



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

SISTEMA DI GESTIONE DI TRASPORTO FERROVIARIO

0266403
Sara Da Canal

Indice

1. Descrizione del Minimondo.....	2
2. Analisi dei Requisiti	4
3. Progettazione concettuale	5
4. Progettazione logica	16
5. Progettazione fisica	21
Appendice: Implementazione.....	39

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
12-34	Tratta	Viaggio	Va fatta una differenza tra la tratta, ovvero il percorso tra due capolinea, e il viaggio, ovvero una singola istanza di percorrenza di una tratta
19	Conducente	Lavoratore	Le informazioni che specifica in questo punto sono relative sia ai conducenti (i.e. macchinisti) che ai capotreno, anche se la loro divisione viene specificata solo successivamente
30	Report	Resoconto	Report viene usato sia per riferirsi alle informazioni sui propri turni che il lavoratore può visualizzare, come succede in questo caso, che per riferirsi ai report di manutenzione inseribili per i veicoli

Specificazione 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 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).

Ciascun viaggio 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 lavoratori, 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 resoconto 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 viaggio 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.

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Treno	Mezzo di proprietà delle ferrovie che permette lo spostamento di merci o persone		Veicolo, tratta, lavoratore
Veicolo	Vagoni o locomotrici di proprietà delle ferrovie		Locomotrice, vagone
Tratta	Percorso seguito dai treni, composto da diverse fermate con definita partenza e arrivo		Fermata, treno, viaggio
Viaggio	Viaggio del treno che avviene lungo una determinata tratta ad uno specifico orario		Treno, tratta, fermata
Fermata	Stazione dove il treno ferma durante il suo percorso		Tratta, viaggio
Lavoratore	Impiegato delle ferrovie		Treno, turno
Macchinista	Impiegato delle ferrovie incaricato di guidare il treno	Conducente	Treno, turno, lavoratore
Capotreno	Lavoratore di turno sui treni passeggeri, tra le sue mansioni deve controllare i biglietti	Controllore	Treno, turno, lavoratore
Turno	Orario in cui un certo lavoratore è di servizio	Orario di lavoro	Treno, lavoratore

Biglietto	Strumento per prenotare un posto sul treno, contiene le informazioni del passeggero		Posto, passeggero
Passeggero	Persona che viaggia su un treno passeggeri, prenotando un biglietto	Acquirente	Biglietto
Merce	Beni trasportati da un treno merci		Treno, azienda
Azienda	Società che invia le sue merci tramite un treno merci	Società	Merce
Posto	Posto prenotabile su un determinato vagone di un treno passeggeri		Biglietto, vagone

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative a treno

I treni gestiti dal servizio sono caratterizzati da una matricola (codice univoco numerico di quattro cifre).

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.

Frasi relative a veicolo

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. 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).

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 lavoratore

Gli amministratori del servizio gestiscono anche i lavoratori, identificati da un codice fiscale, un nome, un cognome, una data di nascita ed un luogo di nascita.

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 biglietto

Si vuole anche realizzare, per quanto riguarda i treni passeggeri, una funzionalità di prenotazione di biglietto.

identificato per tratta e cui viene associato un posto disponibile sul treno con l'aggiunta di un codice di prenotazione univoco

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”.

Frase relative a passeggero

All'atto di acquisto di un biglietto l'acquirente deve indicare il proprio nome, cognome data di nascita, codice fiscale e numero di carta di credito.

Frase relative a merce

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. Per semplicità, si può assumere che in un vagone siano trasportate merci di una sola società, dirette ad una sola società.

Frase relative ad azienda

Si vuole inoltre tenere traccia delle informazioni di fatturazione delle aziende che inviano e ricevono la merce.

Frase relative a tratta

Inoltre, per ciascuna tratta, vengono mantenuti i capilinea di partenza ed arrivo. 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.

Frase relative a viaggio

Ciascun viaggio ha un insieme di fermate identificate dal nome della stazione, dalla città e dalla provincia in cui si trova.

Frase relative a fermata

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.

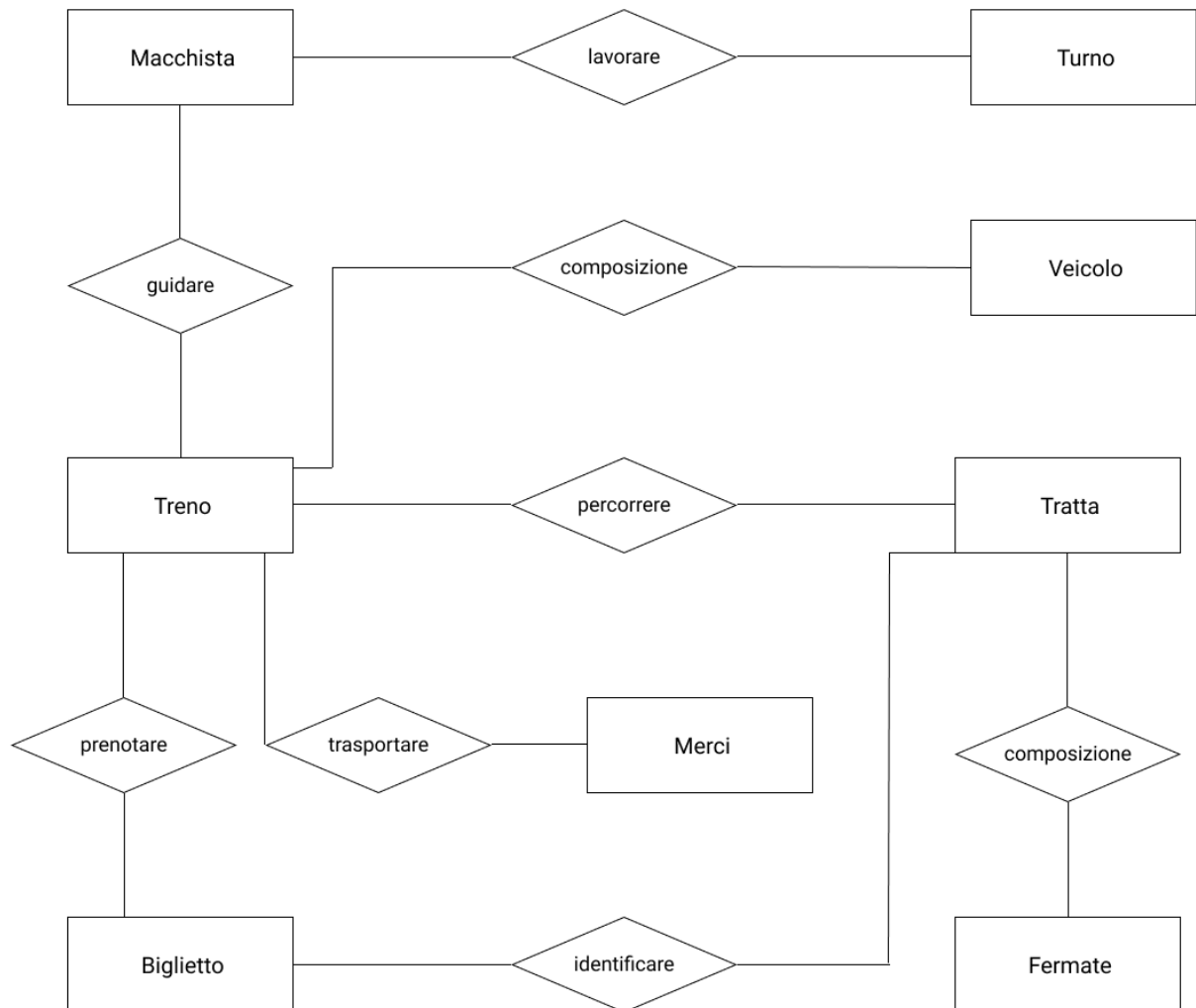
Frase relative a turno

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.

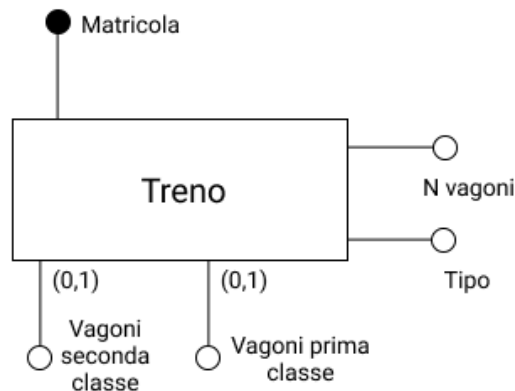
3. Progettazione concettuale

Costruzione dello schema E-R

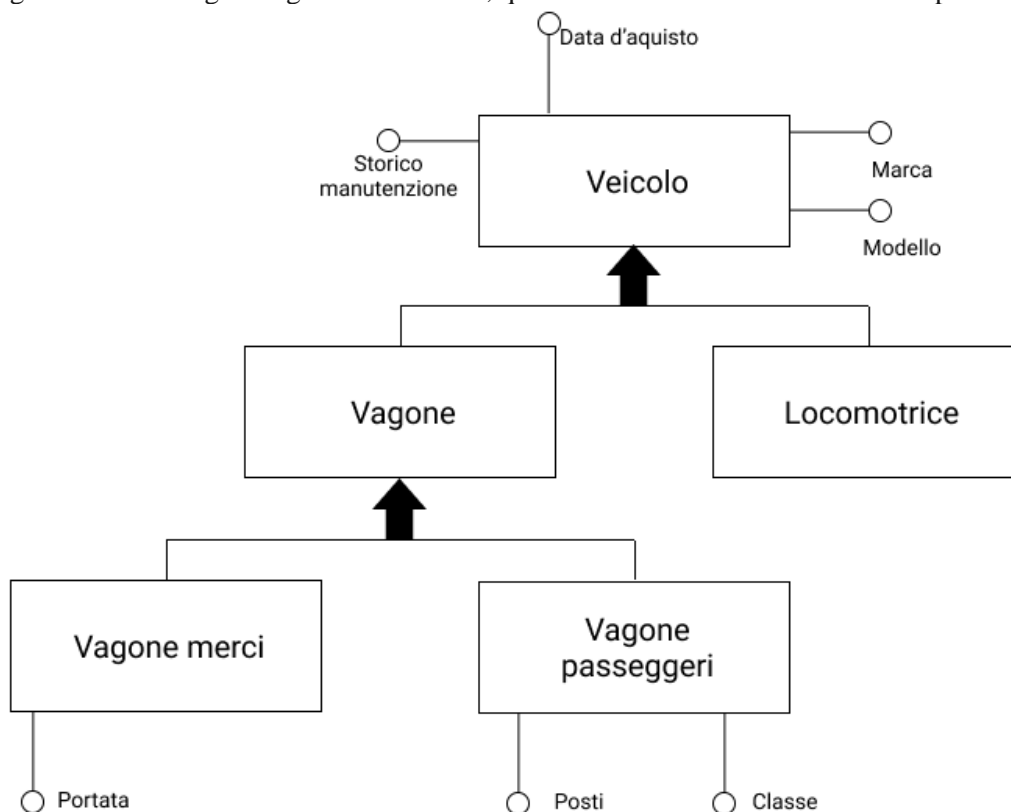
Per lo schema ER ho adottato una strategia mista: inizialmente ho individuato alcuni concetti principali creando uno scheletro che evidenziasse i loro collegamenti, ma poi piuttosto che raffinare lo schema nella sua interezza come richiederebbe una strategia top-down, mi sono concentrata su un'entità per volta per raffinarla, lasciando l'integrazione come ultimo passo e seguendo lo scheletro iniziale per eseguirla.



