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

-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Treno`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Treno` (
  `Matricola` INT NOT NULL,
  `Data_Acquisto` DATE NOT NULL,
  PRIMARY KEY (`Matricola`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Turno`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Turno` (
  `ID` INT NOT NULL,
  `Data` DATE NOT NULL,
  `Ora_inizio` TIME NOT NULL,
  `Ora_fine` TIME NOT NULL,
  PRIMARY KEY (`ID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Tratta`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Tratta` (
  `ID` INT NOT NULL,
  `Capolinea_partenza` VARCHAR(45) NOT NULL,
  `Capolinea_arrivo` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Servizio_Ferroviario`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Servizio_Ferroviario` (
  `ID` INT NOT NULL,
  `Turno_ID` INT NOT NULL,
```

```
  `Treno_Matricola` INT NOT NULL,
  `Tratta_ID` INT NOT NULL,
  `Data` DATE NOT NULL,
  PRIMARY KEY (`ID`),
  INDEX `fk_Tratta_Turno1_idx` (`Turno_ID` ASC) VISIBLE,
  INDEX `fk_Tratta_Treno1_idx` (`Treno_Matricola` ASC) VISIBLE,
  INDEX `fk_Itinerario_Tratta1_idx` (`Tratta_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Tratta_Turno1`
    FOREIGN KEY (`Turno_ID`)
    REFERENCES `AziendaFerroviaria`.`Turno` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Tratta_Treno1`
    FOREIGN KEY (`Treno_Matricola`)
    REFERENCES `AziendaFerroviaria`.`Treno` (`Matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Itinerario_Tratta1`
    FOREIGN KEY (`Tratta_ID`)
    REFERENCES `AziendaFerroviaria`.`Tratta` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Provincia`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Provincia` (
  `Sigla` VARCHAR(2) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Sigla`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Citta`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Citta` (
  `Nome` VARCHAR(45) NOT NULL,
  `Provincia_Sigla` VARCHAR(2) NOT NULL,
  PRIMARY KEY (`Nome`, `Provincia_Sigla`),
  INDEX `fk_Citta_Provincia1_idx` (`Provincia_Sigla` ASC) VISIBLE,
  CONSTRAINT `fk_Citta_Provincia1`
    FOREIGN KEY (`Provincia_Sigla`)
    REFERENCES `AziendaFerroviaria`.`Provincia` (`Sigla`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Stazione`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Stazione` (
  `Nome_Stazione` VARCHAR(45) NOT NULL,
  `Citta_Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Nome_Stazione`),
  INDEX `fk_Stazione_Citta1_idx` (`Citta_Nome` ASC) VISIBLE,
  CONSTRAINT `fk_Stazione_Citta1`
    FOREIGN KEY (`Citta_Nome`)
```

```
    REFERENCES `AziendaFerroviaria`.`Citta` (`Nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Marca`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Marca` (
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Nome`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Modello`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Modello` (
  `Nome` VARCHAR(45) NOT NULL,
  `Marca_Nome` VARCHAR(45) NOT NULL,
  `Tipo` VARCHAR(45) NOT NULL,
  `Portata_max` INT NULL DEFAULT 0,
  `Num_max_passeggeri` INT NULL DEFAULT 0,
  `Classe` VARCHAR(45) NULL DEFAULT '-',
  PRIMARY KEY (`Marca_Nome`, `Nome`),
  INDEX `fk_Modello_Marca1_idx` (`Marca_Nome` ASC) VISIBLE,
  CONSTRAINT `fk_Modello_Marca1`
    FOREIGN KEY (`Marca_Nome`)
    REFERENCES `AziendaFerroviaria`.`Marca` (`Nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Materiale_Rotabile`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Materiale_Rotabile` (
  `ID` INT NOT NULL,
  `Treno_Matricola` INT NOT NULL,
  `Modello_Marca_Nome` VARCHAR(45) NOT NULL,
  `Modello_Nome` VARCHAR(45) NOT NULL,
  INDEX `fk_Vagone_Treno1_idx` (`Treno_Matricola` ASC) VISIBLE,
  PRIMARY KEY (`ID`, `Treno_Matricola`, `Modello_Marca_Nome`, `Modello_Nome`),
  INDEX `fk_Materiale_Rotabile_Modello1_idx` (`Modello_Marca_Nome` ASC,
`Modello_Nome` ASC) VISIBLE,
  CONSTRAINT `fk_Vagone_Treno1`
    FOREIGN KEY (`Treno_Matricola`)
    REFERENCES `AziendaFerroviaria`.`Treno` (`Matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Materiale_Rotabile_Modello1`
    FOREIGN KEY (`Modello_Marca_Nome` , `Modello_Nome`)
    REFERENCES `AziendaFerroviaria`.`Modello` (`Marca_Nome` , `Nome`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


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

```
-- Table `AziendaFerroviaria`.`Attivita_Manutenzione`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Attivita_Manutenzione` (
  `ID` INT NOT NULL AUTO_INCREMENT,
  `Data` DATE NOT NULL,
  `Info` VARCHAR(500) NOT NULL,
  `Materiale_Rotabile_ID` INT NOT NULL,
  PRIMARY KEY (`ID`),
  INDEX `fk_Attivita_Manutenzione_Materiale_Rotabile1_idx`
(`Materiale_Rotabile_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Attivita_Manutenzione_Materiale_Rotabile1`
    FOREIGN KEY (`Materiale_Rotabile_ID`)
    REFERENCES `AziendaFerroviaria`.`Materiale_Rotabile` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Mansione`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Mansione` (
  `ID` INT NOT NULL,
  `Incarico` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`ID`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`login`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`login` (
  `username` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `ruolo` ENUM('amministratore', 'lavoratore', 'controllore', 'manutentore') NOT
NULL,
  PRIMARY KEY (`username`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Lavoratore`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Lavoratore` (
  `CF` VARCHAR(16) NOT NULL,
  `Mansione_ID` INT NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `Data_Nascita` DATE NOT NULL,
  `Luogo_Nascita` VARCHAR(45) NOT NULL,
  `login_username` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`CF`),
  INDEX `fk_Lavoratore_Mansione1_idx` (`Mansione_ID` ASC) VISIBLE,
  INDEX `fk_Lavoratore_login1_idx` (`login_username` ASC) VISIBLE,
  CONSTRAINT `fk_Lavoratore_Mansione1`
    FOREIGN KEY (`Mansione_ID`)
    REFERENCES `AziendaFerroviaria`.`Mansione` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Lavoratore_login1`
    FOREIGN KEY (`login_username`)
```

```
    REFERENCES `AziendaFerroviaria`.`login` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Copertura`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Copertura` (
  `Lavoratore_CF` VARCHAR(16) NOT NULL,
  `Turno_ID` INT NOT NULL,
  PRIMARY KEY (`Lavoratore_CF`, `Turno_ID`),
  INDEX `fk_Copertura_Lavoratore1_idx` (`Lavoratore_CF` ASC) VISIBLE,
  INDEX `fk_Copertura_Turno1_idx` (`Turno_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Copertura_Lavoratore1`
    FOREIGN KEY (`Lavoratore_CF`)
    REFERENCES `AziendaFerroviaria`.`Lavoratore` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Copertura_Turno1`
    FOREIGN KEY (`Turno_ID`)
    REFERENCES `AziendaFerroviaria`.`Turno` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Malattia`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Malattia` (
  `Lavoratore_CF` VARCHAR(16) NOT NULL,
  `Turno_ID` INT NOT NULL,
  PRIMARY KEY (`Lavoratore_CF`, `Turno_ID`),
  INDEX `fk_Malattia_Lavoratore1_idx` (`Lavoratore_CF` ASC) VISIBLE,
  INDEX `fk_Malattia_Turno1_idx` (`Turno_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Malattia_Lavoratore1`
    FOREIGN KEY (`Lavoratore_CF`)
    REFERENCES `AziendaFerroviaria`.`Lavoratore` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Malattia_Turno1`
    FOREIGN KEY (`Turno_ID`)
    REFERENCES `AziendaFerroviaria`.`Turno` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Azienda`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Azienda` (
  `PIVA` VARCHAR(11) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`PIVA`))
ENGINE = InnoDB;


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

```
-- Table `AziendaFerroviaria`.`Fattura`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Fattura` (
  `Numero_Fattura` VARCHAR(45) NOT NULL,
  `Azienda_PIVA_Mittente` VARCHAR(11) NOT NULL,
  `Azienda_PIVA_Destinatario` VARCHAR(11) NOT NULL,
  `Data` DATE NOT NULL,
  `Importo` FLOAT NOT NULL,
  `IVA_applicata` INT NOT NULL,
  INDEX `fk_Fattura_Azienda1_idx` (`Azienda_PIVA_Mittente` ASC) VISIBLE,
  INDEX `fk_Fattura_Azienda2_idx` (`Azienda_PIVA_Destinatario` ASC) VISIBLE,
  PRIMARY KEY (`Numero_Fattura`, `Azienda_PIVA_Mittente`),
  CONSTRAINT `fk_Fattura_Azienda1`
    FOREIGN KEY (`Azienda_PIVA_Mittente`)
    REFERENCES `AziendaFerroviaria`.`Azienda` (`PIVA`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Fattura_Azienda2`
    FOREIGN KEY (`Azienda_PIVA_Destinatario`)
    REFERENCES `AziendaFerroviaria`.`Azienda` (`PIVA`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Merce_trasportata`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Merce_trasportata` (
  `ID` INT NOT NULL,
  `Materiale_Rotabile_ID` INT NOT NULL,
  `Massa` INT NOT NULL,
  `Descrizione` VARCHAR(45) NOT NULL,
  `Servizio_Ferroviario_ID` INT NOT NULL,
  `Fattura_Numero_Fattura` VARCHAR(45) NOT NULL,
  `Fattura_Azienda_PIVA_Mittente` VARCHAR(11) NOT NULL,
  PRIMARY KEY (`ID`),
  INDEX `fk_Merce_trasportata_Materiale_Rotabile1_idx` (`Materiale_Rotabile_ID`
ASC) VISIBLE,
  INDEX `fk_Merce_trasportata_Servizio_Ferroviario1_idx`
(`Servizio_Ferroviario_ID` ASC) VISIBLE,
  INDEX `fk_Merce_trasportata_Fattura1_idx` (`Fattura_Numero_Fattura` ASC,
`Fattura_Azienda_PIVA_Mittente` ASC) VISIBLE,
  CONSTRAINT `fk_Merce_trasportata_Materiale_Rotabile1`
    FOREIGN KEY (`Materiale_Rotabile_ID`)
    REFERENCES `AziendaFerroviaria`.`Materiale_Rotabile` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Merce_trasportata_Servizio_Ferroviario1`
    FOREIGN KEY (`Servizio_Ferroviario_ID`)
    REFERENCES `AziendaFerroviaria`.`Servizio_Ferroviario` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Merce_trasportata_Fattura1`
    FOREIGN KEY (`Fattura_Numero_Fattura` , `Fattura_Azienda_PIVA_Mittente`)
    REFERENCES `AziendaFerroviaria`.`Fattura` (`Numero_Fattura` ,
`Azienda_PIVA_Mittente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Posto`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Posto` (
  `Num_posto` VARCHAR(3) NOT NULL,
  `Materiale_Rotabile_ID` INT NOT NULL,
  PRIMARY KEY (`Num_posto`, `Materiale_Rotabile_ID`),
  INDEX `fk_Posto_Materiale_Rotabile1_idx` (`Materiale_Rotabile_ID` ASC) VISIBLE,
  CONSTRAINT `fk_Posto_Materiale_Rotabile1`
    FOREIGN KEY (`Materiale_Rotabile_ID`)
    REFERENCES `AziendaFerroviaria`.`Materiale_Rotabile` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Passeggero`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Passeggero` (
  `CF` VARCHAR(16) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `Data_nascita` DATE NOT NULL,
  `Carta_di_credito` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Prenotazione`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Prenotazione` (
  `Codice_prenotazione` VARCHAR(5) NOT NULL,
  `Posto_Num_posto` VARCHAR(3) NOT NULL,
  `Passeggero_CF` VARCHAR(16) NOT NULL,
  `Stato` VARCHAR(45) NOT NULL,
  `Servizio_Ferroviario_ID` INT NOT NULL,
  PRIMARY KEY (`Codice_prenotazione`),
  INDEX `fk_Prenotazione_Posto1_idx` (`Posto_Num_posto` ASC) VISIBLE,
  INDEX `fk_Prenotazione_Passeggero1_idx` (`Passeggero_CF` ASC) VISIBLE,
  INDEX `fk_Prenotazione_Servizio_Ferroviario1_idx` (`Servizio_Ferroviario_ID`
ASC) VISIBLE,
  CONSTRAINT `fk_Prenotazione_Posto1`
    FOREIGN KEY (`Posto_Num_posto`)
    REFERENCES `AziendaFerroviaria`.`Posto` (`Num_posto`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Prenotazione_Passeggero1`
    FOREIGN KEY (`Passeggero_CF`)
    REFERENCES `AziendaFerroviaria`.`Passeggero` (`CF`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Prenotazione_Servizio_Ferroviario1`
    FOREIGN KEY (`Servizio_Ferroviario_ID`)
    REFERENCES `AziendaFerroviaria`.`Servizio_Ferroviario` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `AziendaFerroviaria`.`Fermata`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `AziendaFerroviaria`.`Fermata` (
  `Tratta_ID` INT NOT NULL,
  `Stazione_Nome_Stazione` VARCHAR(45) NOT NULL,
  `Orario_Partenza` TIME NULL,
  `Orario_Arrivo` TIME NULL,
  `Sequenza` INT NOT NULL,
  PRIMARY KEY (`Tratta_ID`, `Stazione_Nome_Stazione`),
  INDEX `fk_Fermata_Stazione1_idx` (`Stazione_Nome_Stazione` ASC) INVISIBLE,
  CONSTRAINT `fk_Fermata_Tratta1`
    FOREIGN KEY (`Tratta_ID`)
    REFERENCES `AziendaFerroviaria`.`Tratta` (`ID`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Fermata_Stazione1`
    FOREIGN KEY (`Stazione_Nome_Stazione`)
    REFERENCES `AziendaFerroviaria`.`Stazione` (`Nome_Stazione`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE USER 'amministratore' IDENTIFIED BY 'amministratore';

GRANT EXECUTE ON procedure `AziendaFerroviaria`.`aggiungi_lavoratore` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`aggiungi_azienda` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`aggiungi_passeggero` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`aggiungi_prenotazione` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`aggiungi_spedizione` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`aggiungi_turno` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`lavoratori_disponibili` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`richiesta_malattia` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`assegna_turno` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`configurazione_treno` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`composizione_treno` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`modifica_treno_servizio` TO
'amministratore';
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`modifica_tratta_servizio` TO
'amministratore';
CREATE USER 'lavoratore' IDENTIFIED BY 'lavoratore';

GRANT EXECUTE ON procedure `AziendaFerroviaria`.`report_lavoratore` TO
'lavoratore';
CREATE USER 'controllore' IDENTIFIED BY 'controllore';

GRANT EXECUTE ON procedure `AziendaFerroviaria`.`verifica_prenotazione` TO
'controllore';
```

```
GRANT EXECUTE ON procedure `AziendaFerroviaria`.`convalida_prenotazione` TO
'controllore';
CREATE USER 'manutentore' IDENTIFIED BY 'manutentore';

GRANT EXECUTE ON procedure `AziendaFerroviaria`.`report_manutenzione` TO
'manutentore';

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

Codice del Front-End

main.c

Permette ad un utente di collegarsi al database e di effettuare l'accesso per ottenere i privilegi assegnatoli dal proprio ruolo.

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

#include "defines.h"

#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"
#define ANSI_COLOR_BLUE    "\x1b[34m"
#define ANSI_COLOR_MAGENTA "\x1b[35m"
#define ANSI_COLOR_CYAN    "\x1b[36m"
#define ANSI_COLOR_RESET   "\x1b[0m"

static MYSQL *conn;

typedef enum {
    ADMIN = 1,
    LAVORATORE,
    CONTROLLORE,
    MANUTENTORE,
    FAILED_LOGIN
} role_t;

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){
    char username[128];
    char password[128];

    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, "localhost", "login",
"Lineagialla1!","AziendaFerroviaria", 3306, NULL, CLIENT_MULTI_STATEMENTS |
CLIENT_MULTI_RESULTS) == NULL) {
```

```
        fprintf (stderr, "mysql_real_connect() failed\n");
        mysql_close (conn);
        exit(1);
    }
    printf(ANSI_COLOR_CYAN  "\n*****************************\n");
    printf(  "*** Azienda Ferroviaria DB ***\n");
    printf(  "*****************************" ANSI_COLOR_RESET "\n");
    printf(ANSI_COLOR_GREEN  "Benvenuto, inserisci i tuoi dati per
accedere:\n\n"ANSI_COLOR_RESET);

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


    int role = attempt_login(conn, username, password);

    switch(role) {
        case ADMIN:
            admin(conn, username);
            break;
        case LAVORATORE:
            lavoratore(conn, username);
            break;
        case CONTROLLORE:
            controllore(conn, username);
            break;
        case MANUTENTORE:
            manutentore(conn, username);
            break;
        default:
            printf(ANSI_COLOR_RED"Dati non corretti!\n"ANSI_COLOR_RESET);
            abort();
    }

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

admin.c
Utilizzato per l'accesso come amministratore e per far accedere all'utente a tutte le procedure del suo ruolo.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <errno.h>

#include "defines.h"

#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"
#define ANSI_COLOR_BLUE    "\x1b[34m"
#define ANSI_COLOR_MAGENTA "\x1b[35m"
#define ANSI_COLOR_CYAN    "\x1b[36m"
#define ANSI_COLOR_RESET   "\x1b[0m"
```

```
char command[20];

static void aggiungi_lavoratore(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[8];

    char cf[16];
    int mansione_id;
        char name[45];
        char surname[45];
    MYSQL_TIME *date;   //Date of birth
    char pob[45];   //Place of birth
    date = malloc(sizeof(MYSQL_TIME));
        char username[45];
    char password[45];
    char passwordtmp[45];

    printf("Aggiungi lavoratore\n");

    // Get the required information
    printf("CF: ");
    if(getInput(16, cf, false) < 0) {
        return;
    }
    printf("ID Mansione: ");
    if(getInteger(&mansione_id) < 0){
        return;
    }

    printf("Nome: ");
    if(getInput(45, name, false) < 0) {
        return;
    }

    printf("Cognome: ");
    if(getInput(45, surname, false) < 0) {
        return;
    }

    printf("Luogo di nascita: ");
    if(getInput(45, pob, false) < 0) {
        return;
    }

    printf("Inserisci data di nascita(formato dd/mm/yyyy): ");
    if(getDate(date) < 0){
        return;
    }

    printf("Username: ");
    if(getInput(45, username, false) < 0) {
        return;
    }

    printf("Password: ");
    if(getInput(45, password, true) < 0) {
        return;
    }
```

```
    printf("Ripeti password: ");
    if(getInput(45, passwordtmp, true) < 0) {
        return;
    }

    if(strcmp(password, passwordtmp)) {
        printf("Le password non corrispondono\n");
        return;
    }

    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_lavoratore(?, ?, ?, ?,
?, ?, ?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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

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

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

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

    param[4].buffer_type = MYSQL_TYPE_DATE;
    param[4].buffer = date;
    param[4].buffer_length = sizeof(*date);

    param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[5].buffer = pob;
    param[5].buffer_length = strlen(pob);

    param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[6].buffer = username;
    param[6].buffer_length = strlen(username);

    param[7].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[7].buffer = password;
    param[7].buffer_length = strlen(password);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nell'aggiunta del lavoratore.");
    } else {
```