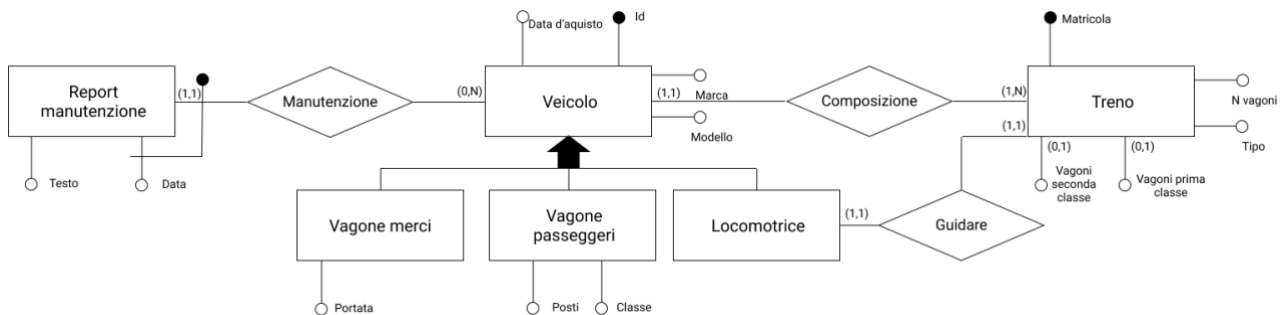
L'entità principale individuata è ovviamente il treno, quindi ho deciso di iniziare da lì inserendo gli attributi nominati esplicitamente, tra cui la matricola come identificatore, più un attributo tipo necessario a distinguere i treni merci e quelli passeggeri. Il treno passeggeri ha qualche attributo in più che ho deciso di lasciare come opzionale. In questa fase ho deciso di lasciare tutti gli attributi relativi al numero di vagoni anche se è ovvio che tra di loro sono ridondanti per occuparmene poi in una fase di revisione.



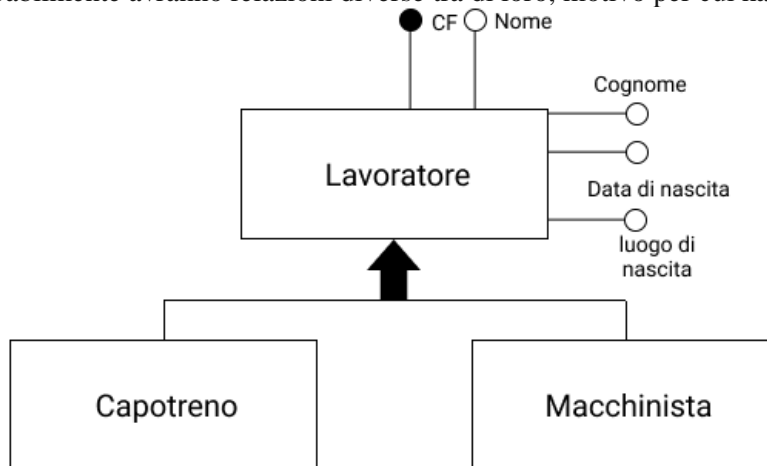
A questo punto sono passata all'entità veicolo. Analizzando le informazioni relative a veicolo ho notato che abbiamo veicoli di due tipi, vagoni oppure locomotrici, e che i vagoni a loro volta sono di tipo merci o passeggeri come il treno. Vagoni merci e passeggeri hanno troppi attributi diversi tra loro però per poter seguire la stessa logica seguita con il treno, quindi li ho mantenuti come entità separate.



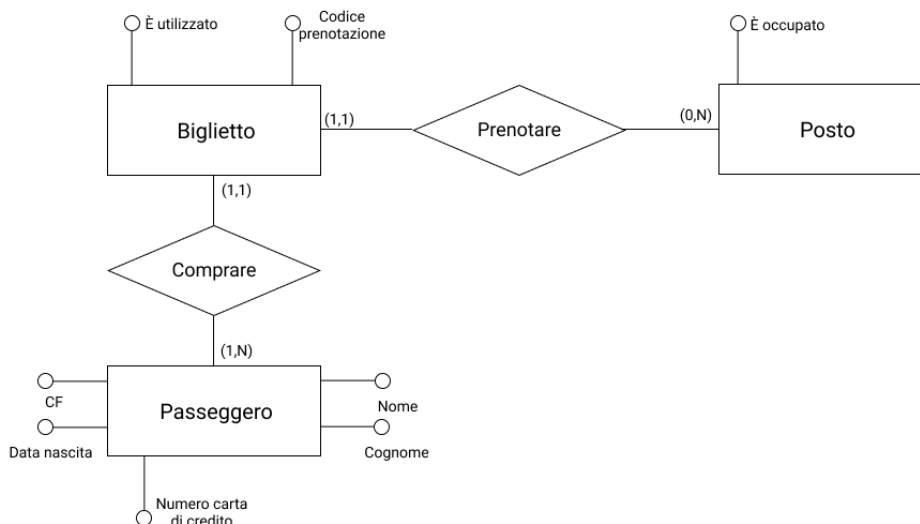
In una raffinazione ancora successiva ho notato due cose: l'entità vagone non aggiungeva nulla e poteva essere eliminata, e l'attributo storico di manutenzione è molto più complicato di un semplice attributo e andava reificato. Ho anche ritenuto necessario inserire un id per ogni veicolo dato che sembrava il modo più semplice per identificarli. In questa fase ho anche collegato treno e veicolo tra loro, perché anche se sono due entità principali, essendoci tra di loro una relazione di composizione le ho ritenute strettamente legate



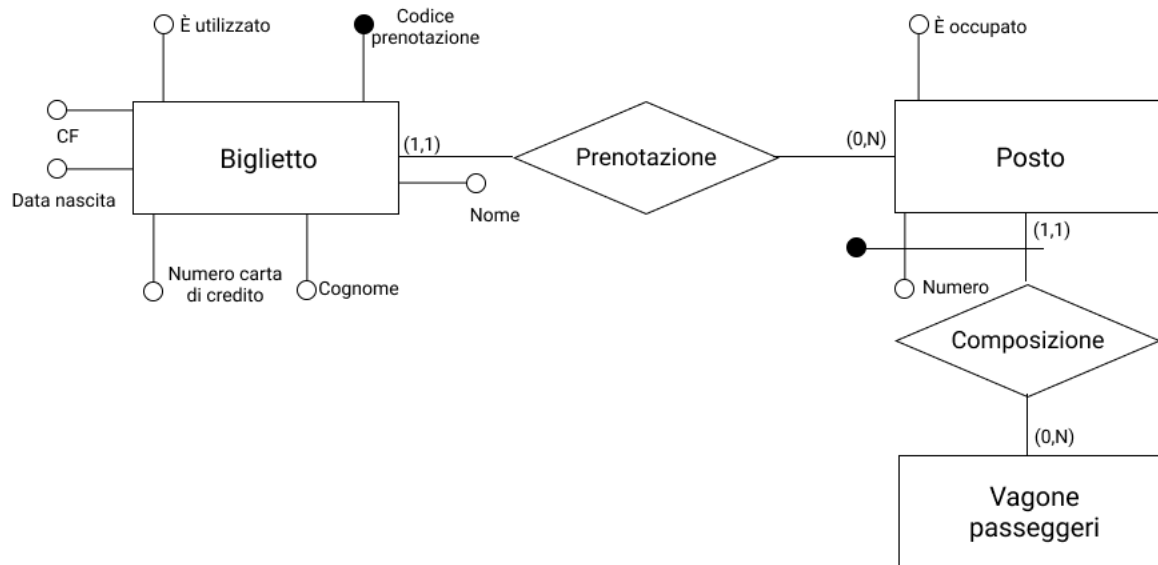
A questo punto ho esaminato l'entità macchinista evidenziata nello schema scheletro, che risulta essere una specializzazione dell'entità superiore lavoratore. L'altro tipo di lavoratori disponibili sono i capotreni. In questa fase ne macchinista ne capotreno hanno attributi specifici, però dato che hanno ruoli diversi probabilmente avranno relazioni diverse tra di loro, motivo per cui ha senso mantenere questa divisione.



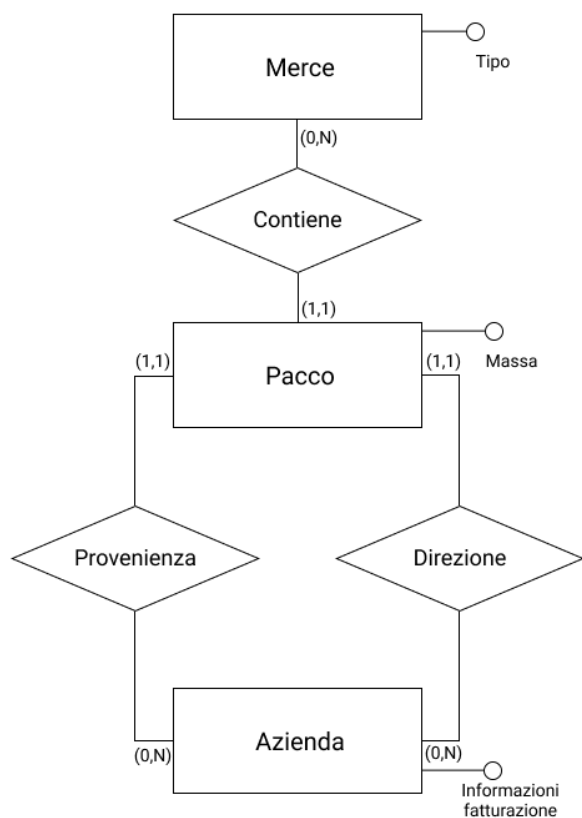
Un'altra entità individuata dallo schema scheletro è l'entità biglietto, legata all'entità passeggero. Lo scheletro evidenzia un collegamento tra biglietto e treno, ma sono giunta alla conclusione che quel collegamento vada risolto introducendo un'entità posto. Ho ritenuto necessario memorizzare se un posto fosse già occupato o meno nonostante questo attributo non fosse esplicitamente specificato



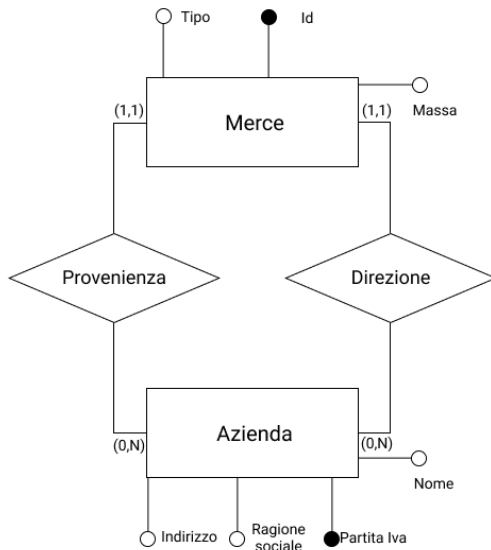
Ad una raffinazione successiva ho notato come effettivamente non servisse un'entità passeggero, dato che le informazioni del passeggero sono di interesse solo relativamente al biglietto specifico e non sembra utile mantenerle come entità separate. Ho anche deciso di evidenziare i collegamenti tra posto e vagoni passeggeri.



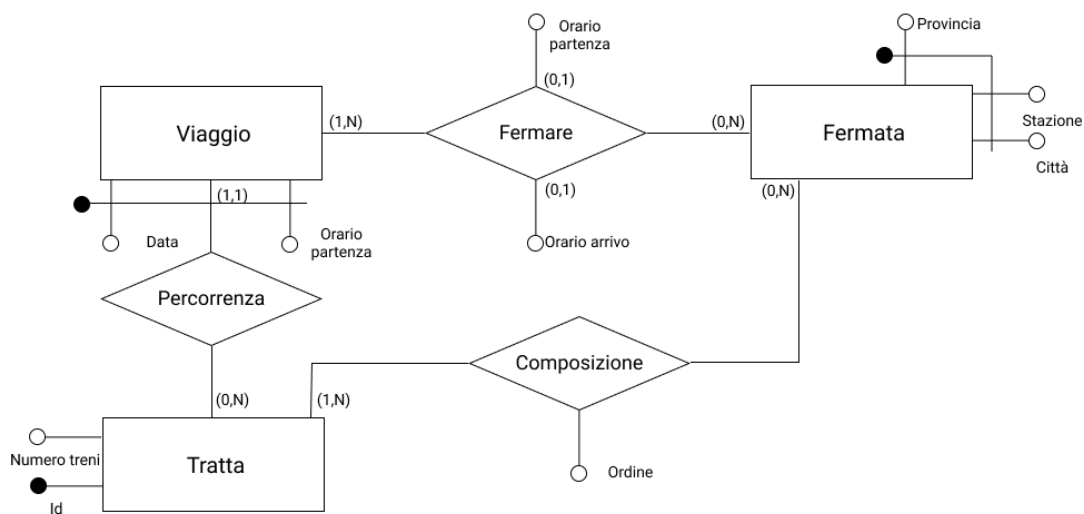
Finiti i biglietti sono passata all'entità merce, che inizialmente ho diviso in "merce", e "pacco", dove merce rappresenta un tipo di merce e pacco la quantità di merce effettivamente trasportata.



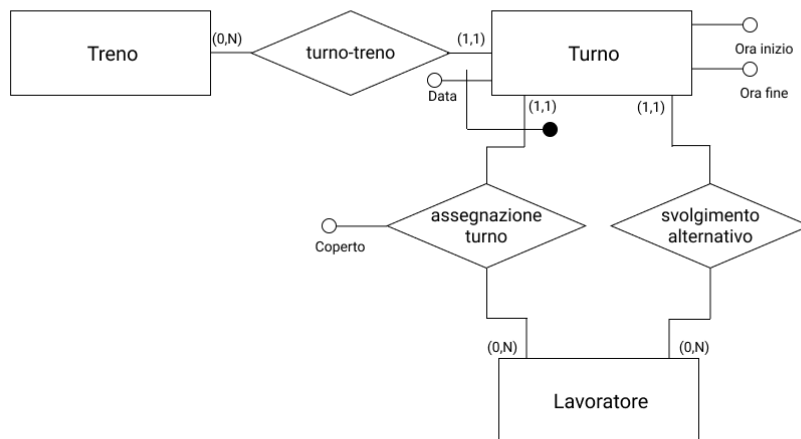
Successivamente ho notato che queste entità potevano effettivamente essere unite in un'unica entità, aggiungendo a pacco l'attributo tipo, dato che merce non aveva altri attributi. Ho mantenuto però il nominativo merce dato che era quello presente nelle specifiche. In questa fase ho deciso di aggiungere un codice identificativo sulla merce ed elaborare sulle informazioni di fatturazione delle aziende.



Andando ad analizzare le entità tratta e fermata, già presenti sullo schema scheletro, ho deciso di evidenziare già in questa fase i loro collegamenti, dato che tra di loro esiste una relazione di composizione. Come già era stato evidenziato, tratta si divide in “tratta” e “viaggio”, dove tratta rappresenta il percorso e viaggio uno specifico viaggio compiuto da uno specifico treno lungo quel percorso. Ho ritenuto utile collegarle entrambe a fermata dato che le relazioni evidenziano aspetti differenti.



L’ultima entità da analizzare più nel dettaglio prima di passare ad una fase di integrazione è turno, che ho pensato di identificare come il turno assegnato a uno specifico lavoratore in uno specifico giorno. Ho mantenuto separate le relazioni di assegnato e svolto per rendere possibile segnare i turni assegnati a qualcuno che poi è stato sostituito e quindi svolti da una persona diversa.



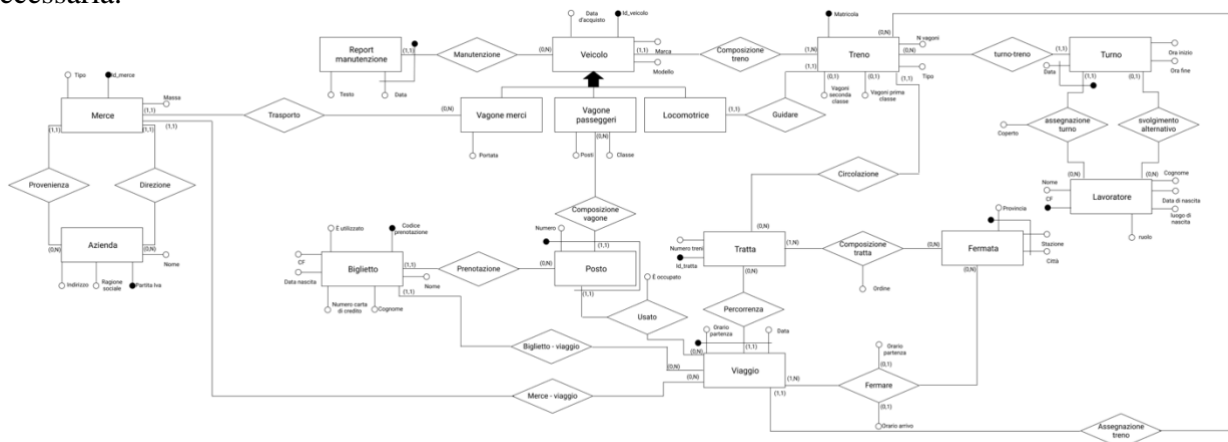
Integrazione finale

Negli schemi parziali ho tre relazioni diverse tutte con nome “composizione”, che sono diventate “composizione treno”, “composizione vagone” e “composizione tratta”.

Ho notato che le entità “capotreno” e “macchinista”, che avevo mantenuto nonostante non avessero differenza di attributi pensando avessero relazioni differenti tra di loro in realtà non risultano utili e quindi le ho eliminate aggiungendo a lavoratore l’attributo ruolo.

Ho anche tre entità identificate tramite un id, che quindi sono stati rinominati come “id_veicolo”, “id_tratta” e “id_merce”.

Ho anche notato che mentre inizialmente avevo identificato “posto” tramite vagone e numero, l’unica cosa che mi interessa in relazione a un posto è il suo essere occupato o meno per un determinato viaggio, motivo per cui anche viaggio è necessario per identificarlo. Se posto avesse avuto molti attributi oltre agli identificatori si sarebbero potute creare due entità, rappresentanti il posto fisico sul vagone e quel determinato posto utilizzato in uno specifico viaggio, ma dato che l’unica cosa di interesse relativa a posto è il suo essere occupato o meno la prima mi è parsa non necessaria.



Regole aziendali

1. I lavoratori con ruolo “capotreno” possono effettuare turni solo su treni di tipo “passeggeri”

2. La composizione dei treni deve rispettare il tipo: vagoni passeggeri compongono treni di tipo “passeggeri” e vagoni merci compongono treni di tipo “merci”
3. La durata di un turno non può essere maggiore di quattro ore
4. Le fermate che compongono una tratta devono sempre essere presenti nell’insieme di fermate associate a un viaggio che segue quella tratta
5. Uno stesso lavoratore non può effettuare più di cinque turni nella stessa settimana
6. Il valore dell’attributo “massa” della merce trasportata su un vagone non può essere maggiore del valore dell’attributo “portata” di quel vagone
7. Ogni viaggio deve essere compiuto da un treno associato con la tratta che il viaggio percorre

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Treno	Mezzo di proprietà delle ferrovie che permette lo spostamento di merci o persone lungo diverse tratte	Matricola, tipo, numero vagoni, numero vagoni 1° classe (opzionale), numero vagoni 2° classe (opzionale)	Matricola
Veicolo	Singolo vagone o locomotrice che compone il treno	Id_veicolo, data di acquisto, marca, modello	Id_veicolo
Vagone merci	Tipo di veicolo adibito al trasporto merci	Portata	Id_veicolo
Vagone passeggeri	Tipo di veicolo adibito al trasporto passeggeri	Posti, classe	Id_veicolo
Locomotrice	Tipo di veicolo che traina il treno		Id_veicolo
Report di manutenzione	Informazioni sugli interventi di manutenzione svolti su un determinato veicolo	Testo, data	Veicolo, data
Lavoratore	Impiegato delle ferrovie che lavora su turni	CF, nome, cognome, data di nascita, luogo di nascita, ruolo	CF
Biglietto	Contiene le informazioni relative alla prenotazione di un posto su uno specifico treno passeggeri	Codice prenotazione, nome, cognome, CF, data di nascita, numero carta di credito, è utilizzato	Codice prenotazione
Posto	Posto specifico su un determinato vagone passeggeri utilizzato in uno specifico viaggio	Numero, è occupato	Numero, vagone passeggeri, viaggio
Merce	Merce trasportata su un vagone merci tra un’azienda di provenienza e una di destinazione	Id_merce, tipo, massa	Id_merce
Azienda	Azienda che usa il sistema di ferrovie per inviare o ricevere merce	Partita IVA, nome, ragione sociale, indirizzo	Partita IVA

Tratta	Percorso tra due capolinea composto da un insieme di fermate ordinate	Id_tratta, numero treni	Id_tratta
Fermata	Stazione in cui ferma il treno	Stazione, provincia, città	Stazione, provincia, città
Viaggio	Viaggio di uno specifico treno seguendo una determinata tratta ad uno specifico orario	Orario partenza, data	Tratta, orario partenza, data
Turno	Turno di lavoro svolto da un lavoratore	Data, ora di inizio, ora di fine	Data, lavoratore

4. Progettazione logica

Volume dei dati

Concetto nello schema	Tipo ¹	Volume atteso
Treno	E	100
Veicolo	E	900
Locomotrice	E	100
Vagone merci	E	250
Vagone passeggeri	E	550
Tratta	E	25
Viaggio	E	9.000
Fermata	E	40
Lavoratore	E	1150
Turno	E	23.000
Posto	E	1.000.000
Biglietti	E	960.000
Merce	E	24.000
Azienda	E	2.000
Report di manutenzione	E	45.000
Composizione treno	R	900
Guidare	R	100
Manutenzione	R	45.000
Trasporto	R	24.000
Provenienza	R	24.000
Direzione	R	24.000
Composizione vagone	R	1.000.000
Prenotazione	R	960.000
Merce-viaggio	R	24.000
Biglietto-viaggio	R	960.000
Utilizzato	R	1.000.000
Percorrenza	R	9.000
Composizione tratta	R	150
Fermare	R	54.000
Assegnazione treno	R	9.000
Circolazione	R	100
Turno-treno	R	23.000
Svolgimento alternativo	R	2.000
Assegnazione turno	R	23.000

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
OP1	Inserimento di un nuovo treno	1/anno
OP2	Assegnazione di un treno a una tratta	100/mese
OP3	Inserimento di un viaggio	300/giorno
OP4	Inserimento di un lavoratore	100/anno