```
        printf("Lavoratore %s aggiunto correttamente\n", username);
    }

    free(date);

    mysql_stmt_close(prepared_stmt);
}


static void aggiungi_turno(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[4];

    int id;
    MYSQL_TIME *date;
    date = malloc(sizeof(MYSQL_TIME));
    MYSQL_TIME *start;
    MYSQL_TIME *end;

        start = malloc(sizeof(MYSQL_TIME));
        end = malloc(sizeof(MYSQL_TIME));

        printf("Aggiungi turno\n");

        // Get the required information
        printf("\nID Turno: ");
        if(getInteger(&id) < 0) {
            return;
        }
        printf("Inserisci data(formato dd/mm/yyyy): ");
        if(getDate(date) < 0){
            return;
        }

        printf("Inserisci orario di inizio(formato hh:mm): ");
        if(getTime(start) < 0) {
            return;
        }

        printf("Inserisci orario di fine(formato hh:mm): ");
        if(getTime(end) < 0) {
            return;
        }

    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_turno(?, ?, ?, ?)",
conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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

    param[1].buffer_type = MYSQL_TYPE_DATE;
    param[1].buffer = date;
    param[1].buffer_length = sizeof(*date);
```

```
        param[2].buffer_type = MYSQL_TYPE_TIME;
    param[2].buffer = start;
    param[2].buffer_length = sizeof(*start);

    param[3].buffer_type = MYSQL_TYPE_TIME;
    param[3].buffer = end;
    param[3].buffer_length = sizeof(*end);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nell'aggiunta del turno.");
    } else {
        printf("Turno %d aggiunto correttamente\n", id);
    }

    free(start);
    free(end);
    free(date);

    mysql_stmt_close(prepared_stmt);
}

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


    int turno;
    char cf[17];

    printf("Assegna turno\n");

    // Get the required information
    printf("CF Lavoratore: ");
    if(getInput(17, cf, false) < 0){
        return;
    }
    printf("\nID Turno: ");
    if(getInteger(&turno) < 0){
        return;
    }

    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call assegna_turno(?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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

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



    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nella assegnazione del turno.");
    } else {
        printf("Lavoratore %s assegnato al turno %d.\n", cf, turno);
    }

    mysql_stmt_close(prepared_stmt);
}

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

    char nome[45];
    char piva[12];


    printf("Aggiungi Azienda\n");

    // Get the required information
    printf("\nPartita IVA: ");
    if(getInput(12, piva, false) < 0){
        return;
    }

    printf("Nome azienda: ");
    if(getInput(45, nome, false) < 0){
        return;
    }


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_azienda(?, ?)", conn))
{
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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

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

```
    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nell'aggiunta dell'azienda.");
    } else {
        printf("Azienda %s aggiunta correttamente.\n", nome);
    }

    mysql_stmt_close(prepared_stmt);
}

static void aggiungi_passeggero(MYSQL *conn) {
        MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[5];

    char cf[16];
    char carta[16];
        char name[45];
        char surname[45];
    MYSQL_TIME *date;//Date of birth
    date = malloc(sizeof(MYSQL_TIME));


    printf("Aggiungi Passeggero\n");

    // Get the required information
    printf("CF: ");
    if(getInput(16, cf, false) < 0) {
        return;
    }

        printf("Nome: ");
    if(getInput(45, name, false) < 0) {
        return;
    }

    printf("Cognome: ");
    if(getInput(45, surname, false) < 0) {
        return;
    }

    printf("Data di nascita(formato dd/mm/yyyy): ");
    if(getDate(date) < 0){
        return;
    }

    printf("Carta di credito: ");
    if(getInput(16, carta, false) < 0) {
        return;
    }

    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_passeggero(?, ?, ?, ?,
?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
```

```
    }

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


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

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

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

    param[3].buffer_type = MYSQL_TYPE_DATE;
    param[3].buffer = date;
    param[3].buffer_length = sizeof(*date);

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

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nell'aggiunta del passeggero.");
    } else {
        printf("Passeggero %s aggiunto correttamente\n", cf);
    }

    free(date);

    mysql_stmt_close(prepared_stmt);

}

static void aggiungi_prenotazione(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[4];

    char codice[6];
    char cf[17];
        char posto[4];
        int servizio;


    printf("Aggiungi Prenotazione\n");

    // Get the required information
    printf("Codice prenotazione (5 cifre): ");
    if(getInput(6, codice, false) < 0) {
        return;
    }
```

```
    printf("Numero posto (3 cifre): ");
    if(getInput(4, posto, false) < 0) {
        return;
    }

        printf("Codice fiscale passeggero: ");
    if(getInput(17, cf, false) < 0) {
        return;
    }


    printf("ID Servizio Ferroviario: ");
    if(getInteger(&servizio) < 0){
        return;
    }


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_prenotazione(?, ?, ?,
?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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


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

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

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

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

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nell'aggiunta della
prenotazione.");
    } else {
        printf("Prenotazione %s per il passeggero %s aggiunta correttamente\n",
codice, cf);
    }


    mysql_stmt_close(prepared_stmt);
```

```
}

static void aggiungi_servizio(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[5];

        char treno[4];
        int servizio;
        int turno;
        int tratta;
    MYSQL_TIME *date;
    date = malloc(sizeof(MYSQL_TIME));


    printf("Aggiungi Servizio\n");

    // Get the required informationù
    printf("ID Servizio Ferroviario: ");
    if(getInteger(&servizio) < 0){
        return;
    }
    printf("Matricola treno (4 cifre): ");
    if(getInput(5, treno, false) < 0) {
        return;
    }

    printf("ID Tratta: ");
    if(getInteger(&tratta) < 0) {
        return;
    }

        printf("ID Turno: ");
    if(getInteger(&turno) < 0) {
        return;
    }

    printf("Data (formato dd/mm/yyyy): ");
    if(getDate(date) < 0){
        return;
    }



    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_servizio(?, ?, ?, ?,
?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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

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

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

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

    param[4].buffer_type = MYSQL_TYPE_DATE;
    param[4].buffer = date;
    param[4].buffer_length = sizeof(*date);


    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nell'aggiunta del servizio
ferroviario.");
    } else {
        printf("Servizio ferroviario %d sulla tratta %d aggiunto
correttamente\n", servizio, tratta);
    }

    free(date);
    mysql_stmt_close(prepared_stmt);
}

static void aggiungi_spedizione(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[7];

    char mittente[11];
    char fattura[45];
    char descrizione[120];
    int massa;
    int servizio;
        int vagone;
        int id;


    printf("Aggiungi Spedizione\n");

    // Get the required information
    printf("Codice spedizione: ");
    if(getInteger(&id) < 0) {
        return;
    }

    printf("Partita IVA mittente: ");
    if(getInput(11, mittente, false) < 0) {
        return;
    }

        printf("Numero fattura: ");
    if(getInput(45, fattura, false) < 0) {
        return;
    }
```

```
    printf("ID Servizio Ferroviario: ");
    if(getInteger(&servizio) < 0){
        return;
    }

    printf("ID Vagone: ");
    if(getInteger(&vagone) < 0){
        return;
    }

    printf("Descrizione contenuto: ");
    if(getInput(120, descrizione, false) < 0) {
        return;
    }

    printf("Massa: ");
    if(getInteger(&massa) < 0){
        return;
    }

    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call aggiungi_spedizione(?, ?, ?, ?,
?, ?, ?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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


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

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

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

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

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

    param[5].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[5].buffer = descrizione;
    param[5].buffer_length = strlen(descrizione);

    param[6].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[6].buffer = fattura;
    param[6].buffer_length = strlen(fattura);
```

```
    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nell'aggiunta della
spedizione.");
    } else {
        printf("Spedizione %d aggiunta correttamente\n", id);
    }


    mysql_stmt_close(prepared_stmt);
}

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

        char cf[16];
    int turno;


    printf("\nRichiesta malattia\n");

    // Get the required information
    printf("\nCF Lavoratore: ");
    if(getInput(16, cf, false) < 0) {
        return;
    }

    printf("ID Turno: ");
    if(getInteger(&turno) < 0) {
        return;
    }

    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call richiesta_malattia(?, ?)",
conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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

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

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }
```

```
    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nella richiesta di malattia.");
    } else {
        printf("Lavoratore %s correttamente inserito in malattia.\n", cf);
    }

    mysql_stmt_close(prepared_stmt);
}


static void lavoratori_disponibili(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[3];

        MYSQL_TIME *date;
    MYSQL_TIME *start;
    MYSQL_TIME *end;
    int status;

    date = malloc(sizeof(MYSQL_TIME));
        start = malloc(sizeof(MYSQL_TIME));
        end = malloc(sizeof(MYSQL_TIME));

    printf("\n");
    printf("Ricerca lavoratori disponibili\n");

    // Get the required information
        printf("Inserisci data turno(formato dd/mm/yyyy): ");
    if(getDate(date) < 0) return;

    printf("Inserisci orario di inizio(formato hh:mm): ");
    if(getTime(start) < 0) return;

        printf("Inserisci orario di fine(formato hh:mm): ");
        if(getTime(end) < 0) return;
        getchar();

    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call lavoratori_disponibili(?, ?,
?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

    param[0].buffer_type = MYSQL_TYPE_DATE;
    param[0].buffer = date;
    param[0].buffer_length = sizeof(*date);

    param[1].buffer_type = MYSQL_TYPE_TIME;
    param[1].buffer = start;
    param[1].buffer_length = sizeof(*start);

    param[2].buffer_type = MYSQL_TYPE_TIME;
    param[2].buffer = end;
    param[2].buffer_length = sizeof(*end);
```

```
    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nella ricerca di lavoratori
disponibili.");
    }

      char buff[100];
      sprintf(buff, "Lavoratori disponibili il giorno %d/%d/%d %u:%u-%u:%u",
date->day, date->month, date->year, start->hour, start->minute, end->hour, end-
>minute);


    do {
        dump_result_set(conn, prepared_stmt, buff);
        status = mysql_stmt_next_result(prepared_stmt);
        if (status > 0)
            finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition",
true);
    } while (status == 0);

        free(start);
    free(end);
    free(date);

    mysql_stmt_close(prepared_stmt);
}

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

    int servizio;
    int tratta;


    printf("\nModifica associazione Servizio a Tratta\n");

    // Get the required information
    printf("ID Servizio Ferroviario: ");
    if(getInteger(&servizio) < 0){
        return;
    }

    printf("ID Tratta: ");
    if(getInteger(&tratta) < 0){
        return;
    }

    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call modifica_tratta_servizio(?,?)",
conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }
```

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

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

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


    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nella modifica associazione
Servizio a Tratta.");
    }else {
        printf("Modifica effettuata con successo.\n");
    }

    mysql_stmt_close(prepared_stmt);
}

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

    int servizio;
    char treno[4];


    printf("\nModifica associazione Treno a Servizio\n");

    // Get the required information
    printf("ID Servizio Ferroviario: ");
    if(getInteger(&servizio) < 0){
        return;
    }

    printf("Matricola treno (4 cifre): ");
    if(getInput(5, treno, false) < 0) {
        return;
    }


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call modifica_treno_servizio(?,?)",
conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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

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

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


    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nella modifica associazione
treno a servizio.");
    }else {
        printf("Modifica effettuata con successo.\n");
    }


    mysql_stmt_close(prepared_stmt);
}

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

    char treno[5];
    int status;

    printf("\nConfigurazione Treno\n");

    // Get the required information
    printf("Matricola Treno: ");
    if(getInput(5, treno, false) < 0) return;


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call configurazione_treno(?)",
conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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


    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

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

```
        print_stmt_error (prepared_stmt, "Errore, impossibile ottenere la
composizione del treno.");
    }

    do {
        dump_result_set(conn, prepared_stmt, "");
        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 composizione(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];

    char treno[5];
    int status;

    printf("\nComposizione Treno\n");

    // Get the required information
    printf("Matricola Treno: ");
    if(getInput(5, treno, false) < 0) return;


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call composizione_treno(?)", conn))
{
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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


    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore, impossibile ottenere la
configurazione del treno.");
    }

    do {
        dump_result_set(conn, prepared_stmt, "");
        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 fermate(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];

    int tratta;
    int status;

    printf("\nFermate della tratta\n");

    // Get the required information
    printf("ID Tratta: ");
    if(getInteger(&tratta) < 0)
    return;


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call fermate_tratta(?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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


    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore, impossibile ottenere le fermate
della tratta.");
    }

    do {
        dump_result_set(conn, prepared_stmt, "");
        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 tratte(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;

    int status;
```

```
    printf("\nTratte giornaliere\n");


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call tratte_giornaliere()", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore, impossibile ottenere le fermate
della tratta.");
    }

    do {
        dump_result_set(conn, prepared_stmt, "");
        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);
}



void admin(MYSQL *conn, char *username) {

    printf("\033[2J\033[H");

    if(mysql_change_user(conn, "admin", "Lineagialla1!", "AziendaFerroviaria")) {
        fprintf(stderr, "mysql_change_user() failed\n");
        exit(EXIT_FAILURE);
    }

    struct sigaction act, sa;

    memset (&act, '\0', sizeof(act));


    printf(ANSI_COLOR_YELLOW "**********************\n" ANSI_COLOR_RESET);
    printf(ANSI_COLOR_YELLOW "***Connected as admin***\n" ANSI_COLOR_RESET);
    printf(ANSI_COLOR_YELLOW "**********************\n" ANSI_COLOR_RESET);
    while(1) {

        printf("%s-admin$ ", username);
        getInput(20, command, false);

        sigaction(SIGINT, &act, NULL);

        if(!strcmp(command, "quit")) {
            printf(ANSI_COLOR_RED"Arrivederci!\n" ANSI_COLOR_RESET);
            return;
        } else if(!strcmp(command, "help")) {
            printf(ANSI_COLOR_CYAN  "******************************"  "\n");
            printf(  "*** Comandi Amministratore ***"  "\n");
            printf( "******************************" ANSI_COLOR_RESET "\n");
            printf(ANSI_COLOR_MAGENTA "addlavoratore" ANSI_COLOR_RESET " - per
aggiungere un lavoratore\n");
```

```
            printf(ANSI_COLOR_MAGENTA "addazienda" ANSI_COLOR_RESET "- per
aggiungere un'azienda\n");
        printf(ANSI_COLOR_MAGENTA "addpasseggero" ANSI_COLOR_RESET "- per
aggiungere un passeggero\n");
            printf(ANSI_COLOR_MAGENTA "addprenotazione" ANSI_COLOR_RESET " - per
aggiungere una prenotazione\n");
            printf(ANSI_COLOR_MAGENTA "addspedizione" ANSI_COLOR_RESET " - per
aggiungere una spedizione\n");
            printf(ANSI_COLOR_MAGENTA "addservizio" ANSI_COLOR_RESET " - per
aggiungere un servizio ferroviario\n");
            printf(ANSI_COLOR_MAGENTA "addturno" ANSI_COLOR_RESET " - per
aggiungere un turno\n");
            printf(ANSI_COLOR_MAGENTA "serviziotratta" ANSI_COLOR_RESET " - per
modificare la tratta di un servizio\n");
            printf(ANSI_COLOR_MAGENTA "serviziotreno" ANSI_COLOR_RESET " - per
modificare il treno di un servizio\n");
            printf(ANSI_COLOR_MAGENTA "comptreno" ANSI_COLOR_RESET " - per
ottenere la composizione di un treno\n");
            printf(ANSI_COLOR_MAGENTA "configtreno" ANSI_COLOR_RESET " - per
ottenere la configurazione di un treno\n");
            printf(ANSI_COLOR_MAGENTA "assegnaturno" ANSI_COLOR_RESET " - per
assegnare un turno ad un lavoratore\n");
            printf(ANSI_COLOR_MAGENTA "malattia" ANSI_COLOR_RESET " - per
inserire un lavoratore in malattia\n");
            printf(ANSI_COLOR_MAGENTA "freelavoratori" ANSI_COLOR_RESET " - per
vedere i lavoratori liberi\n");
            printf(ANSI_COLOR_MAGENTA "tratte" ANSI_COLOR_RESET " - per ottenere
le tratte giornaliere\n");
            printf(ANSI_COLOR_MAGENTA "fermatetratta" ANSI_COLOR_RESET " - per
ottenere le fermate di una tratta\n");
            printf(ANSI_COLOR_YELLOW "quit" ANSI_COLOR_RESET " - per uscire
dall'applicazione\n");
            printf(ANSI_COLOR_YELLOW "clear" ANSI_COLOR_RESET " - per pulire lo
schermo\n");
        printf(ANSI_COLOR_CYAN  "******************************" ANSI_COLOR_RESET
"\n");
        }  else if(!strcmp(command, "clear")) {
            printf("\033[2J\033[H");
        }  else if(!strcmp(command, "addlavoratore")) {
            aggiungi_lavoratore(conn);
        }  else if(!strcmp(command, "addturno")) {
            aggiungi_turno(conn);
        }  else if(!strcmp(command, "assegnaturno")) {
            assegna_turno(conn);
        }  else if(!strcmp(command, "malattia")) {
            richiesta_malattia(conn);
        }  else if(!strcmp(command, "addazienda")) {
            aggiungi_azienda(conn);
        }  else if(!strcmp(command, "addspedizione")) {
            aggiungi_spedizione(conn);
        }  else if(!strcmp(command, "addpasseggero")) {
            aggiungi_passeggero(conn);
        }  else if(!strcmp(command, "addprenotazione")) {
            aggiungi_prenotazione(conn);
        }  else if(!strcmp(command, "addservizio")) {
            aggiungi_servizio(conn);
        } else if(!strcmp(command, "freelavoratori")) {
            lavoratori_disponibili(conn);
        } else if(!strcmp(command, "serviziotratta")) {
            modifica_servizio(conn);
        } else if(!strcmp(command, "serviziotreno")) {
```

```
            modifica_treno(conn);
        } else if(!strcmp(command, "configtreno")) {
            configurazione(conn);
        } else if(!strcmp(command, "comptreno")) {
            composizione(conn);
        } else if(!strcmp(command, "fermatetratta")) {
            fermate(conn);
        } else if(!strcmp(command, "tratte")) {
            tratte(conn);
        } else {
            printf("comando %s non riconosciuto, digita help per aiuto\n",
command);
        }

        sigaction(SIGINT, &sa, NULL);
    }
}
```

lavoratore.c
Utilizzato per l'accesso come lavoratore e per far accedere all'utente a tutte le procedure del suo ruolo.

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

#include "defines.h"
#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_CYAN    "\x1b[36m"

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

    char cf[16];
    int status;

    printf("\n************\nReport Turni\n************\n");

    // Get the required information
    printf("CF: ");
    if(getInput(45, cf, false) < 0) return;


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call report_lavoratore(?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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

```
    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nell'ottenere i turni.");
    }

    do {
        dump_result_set(conn, prepared_stmt, "Turni per il lavoratore
selezionato");
        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);
}



void lavoratore(MYSQL *conn, char *username) {
    char command[20];

    printf("\033[2J\033[H");

    if(mysql_change_user(conn, "lavoratore",
"Lineagialla1!","AziendaFerroviaria")) {
        fprintf(stderr, "mysql_change_user() failed\n");
        exit(EXIT_FAILURE);
    }

  printf(ANSI_COLOR_GREEN  "************************************\n");
    printf(                  "*Connesso al sistema come lavoratore*\n");
    printf(                  "************************************\n"
ANSI_COLOR_RESET);
    printf("Inserisci un comando, digita help per aiuto\n\n");


    struct sigaction act, sa;

    memset (&act, '\0', sizeof(act));



    while(1) {

        printf("%s-lavoratore$ ", username);
        getInput(20, command, false);

        sigaction(SIGINT, &act, NULL);

        if(!strcmp(command, "quit")) {
            printf(ANSI_COLOR_RED"Arrivederci!\n" ANSI_COLOR_RESET);
            return;
        } else if(!strcmp(command, "help")) {
```

```
            printf(ANSI_COLOR_GREEN "*** Comandi lavoratore ***" ANSI_COLOR_RESET
"\n");
            printf(ANSI_COLOR_CYAN "turni" ANSI_COLOR_RESET " - report dei turni"
"\n");;
            printf(ANSI_COLOR_YELLOW "quit" ANSI_COLOR_RESET " - per uscire
dall'applicazione\n");
            printf(ANSI_COLOR_YELLOW  "clear" ANSI_COLOR_RESET " - per pulire il
terminale\n");
            printf(ANSI_COLOR_GREEN "*******************************"
ANSI_COLOR_RESET "\n");
    } else if(!strcmp(command, "turni")) {
            turni(conn);
        } else {
            printf("comando %s non riconosciuto, digita help per aiuto\n",
command);
        }

        sigaction(SIGINT, &sa, NULL);
    }
}
```

manutentore.c
Utilizzato per l'accesso come manutentore e per far accedere all'utente a tutte le procedure del suo ruolo.