¹ Indicare con E le entità, con R le relazioni

OP5	Inserimento dei nuovi turni per tutti i lavoratori	1/mese
OP6	Sostituzione di un turno	500/settimana
OP7	Resoconto turni settimanali per un lavoratore	1.000/giorno
OP8	Acquisto di un biglietto	30.000/giorno
OP9	Inserimento merce trasportata in un viaggio	100/giorno
OP10	Controllo biglietti	30.000/giorno
OP11	Inserimento report di manutenzione	900/mese

Costo delle operazioni

- OP1: 1 accesso in scrittura all'entità treno, 9 accessi in scrittura alla relazione composizione treno (ogni treno è composto in media da 9 veicoli, una locomotrice e otto vagoni), 9 accessi in scrittura all'entità veicolo, 1 accesso in scrittura alla relazione guidare.
Totale: $2 \cdot (1+9+9+1) \cdot f_1 = 40$ accessi l'anno
- OP2: 1 accesso in lettura all'entità treno, un accesso in scrittura alla relazione circolazione, un accesso in lettura all'entità tratta.
Totale: $(1 + 2 \cdot 1 + 1) \cdot f_2 = 3.600$ accessi l'anno
- OP3: 1 accesso in lettura all'entità tratta, 6 accessi in lettura alla relazione composizione tratta (ogni tratta ha in media 6 fermate), 4 accessi in lettura alla relazione circolazione (ad ogni tratta sono assegnati in media 4 treni), 1 accesso in scrittura all'entità viaggio, 1 accesso in scrittura alla relazione percorrenza, 1 accesso in scrittura alla relazione assegnazione treno, 6 accessi in scrittura alla relazione fermare. Vanno anche inseriti i posti per quel viaggio, operazione che comporta in media 8 accessi in lettura alla relazione composizione treno e 8 all'entità vagone e 20 accessi in scrittura all'entità posto per ogni vagone.
Totale: $(1+6+4+2 \cdot (1+1+1+6)+8+8+2 \cdot 8 \cdot 20) \cdot f_3 = 39.420.000$ accessi l'anno (8.695 accessi al giorno)
- OP4: 1 accesso in scrittura all'entità lavoratore.
Totale: $2 \cdot 1 \cdot f_4 = 200$ accessi l'anno
- OP5: 1150 accessi in lettura all'entità lavoratore, per ogni lavoratore 20 accessi in scrittura alla relazione assegnazione turno, 20 all'entità turno e 20 alla relazione turno-treno.
Totale: $(1150 + 2 \cdot 1150 \cdot (20+20+20)) \cdot f_5 = 1.669.800$ accessi l'anno (4.640 accessi al giorno)
- OP6: 1 accesso in lettura all'entità turno, 1 accesso in scrittura alla relazione assegnazione turno, a questo punto serve identificare un lavoratore che non abbia turni in quella data. Per far questo bisogna accedere a tutte le occorrenze della relazione assegnazione turno relative a quel lavoratore, se troviamo che il lavoratore è libero abbiamo finito, altrimenti dobbiamo passare al lavoratore successivo. La probabilità che il lavoratore sia libero è 1/3, quindi in teoria ogni 60 accessi dovremmo individuare un lavoratore libero. A questo punto va acceduto in lettura all'entità lavoratore e in scrittura alla relazione svolgimento alternativo.
Totale: $(1+2 \cdot 1+60+1+2 \cdot 1) \cdot f_6 = 1.584.000$ accessi l'anno (4.400 accessi al giorno)
- OP7: 5 accessi in lettura all'entità turno, 5 alla relazione turno-treno, 5 all'entità treno.
Totale: $15 \cdot f_7 = 4.320.000$ accessi l'anno (12.000 accessi al giorno)
- OP8: 1 accesso in lettura all'entità viaggio, 1 accesso in lettura alla relazione assegnazione treno, 1 accesso in lettura all'entità treno, a questo punto va cercato un posto libero sul treno. Abbiamo bisogno di accedere ai vagoni per trovarne uno della classe giusta. Per far questo servono accessi in lettura: se il posto libero fosse l'ultimo a cui accedessimo, avremmo 8 accessi alla relazione composizione treno, 8 accessi ai vagoni, 20 accessi per vagone alla relazione composizione vagone e 20 accessi per vagone all'entità posto per un totale di 336

accessi. Non avendo un modo per stimare la probabilità che il posto sia libero od occupato in modo affidabile supponiamo che in media servano la metà di questi accessi, quindi 168. A questo punto servono 1 accesso in scrittura a posto e 1 all'entità biglietto, 1 alla relazione biglietto-viaggio, 1 a posto-viaggio e 1 alla relazione prenotazione.

Totale: $(1+1+1+168+2*(1+1+1+1))*f_8 = 1.954.800.000$ accessi l'anno (5.430.000 accessi al giorno)

OP9: Per prima cosa va associata la merce al viaggio, questo richiede 1 accesso in lettura all'entità viaggio, 1 alla relazione assegnazione treno e 1 a treno, necessario per vedere se è un treno merci o passeggeri. Poi va trovato un vagone libero, quindi 8 accessi in lettura alla relazione composizione treno e 8 all'entità vagone merci. A questo punto possiamo inserire la merce con 1 accesso in scrittura a merce, 1 alla relazione trasporto, 1 alla relazione provenienza, 1 a direzione e 1 a merce-viaggio.

Totale: $(1+1+1+8+8+2*(1+1+1+1))*f_9 = 972.000$ accessi l'anno (2.700 accessi al giorno)

OP10: 1 a biglietto-viaggio e 1 a viaggio poi 1 accesso in lettura a prenotazione e 1 a posto.

Totale: $(1+1+1+1)*f_{10} = 43.200.000$ accessi l'anno (120.000 accessi al giorno)

OP11: 1 accesso in lettura a veicolo, 1 in scrittura a manutenzione e 1 in scrittura a report di manutenzione.

Totale: $(1+2*(1+1))*f_{11} = 54.000$ accessi l'anno (150 accessi al giorno)

Ristrutturazione dello schema E-R

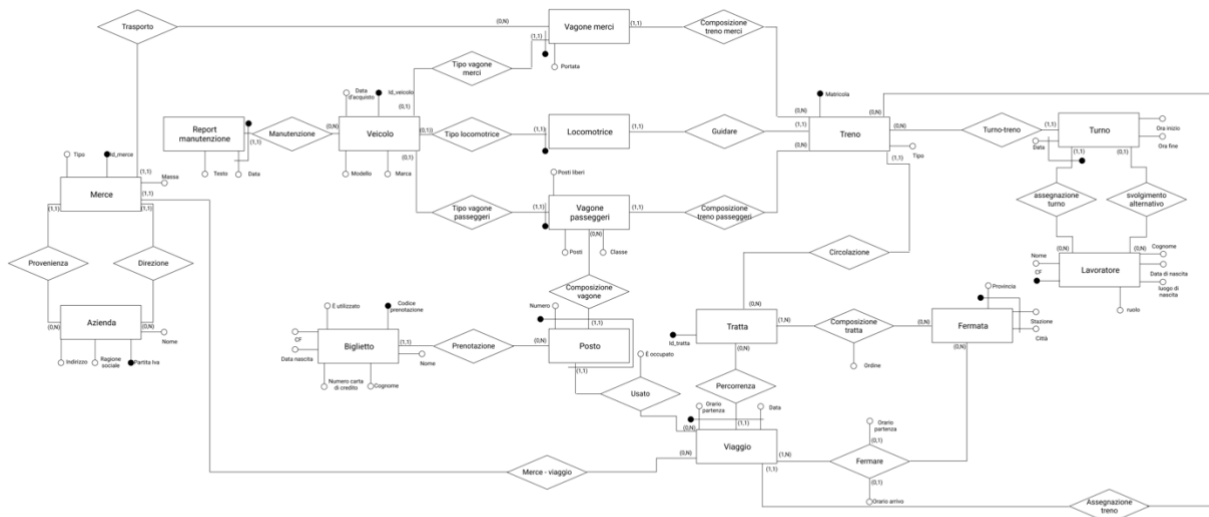
I tre attributi “numero vagoni”, “vagoni prima classe” e “vagoni seconda classe”, tutti e tre relativi all'entità treno, sono facilmente ricavabili dalla relazione composizione treno, e nessuna delle operazioni principali ne fa uso, quindi mantenerli risulta inutile, per questo motivo sono stati rimossi. L'attributo “posti” sull'entità vagoni passeggeri per quanto sembra ricavabile dalla relazione composizione vagone, in realtà è necessario, dato che altrimenti se un vagone non risultasse usato in nessun viaggio perderemmo l'informazione di quanti sono i suoi posti.

L'attributo “numero treni” legato a tratta è facilmente ricavabile grazie alla relazione tra treno e tratta e non viene usato direttamente da nessuna operazione, quindi nuovamente ha senso toglierlo.

L'attributo “coperto” legato alla relazione di assegnazione turno è ricavabile vedendo se il turno partecipa a una relazione di svolgimento alternativo. Togliere questo attributo fa risparmiare circa 150 accessi al giorno nell'operazione di sostituzione turno (OP6) e non influisce su altre delle operazioni principali, quindi va tolto.

La relazione Biglietto-viaggio risulta ridondante dato che è ricavabile da posto, quindi può essere eliminata, viene usata principalmente nel controllo dei biglietti che senza questa relazione mantiene lo stesso costo, e nella prenotazione di un biglietto, per cui togliendo questa relazione risparmiamo 60.000 accessi al giorno.

L'unica generalizzazione dello schema è quella di veicolo in vagone merci, vagone passeggeri e locomotrice. Le tre opzioni sarebbero avere un'unica identità veicolo che raggruppa tutto, avere tre entità separate oppure legare veicolo alle altre tre tramite relazioni. Ho scelto l'ultima opzione perché i tre veicoli hanno relazioni e attributi diversi tra loro, renderli un'unica entità darebbe luogo a molti attributi opzionali e molte relazioni che dipendono dal valore tipo di veicolo aumentando a dismisura le regole aziendali, mentre eliminare del tutto la loro origine comune mi creerebbe problemi per l'entità report di manutenzione, che perderebbe la sua chiave. Ho però deciso di sdoppiare la relazione composizione in *composizione_treno_merci*, che collega un treno a un vagone merci, e *composizione_treno_passeggeri*, che collega ai vagoni passeggeri, per evitare che locomotrice e treno siano legati tra di loro due volte, con spreco di memoria e rischio di inconsistenze.



Trasformazione di attributi e identificatori

L'entità biglietto ha molti attributi, quelli relativi ai dati anagrafici del passeggero, con lo stesso nome rispetto ad attributi dell'entità lavoratore, quindi ho deciso di rinominare gli attributi CF, nome, cognome, data di nascita di biglietto come CF passeggero, nome passeggero, cognome passeggero, data di nascita passeggero. Per lo stesso motivo ho rinominato l'attributo nome dell'entità azienda come nome azienda.

Traduzione di entità e associazioni

TRENO(Matricola, Tipo, Tratta), con vincolo di integrità referenziale tra l'attributo tratta e la relazione TRATTA

VEICOLO(Id_veicolo, Marca, Modello, Data d'acquisto)

LOCOMOTRICE(Id_locomotrice, Treno), con vincolo di integrità referenziale tra l'attributo Id_locomotrice e VEICOLO e tra Treno e la relazione TRENO

VAGONE PASSEGERI(Id_vagone_passeggeri, Treno, Classe, Posti), con vincolo di integrità referenziale tra l'attributo Id_vagone_passeggeri e VEICOLO e tra Treno e la relazione TRENO

VAGONE MERCI(Id_vagone_merci, Treno, Portata), con vincolo di integrità referenziale tra l'attributo Id_vagone_merci e VEICOLO e tra Treno e la relazione TRENO

REPORT DI MANUTENZIONE(Id_veicolo, Data, Testo) con vincolo di integrità referenziale tra l'attributo Id_veicolo e la relazione VEICOLO

TRATTA(Id_tratta)

VIAGGIO(Tratta, Data, Orario partenza, Treno) con vincolo di integrità referenziale tra l'attributo Tratta e la relazione TRATTA e tra l'attributo Treno e la relazione TRENO

FERMATA(Provincia, Città, Stazione)

COMPOSIZIONE_TRATTA(Tratta, Provincia, Città, Stazione, Ordine) con vincolo di integrità referenziale tra l'attributo Tratta e la relazione TRATTA e tra gli attributi (Provincia, Città, Stazione) e la relazione FERMATA

FERMARE(Tratta, DataViaggio, OraViaggio, Provincia, Città, Stazione, Orario di partenza*, Orario di arrivo*) con vincolo di integrità referenziale tra gli attributi (Tratta, DataViaggio, OraViaggio) e la relazione VIAGGIO e tra gli attributi (Provincia, Città, Stazione) e la relazione FERMATA

LAVORATORE(CF, Nome, Cognome, Data di nascita, Luogo di nascita, Ruolo)

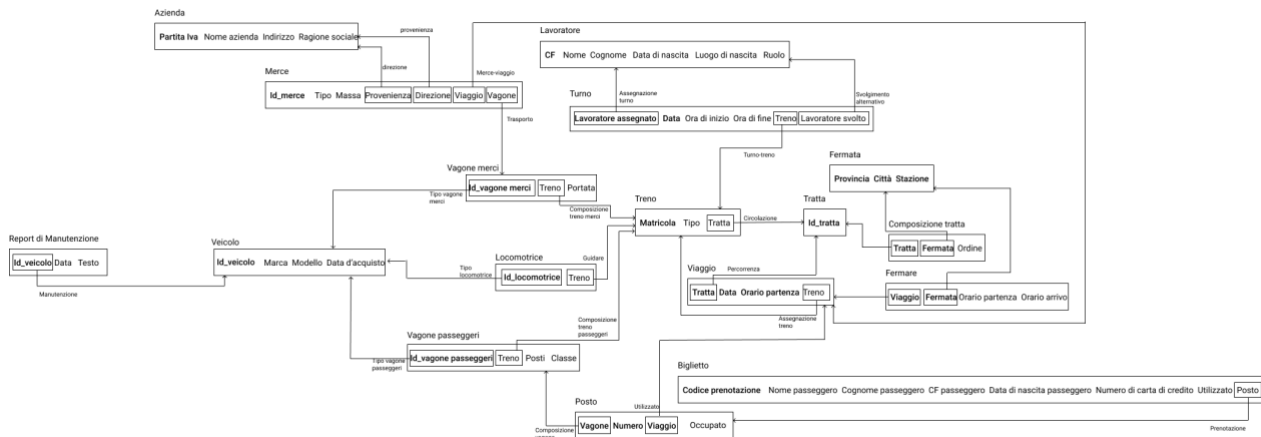
TURNO(Lavoratore assegnato, Data, Ora di inizio, Ora di fine, Treno, Lavoratore svolto*) con vincolo di integrità referenziale tra gli attributi Lavoratore assegnato e Lavoratore svolto e la relazione LAVORATORE e tra l'attributo Treno e la relazione TRENO

BIGLIETTO(Codice prenotazione, Nome passeggero, Cognome passeggero, CF passeggero, Data di nascita passeggero, Numero di carta di credito, Utilizzato, NumeroPosto, VagonePosto, Tratta, DataViaggio, OraViaggio) con vincoli di integrità referenziale tra gli attributi (NumeroPosto, VagonePosto, Tratta, DataViaggio, OraViaggio) e la relazione POSTO e tra l'attributo Viaggio e la relazione VIAGGIO

POSTO(Vagone, Numero, Tratta, DataViaggio, OraViaggio, Occupato) con vincolo di integrità referenziale tra l'attributo Vagone e la relazione VAGONE PASSEGGERI e tra gli attributi (Tratta, DataViaggio, OraViaggio) e la relazione VIAGGIO

AZIENDA(Partita IVA, Nome azienda, Indirizzo, Ragione Sociale)

MERCE(Id_merce, Tipo, Massa, Vagone, Tratta, DataViaggio, OraViaggio, Provenienza, Direzione) con vincoli di integrità referenziale tra l'attributo Vagone e la relazione VAGONE MERCI, gli attributi (Tratta, DataViaggio, OraViaggio) e la relazione VIAGGIO, gli attributi Provenienza e Direzione e la relazione AZIENDA



Normalizzazione del modello relazionale

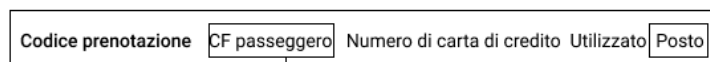
La relazione Biglietto viola la 3NF, infatti presenta le seguenti dipendenze funzionali:

CF passeggero → nome passeggero, cognome passeggero, data di nascita passeggero

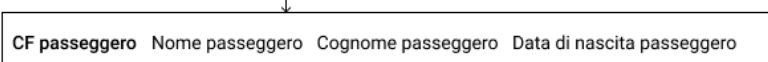
Codice prenotazione → CF passeggero, numero carta di credito, viaggio, posto

In particolare, la prima di queste dipendenze non comprende una chiave primaria evidenziando il fatto che l'entità va divisa.

Biglietto



Passeggero



5. Progettazione fisica

Utenti e privilegi

Ho previsto 5 tipi di utenti: amministratori, lavoratori, capotreni e addetti alla manutenzione, che hanno username e password tramite cui si identificano, e passeggeri per cui non è necessario autorizzarsi. Gli utenti effettivamente creati sono stati sei, dato che ho creato un utente login con come unico privilegio l'execute sulla procedura login, dato che prima di loggare un utente deve avere la possibilità comunque di accedere al database per poter verificare nome e password.

Effettivamente anche l'utente passeggero non è loggato, ho preferito mantenerlo comunque separato dall'utente login anche se non era strettamente necessario. I privilegi sono tutti riferiti all'execute di procedure, nessuno di questi utenti ha privilegi diretti di accesso a tabelle oppure ha la possibilità di modificare lo schema del database.

Login: login

Passeggero: trova_viaggi, prenota_biglietto

Macchinista: visualizza_turni

Capotreno: visualizza_turni, controllo_biglietti

Manutentore: insert_report

Amministratore: insert_treno_merci, insert_treno_passeggeri, insert_lavoratore, insert_utente, assign_train, insert_turno, sostituisci_turno, num_fermate, insert_viaggio, insert_merce.

Strutture di memorizzazione

Tabella Treno		
Attributo	Tipo di dato	Attributi ²
Matricola	Integer	PK, NN, UN
Tipo	Bool	NN
Tratta	Integer	UN

Tabella Tratta		
Attributo	Tipo di dato	Attributi
Id_tratta	Integer	PK, NN, UN, AI

Tabella Fermata		
Attributo	Tipo di dato	Attributi
Provincia	Varchar(20)	PK, NN
Città	Varchar(20)	PK, NN
Stazione	Varchar(30)	PK, NN

Tabella Viaggio		
Attributo	Tipo di dato	Attributi
Tratta	Integer	PK, NN, UN
DataPartenza	Date	PK, NN
OraPartenza	Time	PK, NN
Treno	Integer	NN, UN

Tabella Composizione_tratta		
Attributo	Tipo di dato	Attributi

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

Tratta	Integer	PK, NN, UN, UQ
ProvinciaFermata	Varchar(20)	PK, NN
CittàFermata	Varchar(20)	PK, NN
StazioneFermata	Varchar(30)	PK, NN
Ordine	Smallint	NN, UN, UQ

Tabella Fermare		
Attributo	Tipo di dato	Attributi
Tratta	Integer	PK, NN, UN, UQ
DataViaggio	Date	PK, NN, UQ
OraViaggio	Time	PK, NN
ProvinciaFermata	Varchar(20)	PK, NN, UQ
CittàFermata	Varchar(20)	PK, NN, UQ
StazioneFermata	Varchar(20)	PK, NN, UQ
OraPartenza	Time	UQ
OraArrivo	Time	

Tabella Lavoratore		
Attributo	Tipo di dato	Attributi
CF	Character(17)	NN, PK
Nome	Varchar(20)	NN
Cognome	Varchar(20)	NN
DataNascita	Date	
LuogoNascita	Varchar(30)	
Ruolo	Bool	NN

Tabella Turno		
Attributo	Tipo di dato	Attributi
LavoratoreAssegnato	Character(17)	PK, NN
DataTurno	Date	PK, NN
OraInizio	Time	NN
OraFine	Time	NN
Treno	Integer	NN, UN
LavoratoreSvolto	Character(17)	

Tabella Veicolo		
Attributo	Tipo di dato	Attributi
Id_veicolo	Integer	PK, NN, UN, AI
Marca	Varchar(20)	NN
Modello	Varchar(20)	NN
DataAcquisto	Date	NN

Tabella Locomotrice		
Attributo	Tipo di dato	Attributi
Id_locomotrice	Integer	PK, NN, UN
Treno	Integer	UN, UQ

Tabella Vagone_merci		
----------------------	--	--

Attributo	Tipo di dato	Attributi
Id_vagone_merci	Integer	PK, NN, UN
Portata	Smallint	NN, UN
Treno	Integer	UN

Tabella Vagone_passeggeri		
Attributo	Tipo di dato	Attributi
Id_vagone_passeggeri	Integer	PK, NN, UN
Posti	Smallint	NN, UN
Classe	Smallint	NN, UN
Treno	Integer	UN

Tabella Report_manutenzione		
Attributo	Tipo di dato	Attributi
Id_veicolo	Integer	PK, NN, UN
DataManutenzione	Date	PK, NN
Testo	Text	NN

Tabella Azienda		
Attributo	Tipo di dato	Attributi
PartitaIva	Character(12)	PK, NN
Nome	Varchar(20)	NN
Tipo	Varchar(4)	NN
Indirizzo	Varchar(50)	NN

Tabella Merce		
Attributo	Tipo di dato	Attributi
Id_merce	Integer	PK, NN, UN, AI
Tipo	Varchar(30)	NN
Massa	Integer	NN, UN
Provenienza	Character(12)	
Direzione	Character(12)	
Vagone	Integer	UN
Viaggio	Integer	UN
DataViaggio	Date	
OraViaggio	Time	

Tabella Posto		
Attributo	Tipo di dato	Attributi
Vagone	Integer	PK, NN, UN
Numero	Smallint	PK, NN, UN
Tratta	Integer	PK, NN, UN
DataViaggio	Date	PK, NN
OraViaggio	Time	PK, NN
Occupato	Bool	NN

Tabella Passeggero		
Attributo	Tipo di dato	Attributi

CF_passeggero	Character(17)	PK, NN
Nome_passeggero	Varchar(20)	NN
Cognome_passeggero	Varchar(20)	NN
Data_nascita_passeggero	Date	NN

Tabella Biglietto		
Attributo	Tipo di dato	Attributi
Codice_prenotazione	Integer	PK, NN, UN, AI
CF_passeggero	Varchar(17)	NN
NumeroCartaCredito	Varchar(17)	NN
Utilizzato	Bool	NN
VagonePosto	Integer	NN, UN, UQ
NumeroPosto	Smallint	NN, UN, UQ
TrattaViaggio	Integer	NN, UN, UQ
DataViaggio	Date	NN, UQ
OraViaggio	Time	NN, UQ