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

#include "defines.h"
#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_CYAN    "\x1b[36m"
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"

static void report(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[4];

    int id;
    char info[500];
    int mr_id;
    MYSQL_TIME  *date;
    date = malloc(sizeof(MYSQL_TIME));


    printf("\n*********************\nReport
Manutenzione\n*********************\n");

    // Get the required information
    //printf("ID REPORT: ");
    //if(getInteger(&id) < 0) return;
    printf("Inserisci data di manutenzione(formato dd/mm/yyyy): ");
    if(getDate(date) < 0) return;
    printf("Inserisci ID materiale rotabile: ");
    if(getInteger(&mr_id) < 0) return;
    printf("Descrizione manutenzione: ");
    if(getInput(500, info, false) < 0) return;
```

```
    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call report_manutenzione(?,?,?,?)",
conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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

    param[1].buffer_type = MYSQL_TYPE_DATE;
    param[1].buffer = date;
    param[1].buffer_length = sizeof(*date);

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

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

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nell'inserimento del report.");
        goto out;
    }

    printf("Report aggiunto con successo...\n");

        out:
        free(date);
    mysql_stmt_close(prepared_stmt);

}



void manutentore(MYSQL *conn, char *username) {
    char command[20];

    printf("\033[2J\033[H");

    if(mysql_change_user(conn, "manutentore",
"Lineagialla1!","AziendaFerroviaria")) {
        fprintf(stderr, "mysql_change_user() failed\n");
        exit(EXIT_FAILURE);
    }
```

```
    printf(ANSI_COLOR_GREEN  "************************************\n");
    printf(  "*Connesso al sistema come manutentore*\n");
    printf(  "************************************\n" ANSI_COLOR_RESET);
    printf("Inserisci un comando, digita help per aiuto\n\n");

    struct sigaction act, sa;

    memset (&act, '\0', sizeof(act));



    while(1) {

        printf("%s-Manutentore$ ", username);
        getInput(20, command, false);

        sigaction(SIGINT, &act, NULL);

        if(!strcmp(command, "quit")) {
            printf(ANSI_COLOR_RED"Arrivederci!\n" ANSI_COLOR_RESET);
            return;
        } else if(!strcmp(command, "help")) {
            printf(ANSI_COLOR_GREEN "*** Comandi manutentore ***"
ANSI_COLOR_RESET "\n");
            printf(ANSI_COLOR_CYAN "report" ANSI_COLOR_RESET " - report della
manutenzione" "\n");;
            printf(ANSI_COLOR_YELLOW "quit" ANSI_COLOR_RESET " - per uscire
dall'applicazione\n");
            printf(ANSI_COLOR_YELLOW "clear" ANSI_COLOR_RESET " - per pulire il
terminale\n");
            printf(ANSI_COLOR_GREEN "******************************"
ANSI_COLOR_RESET "\n");
    } else if(!strcmp(command, "report")) {
            report(conn);
        } else {
            printf("comando %s non riconosciuto, digita help per aiuto\n",
command);
        }

        sigaction(SIGINT, &sa, NULL);
    }
}
```

controllore.c
Utilizzato per l'accesso come controllore e per far accedere all'utente a tutte le procedure del suo ruolo.

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

#include "defines.h"
#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ANSI_COLOR_CYAN    "\x1b[36m"
#define ANSI_COLOR_YELLOW  "\x1b[33m"

static void verifica(MYSQL *conn) {
```

```
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];

    char codice[5];
    int status;

    printf("\n*********************\nVerifica
Prenotazione\n*********************\n");

    // Get the required information
    printf("Codice Prenotazione: ");
    if(getInput(5, codice, false) < 0) return;


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call verifica_prenotazione(?)",
conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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


    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nella verifica della
prenotazione.");
    }

    do {
        dump_result_set(conn, prepared_stmt, "");
        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 convalida(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];

    char codice[5];


    printf("\n*********************\nConvalida
Prenotazione\n*********************\n");
```

```
    // Get the required information
    printf("Codice Prenotazione: ");
    if(getInput(5, codice, false) < 0) return;


    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call convalida_prenotazione(?)",
conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Impossibile inizializzare lo
statement.\n", false);
    }

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

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


    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Errore nel binding dei
parametri.\n", true);
    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "Errore nella convalida della
prenotazione.");
    }else {
        printf("Convalida eseguita correttamente\n");
    }

    mysql_stmt_close(prepared_stmt);
}



void controllore(MYSQL *conn, char *username) {
    char command[20];

    printf("\033[2J\033[H");

    if(mysql_change_user(conn, "controllore",
"Lineagialla1!","AziendaFerroviaria")) {
        fprintf(stderr, "mysql_change_user() failed\n");
        exit(EXIT_FAILURE);
    }

    printf(ANSI_COLOR_GREEN  "*************************************\n");
    printf(                  "*Connesso al sistema come controllore*\n");
    printf(                  "*************************************\n"
ANSI_COLOR_RESET);
    printf("Inserisci un comando, digita help per aiuto\n\n");


    struct sigaction act, sa;

    memset (&act, '\0', sizeof(act));
```

```
    while(1) {

        printf("%s-Controllore$ ", username);
        getInput(20, command, false);

        sigaction(SIGINT, &act, NULL);

        if(!strcmp(command, "quit")) {
            printf(ANSI_COLOR_RED"Arrivederci!\n" ANSI_COLOR_RESET);
            return;
        } else if(!strcmp(command, "help")) {
            printf(ANSI_COLOR_GREEN "***** Comandi controllore *****"
ANSI_COLOR_RESET "\n");
            printf(ANSI_COLOR_CYAN "verifica" ANSI_COLOR_RESET " - verifica della
prenotazione" "\n");;
        printf(ANSI_COLOR_CYAN "convalida" ANSI_COLOR_RESET " - convalida
prenotazione" "\n");;
            printf(ANSI_COLOR_YELLOW "quit" ANSI_COLOR_RESET " - per uscire
dall'applicazione\n");
            printf(ANSI_COLOR_YELLOW "clear" ANSI_COLOR_RESET " - per pulire il
terminale\n");
            printf(ANSI_COLOR_GREEN "*******************************"
ANSI_COLOR_RESET "\n");
    } else if(!strcmp(command, "verifica")) {
            verifica(conn);
        }else if(!strcmp(command, "convalida")) {
                            convalida(conn);
                    } else {
            printf("comando %s non riconosciuto, digita help per aiuto\n",
command);
        }

        sigaction(SIGINT, &sa, NULL);
    }
}
```

utils.c

Contiene tutte le funzioni utili per interagire con la base di dati nel codice c.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <termios.h>
#include <errno.h>

#include "defines.h"

static volatile sig_atomic_t signo;
typedef struct sigaction sigaction_t;
static void handler(int s);

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
    }
}

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);
}

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;
}

//TODO refactor from this
int getInput(unsigned int lung, char *stringa, bool hide) {
    char c;
    unsigned int i;

    // Dichiara le variabili necessarie ad un possibile mascheramento dell'input
```

```
    struct sigaction sa, savealrm, saveint, savehup, savequit, saveterm;
    struct sigaction 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 pi lung - 1 caratteri
    for(i = 0; i < lung; i++) {
        int size = fread(&c, sizeof(char), 1, stdin);
        if(size == 0 && errno == EINTR) {
            if(hide) {
                (void) tcsetattr(fileno(stdin), TCSAFLUSH, &oterm);

                (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);
            }
            return -1;
        }
        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 pi 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 strlen(stringa);
}

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


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;
    MYSQL_TIME *time;
    size_t attr_size;

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

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

    if (num_fields > 0) {
        /* there is a result set to fetch */
        printf("%s\n", title);

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

        dump_result_set_header(rs_metadata);

        fields = mysql_fetch_fields(rs_metadata);

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

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

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

            // Setup the binding for the current parameter
```

```
            rs_bind[i].buffer_type = fields[i].type;
            rs_bind[i].buffer = malloc(attr_size + 1);
            rs_bind[i].buffer_length = attr_size + 1;

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

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

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

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

            putchar('|');

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

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

                switch (rs_bind[i].buffer_type) {

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

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

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

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

                    case MYSQL_TYPE_LONG:
                    case MYSQL_TYPE_SHORT:
                    case MYSQL_TYPE_TINY:
                    case MYSQL_TYPE_LONGLONG:
```

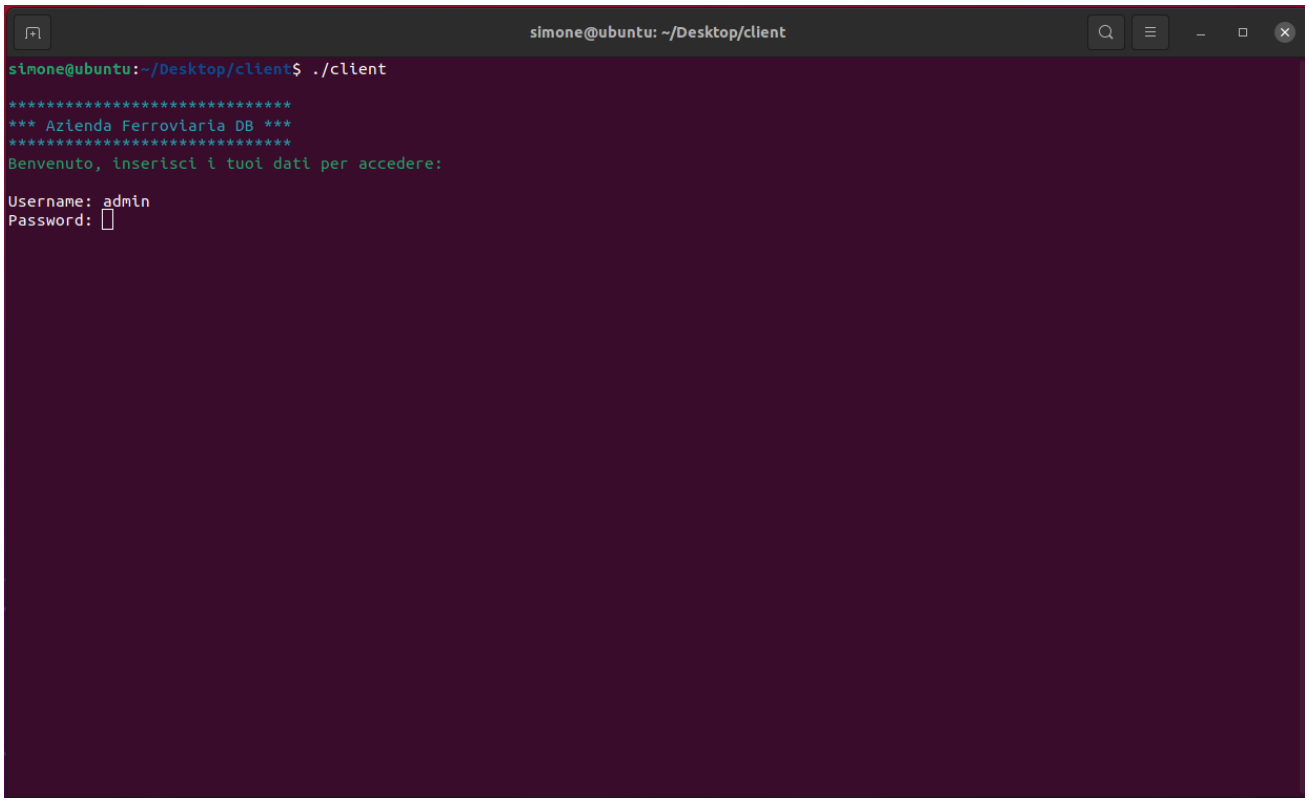
```
                        printf(" %-*d |", (int)fields[i].max_length, *(int
*)rs_bind[i].buffer);
                        break;

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

                    case MYSQL_TYPE_TIME:
                        time = (MYSQL_TIME *)rs_bind[i].buffer;
                        printf("%.02d:%.02d %*s|", time->hour, time->minute,
(int)fields[i].max_length - (int)strlen("hh:mm"), "");
                        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);
    }
}

int getInteger(int *dest) {
    int size = scanf("%d", dest);
    if(size < 0 && errno == EINTR) {
        return -1;
    }

    getchar();
    return size;
}

int getDate(MYSQL_TIME *date) {
    unsigned int day, month, year;

    int size = scanf("%d/%d/%d",&day,&month,&year);
    if(size < 0 && errno == EINTR) {
        return -1;
    }

    getchar();

    date->day = day;
    date->month = month;
    date->year = year;

    return size;
}
```

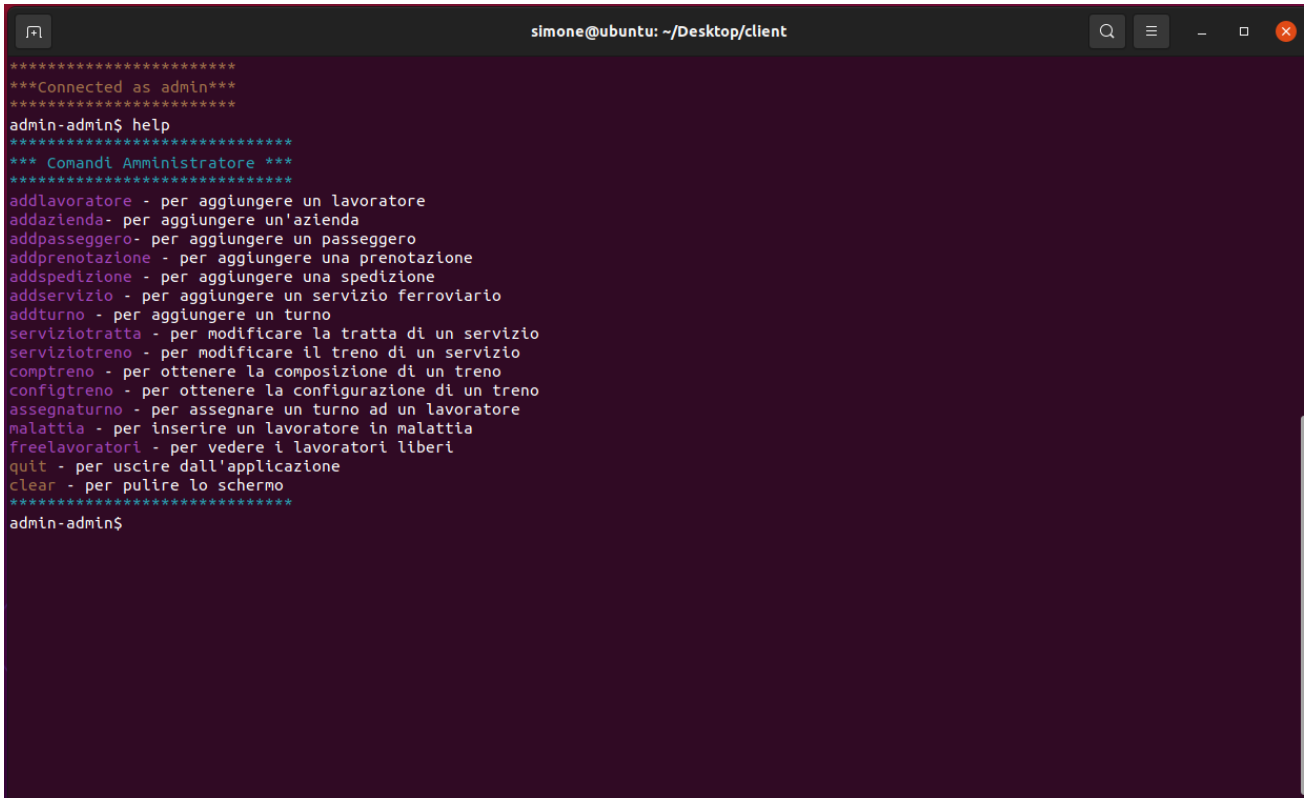
```
int getTime(MYSQL_TIME *time) {
    int size = scanf("%u:%u",&time->hour, &time->minute);
    if(size < 0 && errno == EINTR) {
        return -1;
    }
    time->second = 0;
    return size;
}
```

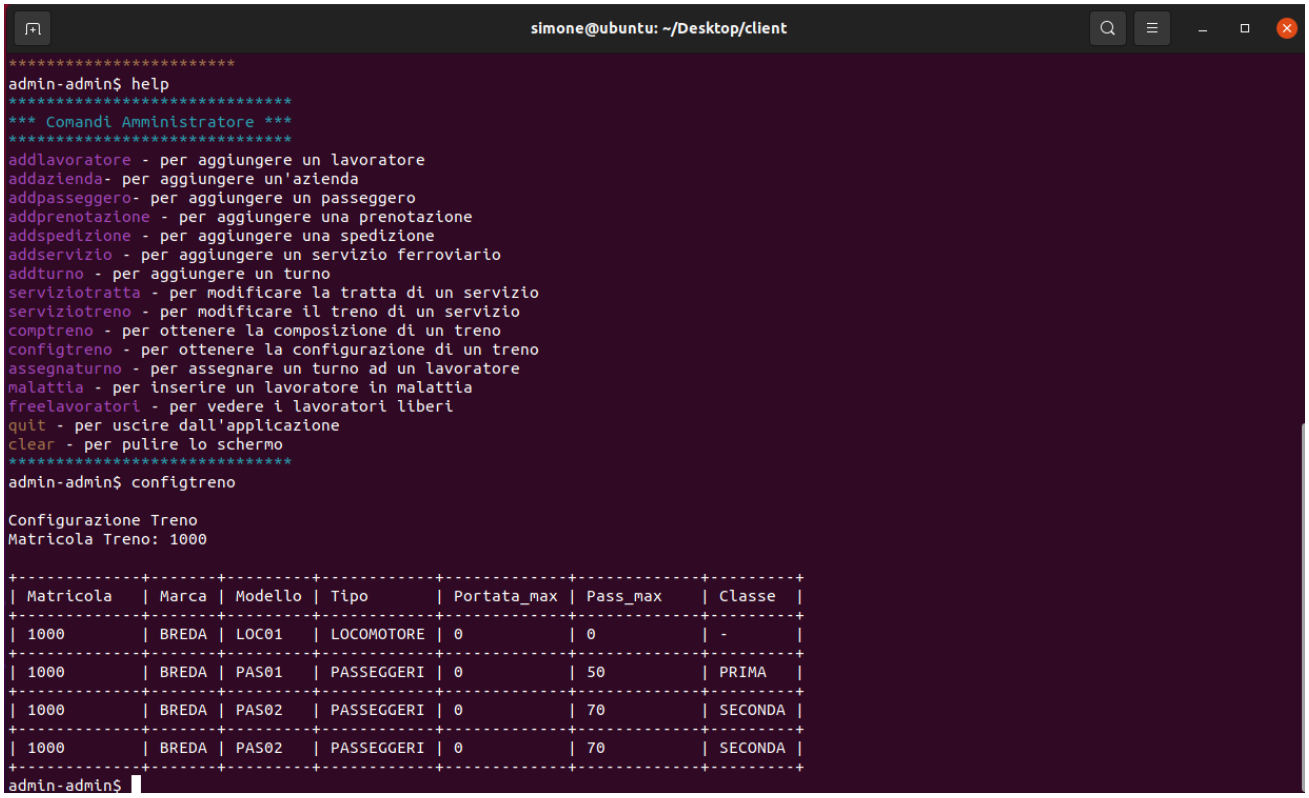
defines.h

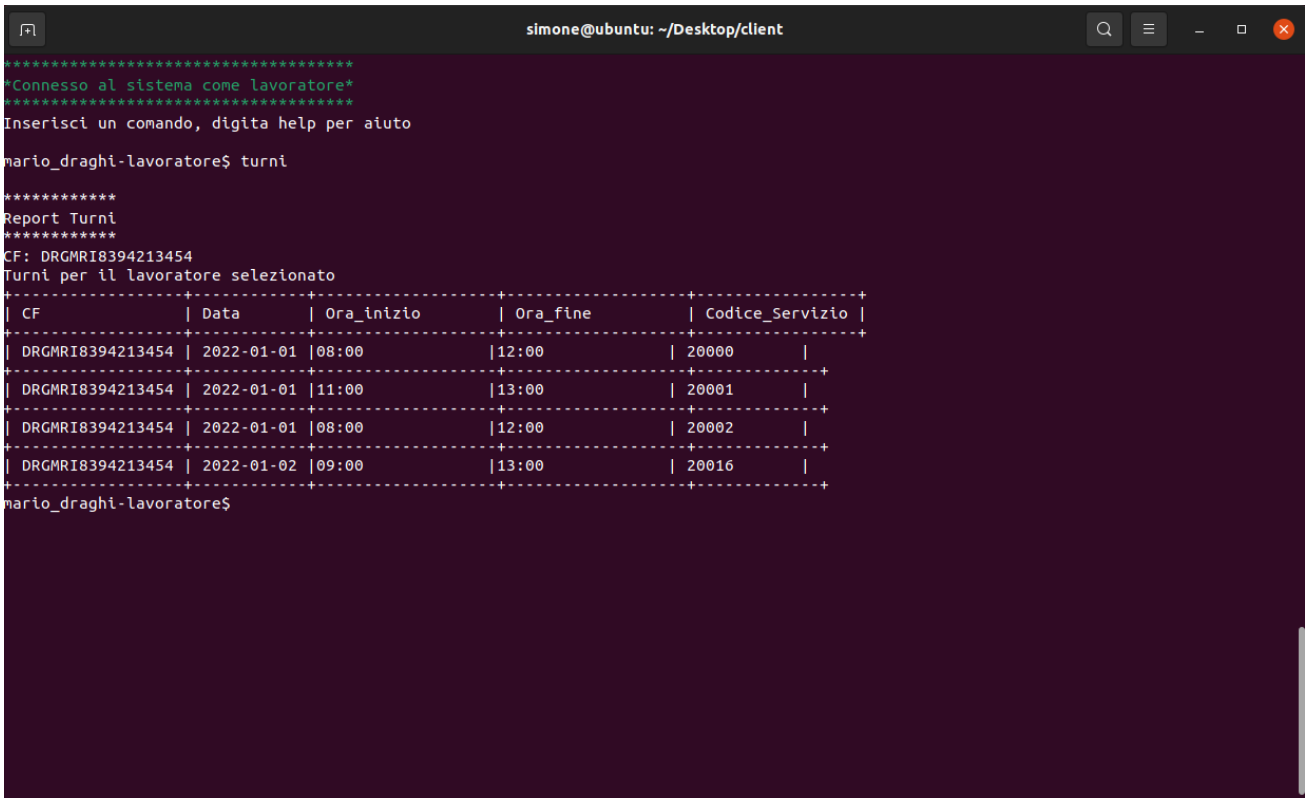
```
#include <stdbool.h>
#include <mysql.h>