Indici

Per quanto riguarda gli indici ce n'è uno per ogni primary key, che è l'attributo più usato per le ricerche, e uno per ogni foreign key, usate nelle operazioni di join. Ho aggiunto in più due indici, sulla colonna DataPartenza di Viaggio e la colonna DataTurno di Turno, dato che le operazioni di delete per queste due tabelle vengono fatte usando queste colonne come chiave di ricerca. Ho aggiunto anche un indice sulla tabella Biglietto che comprendono le colonne TrattaViaggio, DataViaggio, OraViaggio, primary key per Viaggio, dato che ho usato in più di un'occasione il join tra queste due tabelle ma questi attributi non sono foreign key per Biglietto

Tabella <Azienda>	
Indice <Primary>	Tipo ³ :
PartitaIva	<PR>
Tabella <Biglietto>	
Indice <Primary>	Tipo:
Codice_prenotazione	<PR>
Indice <FK_BigliettoPasseggero>	Tipo:
CF_passeggero	<IDX>
Indice <BigliettoViaggio>	Tipo:
TrattaViaggio	<IDX>
DataViaggio	<IDX>
OraViaggio	<IDX>
Indice <FK_BigliettoPosto>	Tipo:
VagonePosto	<IDX>
NumeroPosto	<IDX>
TrattaViaggio	<IDX>
DataViaggio	<IDX>
OraViaggio	<IDX>
Tabella <Composizione_tratta>	
Indice <Primary>	Tipo:
Tratta	<PR>

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

ProvinciaFermata	<PR>
CittaFermata	<PR>
StazioneFermata	<PR>
Indice <FK_TrattaComposizione>	Tipo:
Tratta	<IDX>
Indice <FK_FermataComposizione>	Tipo:
ProvinciaFermata	<IDX>
CittaFermata	<IDX>
StazioneFermata	<IDX>
Tabella <Fermare>	
Indice <Primary>	Tipo:
Tratta	<PR>
DataViaggio	<PR>
OraViaggio	<PR>
ProvinciaFermata	<PR>
CittaFermata	<PR>
StazioneFermata	<PR>
Indice <FK_FermareViaggio>	Tipo:
Tratta	<IDX>
DataViaggio	<IDX>
OraViaggio	<IDX>
Indice <FK_FermareFermata>	Tipo:
ProvinciaFermata	<IDX>
CittaFermata	<IDX>
StazioneFermata	<IDX>
Tabella <Fermata>	
Indice <Primary>	Tipo:
Provincia	<PR>
Citta	<PR>
Stazione	<PR>
Tabella <Lavoratore>	
Indice <Primary>	Tipo:
CF	<PR>
Tabella <Locomotrice>	
Indice <Primary>	Tipo:
Id_locomotrice	<PR>
Indice <Treno>	Tipo:
Treno	<UQ>
Tabella <Merce>	
Indice <Primary>	Tipo:
Id_merce	<PR>
Indice <FK_ViaggioMerce>	Tipo:
Viaggio	<IDX>
DataViaggio	<IDX>
OraViaggio	<IDX>
Indice <FK_AziendaMerce1>	Tipo:
Provenienza	<IDX>
Indice <FK_AziendaMerce2>	Tipo:

Direzione	<IDX>
Indice <FK_VagoneMerce>	Tipo:
Vagone	<IDX>
Tabella <Passeggero>	
Indice <Primary>	Tipo:
CF_passeggero	<PR>
Tabella <Posto>	
Indice <Primary>	Tipo:
Vagone	<PR>
Numero	<PR>
Tratta	<PR>
DataViaggio	<PR>
OraViaggio	<PR>
Indice <Vagone>	Tipo:
Vagone	<IDX>
Indice <FK_PostoViaggio>	Tipo:
Tratta	<IDX>
DataViaggio	<IDX>
OraViaggio	<IDX>
Tabella <Report_di_manutenzione>	
Indice <Primary>	Tipo:
Id_veicolo	<PR>
DataManutenzione	<PR>
Tabella <Tratta>	
Indice <Primary>	Tipo:
Id_tratta	<PR>
Tabella <Treno>	
Indice <Primary>	Tipo:
Matricola	<PR>
Indice <Tratta>	Tipo:
Tratta	<IDX>
Tabella <Turno>	
Indice <Primary>	Tipo:
LavoratoreAssegnato	<PR>
DataTurno	<PR>
Indice <FK_TurnoTreno>	Tipo:
Treno	<IDX>
Indice <FK_LavoratoreTurno1>	Tipo:
LavoratoreAssegnato	<IDX>
Indice <FK_LavoratoreTurno2>	Tipo:
LavoratoreSvolto	<IDX>
Indice <Data>	Tipo:
DataTurno	<IDX>
Tabella <Vagone_merci>	
Indice <Primary>	Tipo:
Id_vagone_merci	<PR>
Indice <FK_VagoneMTreno>	Tipo:
Treno	<IDX>

Tabella <Vagone_passeggeri>	
Indice <Primary>	Tipo:
Id_vagone_passeggeri	<PR>
Indice <FK_VagonePTreno>	Tipo:
Treno	<IDX>
Tabella <Veicolo>	
Indice <Primary>	Tipo:
Id_veicolo	<PR>
Tabella <Viaggio>	
Indice <Primary>	Tipo:
Tratta	<PR>
DataPartenza	<PR>
OraPartenza	<PR>
Indice <FK_ViaggioTreno>	Tipo:
Treno	<IDX>
Indice <Data>	Tipo:
DataPartenza	<IDX>

Trigger

Trigger che implementa la regola “i capotreni lavorano solo su treni passeggeri”. Sia il tipo di treno che il ruolo del lavoratore, potendo assumere solo due valori, sono memorizzati sotto forma di bool, dove per il tipo 0 rappresenta “merci” e 1 “passeggeri”, per il ruolo 0 rappresenta “macchinista” e 1 “capotreno”. Questo trigger controlla gli inserimenti dei turni, se il turno deve essere svolto da un capotreno si assicura che il treno sia di tipo passeggeri.

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

```
    DECLARE tipotreno, ruololavoratore BOOL;
    SELECT Treno.Tipo FROM Treno
    WHERE Treno.Matricola = NEW.Treno INTO tipotreno;
    SELECT Lavoratore.Ruolo FROM Lavoratore
    WHERE Lavoratore.CF = NEW.LavoratoreAssegnato INTO ruololavoratore;
    IF ruololavoratore = 1 AND tipotreno = 0 THEN
        SIGNAL SQLSTATE '45000';
    END IF;
```

```
END
```

Sempre per la stessa regola aziendale va anche creato un trigger che effettui lo stesso controllo quando viene modificato un turno per inserire lo svolgimento effettuato da un diverso lavoratore:

```
CREATE DEFINER = CURRENT_USER TRIGGER `Ferrovie`.`Turno_BEFORE_UPDATE`
BEFORE UPDATE ON `Turno` FOR EACH ROW
BEGIN
```

```
    DECLARE tipotreno, ruololavoratore BOOL;
    SELECT Tipo FROM Treno
    WHERE Treno.Matricola = NEW.Treno INTO tipotreno;
    IF tipotreno = 0 THEN
        SELECT Ruolo FROM Lavoratore
        WHERE Lavoratore.CF = NEW.LavoratoreSvolto INTO ruololavoratore;
        IF ruololavoratore = 1 THEN
```

```
        SIGNAL SQLSTATE '45000';
    END IF;
END IF;
END
```

La successiva regola aziendale usa nuovamente il tipo di treno: controlla nelle fasi di insert e update di un vagone che il campo Treno di un vagone merci contenga solo treni di tipo merci, quindi 0, e che il campo Treno di un vagone passeggeri contenga solo treni di tipo passeggeri (1)

```
CREATE DEFINER = CURRENT_USER TRIGGER
`Ferrovie`.`Vagone_merci_BEFORE_INSERT` BEFORE INSERT ON `Vagone_merci` FOR
EACH ROW
BEGIN
    DECLARE tipotreno BOOL;
    SELECT Tipo FROM Treno
    WHERE Treno.Matricola = NEW.Treno INTO tipotreno;
    IF tipotreno <> 0 THEN
        SIGNAL SQLSTATE '45000';
    END IF;
END
```

```
CREATE DEFINER = CURRENT_USER TRIGGER
`Ferrovie`.`Vagone_merci_BEFORE_UPDATE` BEFORE UPDATE ON `Vagone_merci` FOR
EACH ROW
BEGIN
    DECLARE tipotreno BOOL;
    SELECT Tipo FROM Treno
    WHERE Treno.Matricola = NEW.Treno INTO tipotreno;
    IF tipotreno <> 0 THEN
        SIGNAL SQLSTATE '45000';
    END IF;
END
```

```
CREATE DEFINER = CURRENT_USER TRIGGER
`Ferrovie`.`Vagone_passeggeri_BEFORE_INSERT` BEFORE INSERT ON `Vagone_passeggeri`
FOR EACH ROW
BEGIN
    DECLARE tipotreno BOOL;
    SELECT Tipo FROM Treno
    WHERE Treno.Matricola = NEW.Treno INTO tipotreno;
    IF tipotreno <> 1 THEN
        SIGNAL SQLSTATE '45000';
    END IF;
END
```

```
CREATE DEFINER = CURRENT_USER TRIGGER
`Ferrovie`.`Vagone_passeggeri_BEFORE_UPDATE` BEFORE UPDATE ON `Vagone_passeggeri`
FOR EACH ROW
BEGIN
    DECLARE tipotreno BOOL;
    SELECT Tipo FROM Treno
    WHERE Treno.Matricola = NEW.Treno INTO tipotreno;
```

```

IF tipotreno <> 1 THEN
    SIGNAL SQLSTATE '45000';
END IF;

```

END

Il successivo trigger serve a controllare che un turno non duri più di quattro ore:

```

CREATE DEFINER = CURRENT_USER TRIGGER
`Ferrovie`.`Turno_INTERVAL` BEFORE INSERT ON `Turno` FOR EACH ROW
BEGIN
    DECLARE timeinterval TIME;
    SET timeinterval = TIMEDIFF( NEW.OraFine, NEW.OraInizio);
    IF timeinterval > '04:00:00' THEN
        SIGNAL SQLSTATE '45000';
    END IF;

```

END

Questo trigger serve a controllare che nella fase di inserimento di un viaggio la fermata per cui vengono inseriti gli orari appartenga effettivamente alla tratta seguita.

```

CREATE DEFINER = CURRENT_USER TRIGGER `Ferrovie`.`Fermare_BEFORE_INSERT`
BEFORE INSERT ON `Fermare` FOR EACH ROW
BEGIN
    IF (NEW.ProvinciaFermata, NEW.CittaFermata, NEW.StazioneFermata) NOT IN
        (SELECT ProvinciaFermata, CittaFermata, StazioneFermata
        FROM Composizione_tratta WHERE Composizione_tratta.Tratta = NEW.Tratta)
    THEN
        SIGNAL SQLSTATE '45000';
    END IF;

```

END

Il trigger seguente serve a controllare che ogni lavoratore non abbia più di cinque turni a settimana, per implementarlo ho ritenuto utile creare una vista che memorizzasse per ogni data la settimana corrispondente e usarla per semplificare la select nel trigger.

```

CREATE DEFINER = CURRENT_USER TRIGGER `Ferrovie`.`Turno_FIVE_PER_WEEK`
BEFORE INSERT ON `Turno` FOR EACH ROW
BEGIN
    DECLARE nperweek INT;
    SELECT COUNT(*) FROM week_turno
    WHERE week_turno.LavoratoreAssegnato = NEW.LavoratoreAssegnato and
    week_turno.Settimana = WEEK(NEW.DataTurno, 0) INTO nperweek;
    IF nperweek > 4 THEN
        SIGNAL SQLSTATE '45000';
    END IF;