extern void print_stmt_error (MYSQL_STMT *stmt, char *message);
extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);
extern void print_error (MYSQL *conn, char *message);
int getInput(unsigned int lung, char *stringa, bool hide);
extern void lavoratore(MYSQL *conn, char *username);
extern void controllore(MYSQL *conn, char *username);
extern void manutentore(MYSQL *conn, char *username);
extern void admin(MYSQL *conn, char *username);
extern void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);
extern void finish_with_error(MYSQL *conn, char *message);
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message,
bool close_stmt);
extern void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);
int getInteger(int *dest);
int getDate(MYSQL_TIME *date);
int getTime(MYSQL_TIME *time);
```

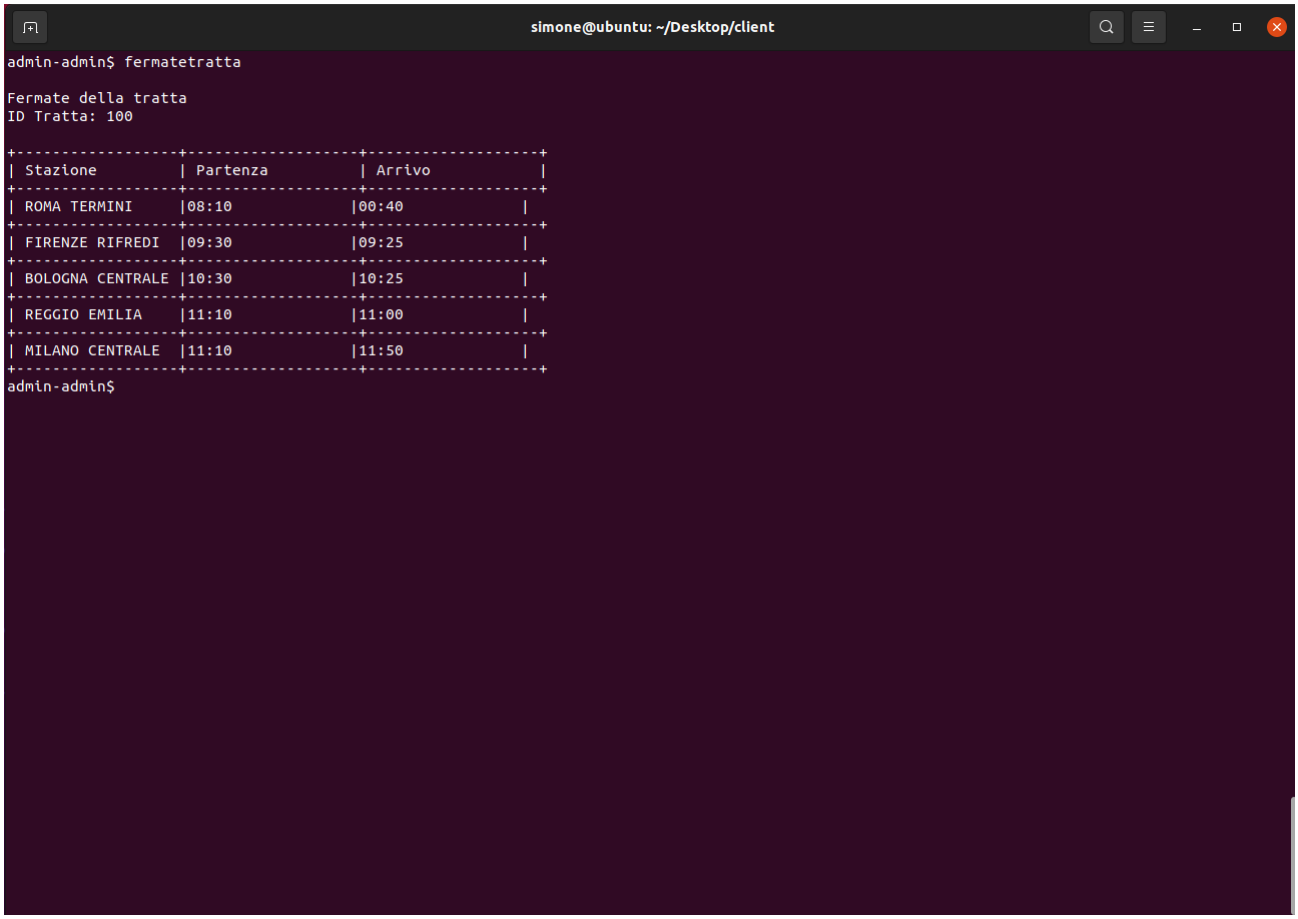
Print screen del Front End

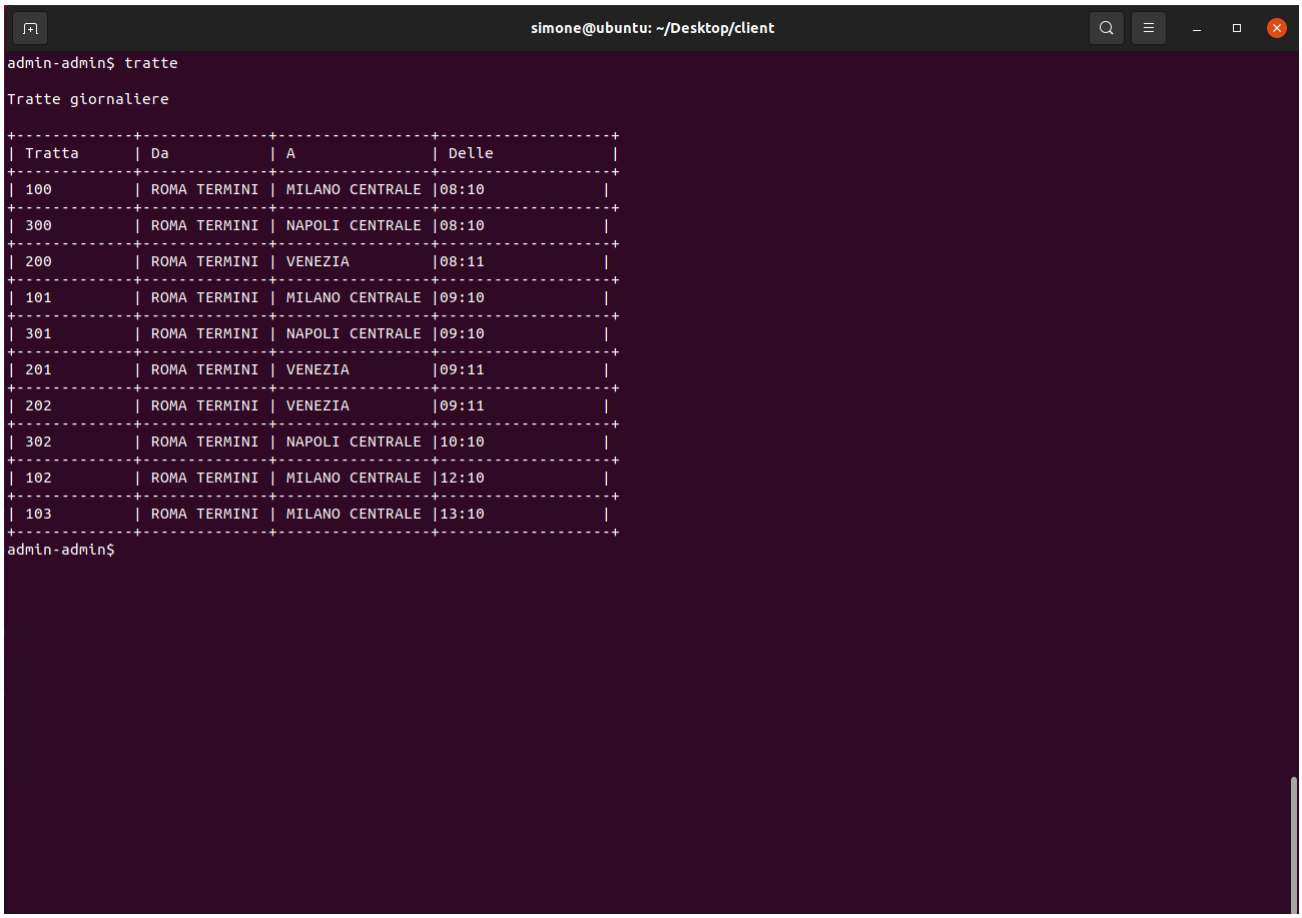


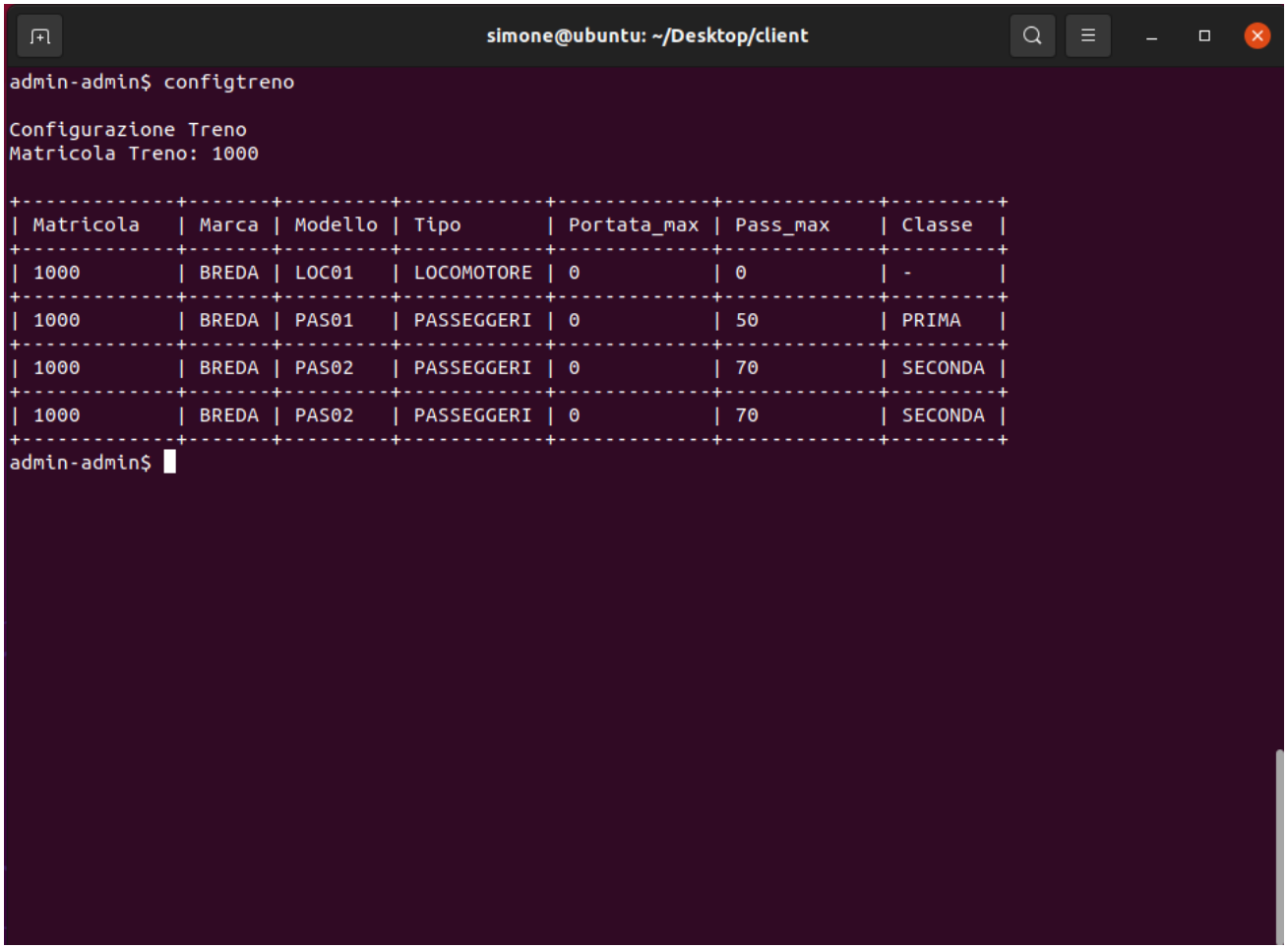



simone@ubuntu: ~/Desktop/client
****
admin-admin$ help
****
*** Comandi Amministratore ***
****
addlavoratore - per aggiungere un lavoratore
addazienda- per aggiungere un'azienda
addpasseggero- per aggiungere un passeggero
addprenotazione - per aggiungere una prenotazione
addspedizione - per aggiungere una spedizione
addservizio - per aggiungere un servizio ferroviario
addturno - per aggiungere un turno
serviziotratta - per modificare la tratta di un servizio
serviziotreno - per modificare il treno di un servizio
comptreno - per ottenere la composizione di un treno
configtreno - per ottenere la configurazione di un treno
assegnaturno - per assegnare un turno ad un lavoratore
malattia - per inserire un lavoratore in malattia
freelavoratori - per vedere i lavoratori liberi
quit - per uscire dall'applicazione
clear - per pulire lo schermo
****
admin-admin$ configtreno
Configurazione Treno
Matricola Treno: 1000
Matricola	Marca	Modello	Tipo	Portata_max	Pass_max	Classe
1000	BREDA	LOC01	LOCOMOTORE	0	0	-
1000	BREDA	PAS01	PASSEGGERI	0	50	PRIMA
1000	BREDA	PAS02	PASSEGGERI	0	70	SECONDA
1000	BREDA	PAS02	PASSEGGERI	0	70	SECONDA
admin-admin$


simone@ubuntu: ~/Desktop/client
****
Connesso al sistema come lavoratore
****
Inserisci un comando, digita help per aiuto
mario_draghi-lavoratore$ turni
****
Report Turni
****
CF: DRGMRI8394213454
Turni per il lavoratore selezionato
CF	Data	Ora_inizio	Ora_fine	Codice_Servizio
DRGMRI8394213454	2022-01-01	08:00	12:00	20000
DRGMRI8394213454	2022-01-01	11:00	13:00	20001
DRGMRI8394213454	2022-01-01	08:00	12:00	20002
DRGMRI8394213454	2022-01-02	09:00	13:00	20016
mario_draghi-lavoratore$


simone@ubuntu: ~/Desktop/client
admin-admin$ fermatetratta
Fermate della tratta
ID Tratta: 100
Stazione	Partenza	Arrivo
ROMA TERMINI	08:10	00:40
FIRENZE RIFREDI	09:30	09:25
BOLOGNA CENTRALE	10:30	10:25
REGGIO EMILIA	11:10	11:00
MILANO CENTRALE	11:10	11:50
admin-admin$


simone@ubuntu: ~/Desktop/client
admin-admin$ tratte
Tratte giornaliere
Tratta	Da	A	Delle
100	ROMA TERMINI	MILANO CENTRALE	08:10
300	ROMA TERMINI	NAPOLI CENTRALE	08:10
200	ROMA TERMINI	VENEZIA	08:11
101	ROMA TERMINI	MILANO CENTRALE	09:10
301	ROMA TERMINI	NAPOLI CENTRALE	09:10
201	ROMA TERMINI	VENEZIA	09:11
202	ROMA TERMINI	VENEZIA	09:11
302	ROMA TERMINI	NAPOLI CENTRALE	10:10
102	ROMA TERMINI	MILANO CENTRALE	12:10
103	ROMA TERMINI	MILANO CENTRALE	13:10
admin-admin$


simone@ubuntu: ~/Desktop/client
admin-admin$ configtreno
Configurazione Treno
Matricola Treno: 1000
+-------------+-------+---------+------------+-------------+-------------+---------+
| Matricola | Marca | Modello | Tipo | Portata_max | Pass_max | Classe |
+-------------+-------+---------+------------+-------------+-------------+---------+
| 1000 | BREDA | LOC01 | LOCOMOTORE | 0 | 0 | - |
+-------------+-------+---------+------------+-------------+-------------+---------+
| 1000 | BREDA | PAS01 | PASSEGGERI | 0 | 50 | PRIMA |
+-------------+-------+---------+------------+-------------+-------------+---------+
| 1000 | BREDA | PAS02 | PASSEGGERI | 0 | 70 | SECONDA |
+-------------+-------+---------+------------+-------------+-------------+---------+
| 1000 | BREDA | PAS02 | PASSEGGERI | 0 | 70 | SECONDA |
+-------------+-------+---------+------------+-------------+-------------+---------+
admin-admin$