```

END

L'ultimo trigger serve a controllare che non venga caricata su un vagone merce superiore alla sua portata:

```

CREATE DEFINER = CURRENT_USER TRIGGER `Ferrovie`.`Merce_BEFORE_INSERT`
BEFORE INSERT ON `Merce` FOR EACH ROW
BEGIN
    DECLARE portatavagone SMALLINT;
    SELECT Portata FROM Vagone_merci
    WHERE Vagone_merci.Id_vagone_merci = NEW.Vagone INTO portatavagone;
    IF portatavagone < NEW.Massa THEN
        SIGNAL SQLSTATE '45000';

```

```

        END IF;
    END
    Trigger che si assicura che ogni viaggio sia effettuato da un treno associato con la tratta che il
    viaggio segue:
    CREATE
    DEFINER=`root`@`localhost`
    TRIGGER `Ferrovie`.`Viaggio_BEFORE_INSERT`
    BEFORE INSERT ON `Ferrovie`.`Viaggio`
    FOR EACH ROW
    BEGIN
        IF NEW.Treno NOT IN (SELECT Treno.Matricola
                                FROM Treno
                                WHERE Treno.Tratta = NEW.Tratta) THEN
            SIGNAL SQLSTATE '45000';
        END IF;
    END
END

```

Eventi

Ho creato un evento che si attiva periodicamente per cancellare i viaggi passati, dato che una volta che il viaggio è stato svolto non abbiamo più informazioni di interesse legate a quel viaggio.

```

CREATE EVENT IF NOT EXISTS `Viaggio_cleanup`
ON SCHEDULE EVERY 1 DAY
ON COMPLETION PRESERVE
DO BEGIN
    delete from Viaggio WHERE DataPartenza < DATE(NOW());
END

```

Sempre per effettuare cancellazioni periodiche ho creato un evento che il primo di ogni mese svuota la tabella dei turni, dato che ogni mese ne vengono inseriti di nuovi.

```

CREATE EVENT IF NOT EXISTS `Turno_cleanup`
ON SCHEDULE EVERY 1 MONTH STARTS '2021-01-01 00:00:00'
ON COMPLETION PRESERVE
DO BEGIN
    DELETE FROM Turno WHERE DataTurno < DATE(NOW());
END

```

Viste

Vista che mostra i turni che un lavoratore deve svolgere effettivamente, quindi i turni per il quale un lavoratore non viene sostituito e le sostituzioni che deve effettuare, e in quale settimana il turno viene svolto. È necessaria sia per verificare che un lavoratore non svolga più di cinque turni a settimana.

```

CREATE VIEW `week_turno` AS
    SELECT `turno`.`LavoratoreAssegnato` AS `LavoratoreAssegnato`, `turno`.`DataTurno` AS
    `dataTurno`, WEEK(`turno`.`DataTurno`, 0) AS `Settimana`
    FROM `turno`
    WHERE (`turno`.`LavoratoreSvolto` IS NULL)
UNION
    SELECT `turno`.`LavoratoreSvolto` AS `LavoratoreAssegnato`, `turno`.`DataTurno` AS
    `DataTurno`, WEEK(`turno`.`DataTurno`, 0) AS `Settimana`
    FROM `turno`
    WHERE (`turno`.`LavoratoreSvolto` IS NOT NULL)

```

Vista che memorizza per ogni data e ora quale vagone merci è occupato.

```
CREATE VIEW `occupazione_vagoni_merci` AS
  SELECT `merce`.`DataViaggio` AS `Data`, `merce`.`OraViaggio` AS `Ora`,
  `merce`.`Vagone` AS `Vagone`
  FROM `merce`
```

Stored Procedures e transazioni

Procedure per effettuare il login, dati username e password ritorna il ruolo corrispondente:

```
CREATE PROCEDURE `login`(in var_username varchar(20), in var_pass varchar(20), out var_role
INT)
BEGIN
  DECLARE var_user_role ENUM('AMMINISTRATORE', 'MACCHINISTA',
  'CAPOTRENO', 'MANUTENTORE');
  SELECT `Ruolo`
  FROM `Utenti`
  WHERE `Username` = var_username AND `U_password` = var_pass INTO var_user_role;
  IF var_user_role = 'AMMINISTRATORE' THEN
    SET var_role = 1;
  ELSEIF var_user_role = 'MACCHINISTA' THEN
    SET var_role = 2;
  ELSEIF var_user_role = 'CAPOTRENO' THEN
    SET var_role = 3;
  ELSEIF var_user_role = 'MANUTENTORE' THEN
    SET var_role = 4;
  ELSE
    SET var_role = 5;
  END IF;
END
```

Procedura per visualizzare i possibili viaggi tra due città in un determinato giorno, assicurandosi che siano viaggi svolti da treni passeggeri in modo da poter far scegliere per quale viaggio prenotare un biglietto:

```
CREATE PROCEDURE `trova_viaggio`(IN var_provincia_par varchar(20), IN var_partenza
varchar(20), IN var_provincia_arr varchar(20), IN var_arrivo varchar(20), IN var_data date)
BEGIN
  SELECT DISTINCT Partenza.Tratta, Partenza.DataViaggio, Partenza.OraPartenza,
  Partenza.StazioneFermata as Partenza, Arrivo.OraArrivo, Arrivo.StazioneFermata as Arrivo
  FROM Fermare as Partenza
  JOIN Fermare AS Arrivo ON (Partenza.Tratta = Arrivo.Tratta AND Partenza.DataViaggio =
  Arrivo.DataViaggio AND Partenza.OraViaggio = Arrivo.OraViaggio)
  JOIN Viaggio AS v ON (v.Tratta = Partenza.tratta AND v.DataPartenza =
  Partenza.DataViaggio)
  WHERE (Partenza.ProvinciaFermata = var_provincia_par AND Partenza.CittaFermata =
  var_partenza AND Arrivo.ProvinciaFermata = var_provincia_arr AND Arrivo.CittaFermata
  = var_arrivo AND var_data = v.DataPartenza AND Partenza.OraPartenza <
  Arrivo.OraArrivo AND v.Treno IN (SELECT Matricola
  FROM Treno
  WHERE Tipo = 1));
END
```

Procedura per prenotare un biglietto, segnando il posto come occupato e decrementando il numero di posti liberi del vagone, e poi aggiungendo i dati del passeggero se non presenti e aggiungendo un

nuovo biglietto, di cui ritorna il codice di prenotazione. Il livello di isolamento è repeatable read per evitare che tra la select e l'update di vagone e posto quel posto venga occupato da un'altra procedura:

```
CREATE PROCEDURE `prenota_biglietto`(IN var_tratta INT, IN var_data DATE, IN
var_prov_fermata VARCHAR(20), IN var_citta_fermata VARCHAR(20), IN var_staz_fermata
VARCHAR(30), IN var_ora TIME, IN var_classe SMALLINT,
IN var_cf CHARACTER(17), IN var_nome VARCHAR(20), IN var_cognome VARCHAR(20), IN
var_nascita DATE, IN var_num CHARACTER(17), OUT var_codice INT)
BEGIN
    DECLARE var_posto, var_vagone INT;
    DECLARE var_ora_viaggio TIME;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK; -- rollback any changes made in the transaction
        RESIGNAL; -- raise again the sql exception to the caller
    END;
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    START TRANSACTION;
    SELECT Viaggio.OraPartenza
    FROM Viaggio JOIN Fermare ON (Viaggio.Tratta = Fermare.Tratta AND
    Viaggio.DataPartenza = Fermare.DataViaggio AND Fermare.OraViaggio =
    Viaggio.OraPartenza)
    WHERE Viaggio.Tratta = var_tratta AND Viaggio.DataPartenza = var_data AND
    Fermare.ProvinciaFermata = var_prov_fermata AND Fermare.CittaFermata =
    var_citta_fermata AND Fermare.StazioneFermata = var_staz_fermata AND
    Fermare.OraPartenza = var_ora
    INTO var_ora_viaggio;
    SELECT V.Id_vagone_passeggeri FROM Posto JOIN Vagone_passeggeri AS V ON
    Posto.Vagone = V.Id_vagone_passeggeri
    WHERE Posto.Occupato = 0 AND V.classe = var_classe AND Posto.Tratta =
    var_tratta AND Posto.DataViaggio = var_data AND Posto.OraViaggio =
    var_ora_viaggio
    LIMIT 1 INTO var_vagone;
    IF var_vagone IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non ci sono posti
        liberi';
    END IF;
    SELECT Numero FROM Posto
    WHERE Posto.Vagone = var_vagone AND Posto.Tratta = var_tratta AND
    Posto.DataViaggio = var_data AND Posto.OraViaggio = var_ora_viaggio AND
    Posto.Occupato = 0
    LIMIT 1 INTO var_posto;
    UPDATE Posto SET Occupato = 1 WHERE Vagone = var_vagone AND Numero =
    var_posto and Tratta = var_tratta AND DataViaggio = var_data AND OraViaggio =
    var_ora_viaggio;
    INSERT IGNORE INTO Passeggero values (var_cf, var_nome, var_cognome,
    var_nascita);
    INSERT INTO Biglietto (CF_passeggero, NumeroCartaCredito, VagonePosto,
    NumeroPosto, TrattaViaggio, DataViaggio, OraViaggio) VALUES (var_cf, var_num,
    var_vagone, var_posto, var_tratta, var_data, var_ora_viaggio);
    SELECT LAST_INSERT_ID() INTO var_codice;
```



```
COMMIT;
```

```
END
```

Procedura per inserire un treno merci, con i relativi vagoni e locomotrice. Ho preferito fare due procedure diverse per l'inserimento di treni merci e passeggeri dato che i parametri dei vagoni tra le due procedure sono differenti. La funzione split serve a dividere una stringa formata da interi separati dal simbolo '\$' necessaria per passare alla procedure la portata di ogni vagone senza sapere a priori il numero di vagoni. Il livello di isolamento è read uncommitted dato che la procedure si occupa solo di inserimenti senza effettuare letture.

```
CREATE PROCEDURE `insert_treno_merci`(IN Var_matricola INTEGER, IN Var_marca
VARCHAR(20), IN Var_modello VARCHAR(20), IN var_num_vagoni INT, IN var_portata
VARCHAR(150))
```

```
BEGIN
```

```
    DECLARE count INT default 0;
```

```
    DECLARE p INT;
```

```
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
```

```
    BEGIN
```

```
        ROLLBACK; -- rollback any changes made in the transaction
```

```
        RESIGNAL; -- raise again the sql exception to the caller
```

```
    END;
```

```
    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
    START TRANSACTION;
```

```
        INSERT INTO Treno (Matricola, Tipo) values (Var_matricola, 0);
```

```
        INSERT INTO Veicolo (Marca, Modello, DataAcquisto) values (Var_marca,
Var_modello, DATE(NOW()));
```

```
        INSERT INTO Locomotrice values (LAST_INSERT_ID(), Var_matricola);
```

```
        WHILE count < var_num_vagoni DO
```

```
            SET p = SPLIT(var_portata, count);
```

```
            INSERT INTO Veicolo (Marca, Modello, DataAcquisto) values (Var_marca,
Var_modello, DATE(NOW()));
```

```
            INSERT INTO Vagone_merci VALUES (LAST_INSERT_ID(), p,
Var_matricola);
```

```
            SET count = count + 1;
```

```
        END WHILE;
```

```
    COMMIT;
```

```
END
```

Funzione split:

```
CREATE FUNCTION `SPLIT`(portata VARCHAR(500), count INT) RETURNS int
```

```
NO SQL
```

```
BEGIN
```

```
    DECLARE i INT DEFAULT 0;
```

```
    DECLARE start INT DEFAULT 0;
```

```
    DECLARE end INT ;
```

```
    SET end = LOCATE('$', portata);
```

```
    WHILE i < count DO
```

```
        SET start = end;
```

```
        SET end = LOCATE('$', portata, start+1);
```

```
        SET i = i+1;
```

```
    END WHILE;
```

```
    SET start = start+1;
```

```
    SET end = end-start;
```

```
RETURN CONVERT(SUBSTRING(portata, start, end), UNSIGNED INT);
```

```
END
```

Procedura per inserire un treno passeggeri, simile a quella relativa a un treno merci ma oltre ad inserire i vagoni per ogni vagone vanno inseriti i relativi posti.

```
CREATE PROCEDURE `insert_treno_passeggeri`(IN Var_matricola INTEGER, IN Var_marca
VARCHAR(20), IN Var_modello VARCHAR(20), IN var_num_vagoni INT, IN var_posti
VARCHAR(50), IN var_classe VARCHAR(50))
```

```
BEGIN
```

```
    DECLARE count INT default 0;
```

```
    DECLARE p INT;
```

```
    DECLARE c INT;
```

```
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
```

```
    BEGIN
```

```
        ROLLBACK; -- rollback any changes made in the transaction
```

```
        RESIGNAL; -- raise again the sql exception to the caller
```

```
    END;
```

```
    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
```

```
    START TRANSACTION;
```

```
        INSERT INTO Treno (Matricola, Tipo) values (Var_matricola, 1);
```

```
        INSERT INTO Veicolo (Marca, Modello, DataAcquisto) values (Var_marca,
Var_modello, DATE(NOW()));
```

```
        INSERT INTO Locomotrice values (LAST_INSERT_ID(), Var_matricola);
```

```
        WHILE count < var_num_vagoni DO
```

```
            SET p = SPLIT(var_posti, count);
```

```
            SET c = SPLIT(var_classe, count);
```

```
            INSERT INTO Veicolo (Marca, Modello, DataAcquisto) values (Var_marca,
Var_modello, DATE(NOW()));
```

```
            INSERT INTO Vagone_passeggeri VALUES (LAST_INSERT_ID(), p, c,
Var_matricola, p);
```

```
            SET count = count + 1;
```

```
        END WHILE;
```

```
    COMMIT;
```

```
END
```

Procedura per modificare l'assegnazione di un treno a una tratta:

```
CREATE PROCEDURE `assign_train`(IN var_tratta INT, IN var_treno INT)
```

```
BEGIN
```

```
    UPDATE Treno SET Tratta = var_tratta WHERE Matricola = var_treno;
```

```
END
```

Procedura per inserire un nuovo viaggio con le relative fermate, dopo averle prese in ordine a partire dalla tratta. La funzione SPLIT_STRING segue la stessa logica di SPLIT ma ritorna una stringa piuttosto che un intero. Il livello di isolamento necessario per essere sicuri che la procedura arrivi al risultato corretto è serializable dato che dipende da modifiche e insert in molte tabelle.

```
CREATE PROCEDURE `insert_viaggio`(IN var_tratta INT, IN var_data DATE, IN var_treno INT,
IN var_oraPartenza VARCHAR(150), IN var_oraArrivo VARCHAR(150))
```

```
BEGIN
```

```
    DECLARE done INT DEFAULT False;
```

```
    DECLARE count INT DEFAULT 0;
```

```
    DECLARE num_fermate INT;
```

```
    DECLARE ora, p, a TIME;
```

```
    DECLARE prov, citt VARCHAR(20);
```

```
DECLARE staz VARCHAR(30);
DECLARE var_posti, var_vagone INT;
DECLARE cur CURSOR FOR SELECT Id_vagone_passeggeri FROM Vagone_passeggeri
WHERE Treno = var_treno;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = true;
DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
    ROLLBACK; -- rollback any changes made in the transaction
    RESIGNAL; -- raise again the sql exception to the caller
END;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
    SELECT COUNT(*) FROM Composizione_tratta WHERE Tratta = var_tratta INTO
    num_fermate;
    SET ora = SPLIT_STRING(var_oraPartenza, 0);
    INSERT INTO Viaggio values (var_tratta, var_data, ora, var_treno);
    WHILE count < num_fermate DO
        IF count <> num_fermate -1 THEN
            SET p = SPLIT_STRING(var_oraPartenza, count);
        ELSE
            SET p = NULL;
        END IF;
        IF count <> 0 THEN
            SET a = SPLIT_STRING(var_oraArrivo, count);
        ELSE
            SET a = NULL;
        END IF;
        SELECT ProvinciaFermata FROM Composizione_tratta WHERE Tratta =
        var_tratta AND Ordine = count+1 INTO prov;
        SELECT CittaFermata FROM Composizione_tratta WHERE Tratta =
        var_tratta AND Ordine = count+1 INTO citt;
        SELECT StazioneFermata FROM Composizione_tratta WHERE Tratta =
        var_tratta AND Ordine = count+1 INTO staz;
        INSERT INTO Fermate VALUES (var_tratta,var_data,ora,prov, citt, staz, p,
        a);
        SET count = count +1;
    END WHILE;
    IF (SELECT Tipo FROM Treno WHERE Matricola = var_treno) = 1 THEN
        OPEN cur;
        insert_posti: LOOP
            FETCH cur INTO var_vagone;
            IF done THEN
                LEAVE insert_posti;
            END IF;
            SET count = 0;
            SELECT Posti FROM Vagone_passeggeri WHERE
            Id_vagone_passeggeri = var_vagone INTO var_posti;
            WHILE count < var_posti DO
                INSERT INTO Posto values (var_vagone, count+1, 0,
                var_tratta,var_data, ora);
```

```

        SET count = count + 1;
    END WHILE;
END LOOP;
END IF;
COMMIT;
END

```

Procedura per l'inserimento di un nuovo lavoratore:

```

CREATE PROCEDURE `insert_lavoratore`(IN var_cf CHARACTER(17), IN var_nome
VARCHAR(20), IN var_cognome VARCHAR(20), IN var_data DATE, IN var_luogo
VARCHAR(30), IN var_ruolo BOOL)
BEGIN
    INSERT INTO Lavoratore VALUES (var_cf, var_nome, var_cognome, var_data, var_luogo,
var_ruolo);
END

```

Procedura per inserire un nuovo turno:

```

CREATE PROCEDURE `insert_turno`(IN var_lavoratore CHARACTER(17), IN var_data DATE,
IN var_inizio TIME, IN var_fine TIME, IN var_treno INT)
BEGIN
    INSERT INTO Turno (LavoratoreAssegnato, DataTurno, OraInizio, OraFine, Treno)
VALUES (var_lavoratore, var_data, var_inizio, var_fine, var_treno);
END

```

Procedura per effettuare la sostituzione di un turno, in cui il lavoratore libero viene trovato automaticamente dal sistema. Il livello di isolamento è serializable per evitare che tra la select del lavoratore e l'update del turno vengano assegnati altri turni al lavoratore selezionato portandolo a superare i cinque turni settimanali:

```

CREATE PROCEDURE `sostituisci_turno`(IN var_lavoratore CHARACTER(17), IN var_data
DATE)
BEGIN
    DECLARE var_new CHARACTER(17);
    DECLARE var_tipo BOOL;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    START TRANSACTION;
    SELECT Ruolo FROM Lavoratore WHERE CF = var_lavoratore INTO var_tipo;
    SELECT Lavoratore.Cf FROM Lavoratore
    WHERE Lavoratore.Ruolo = var_tipo AND CF not in (SELECT
LavoratoreAssegnato
FROM week_turno)
LIMIT 1 INTO var_new;
    IF var_new IS NULL THEN
        SELECT T.LavoratoreAssegnato
        FROM week_turno AS T JOIN Lavoratore AS L on L.CF =
T.LavoratoreAssegnato
        WHERE var_data <> T.DataTurno AND L.Ruolo = var_tipo AND L.CF <>
var_lavoratore AND T.Settimana <> WEEK(var_data)LIMIT 1 INTO
var_new;
    END IF;
END

```

```

END IF;
IF var_new IS NULL THEN
    SELECT T.LavoratoreAssegnato
    FROM week_turno as T JOIN Lavoratore as L on L.CF =
    T.LavoratoreAssegnato
    WHERE var_data <> T.DataTurno AND T.Settimana = WEEK(var_data)
    AND L.Ruolo = var_tipo AND L.CF <> var_lavoratore
    GROUP BY T.LavoratoreAssegnato
    HAVING COUNT(T.Settimana) < 5 LIMIT 1 INTO var_new;
END IF;
UPDATE Turno SET LavoratoreSvolto = var_new
WHERE LavoratoreAssegnato = var_lavoratore AND DataTurno = var_data;
COMMIT;
END

```

Procedura per permettere a un impiegato di visualizzare i propri turni per la settimana corrente:

```

CREATE PROCEDURE `visualizza_turni`(IN var_CF CHARACTER(17))
BEGIN
    SELECT DataTurno as Data, OraInizio, OraFine, Treno
    FROM Turno WHERE (LavoratoreAssegnato = var_CF OR LavoratoreSvolto = var_CF)
    AND WEEK(DataTurno, 0) = WEEK(NOW(), 0);
END

```

Procedura per inserire una nuova merce, assegnandole un viaggio e selezionando un vagone adatto a trasportarla per portata e che sia libero. Il livello di isolamento è serializable perché un diverso inserimento di una nuova merce potrebbe occupare il vagone selezionato prima che venga usato dalla procedure.

```

CREATE PROCEDURE `insert_merce`(IN var_tipo VARCHAR(30), IN var_massa INT, IN
var_prov CHAR(12), IN var_arr CHAR(12), IN var_tratta INT, IN var_data DATE, IN var_ora
TIME)
BEGIN
    DECLARE idvagone INT;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;
    SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
    START TRANSACTION;
    SELECT Id_vagone_merci FROM Vagone_merci as Vagone JOIN Treno on
    Vagone.Treno = Treno.Matricola JOIN Viaggio ON Treno.Matricola = Viaggio.Treno
    WHERE Viaggio.Tratta = var_tratta AND Viaggio.DataPartenza = var_data AND
    Viaggio.OraPartenza = var_ora AND Vagone.Portata <= var_massa AND
    id_vagone_merci NOT IN (SELECT Vagone
        FROM occupazione_vagoni_merci
        WHERE Data = var_data AND Ora = var_ora) LIMIT 1
    INTO idvagone;
    INSERT INTO Merce (Tipo, Massa, Vagone, Provenienza, Direzione, Viaggio,
    DataViaggio, OraViaggio) VALUES (var_tipo, var_massa, idvagone, var_prov,
    var_arr, var_tratta, var_data, var_ora);
    COMMIT;
END

```

Procedura per il controllo dei biglietti, stampa le informazioni relative al viaggio e al posto, poi verifica che il biglietto non sia già stato utilizzato e in caso affermativo lo segna come utilizzato. Il livello di isolamento è repeatable read per far sì che al momento dell'update il valore di Utilizzato trovato precedentemente non sia cambiato.

```
CREATE PROCEDURE `controllo_biglietti`(IN var_codice INT, OUT var_valido TINYINT(1))
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;
    SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    START TRANSACTION;
        SELECT CF_passeggero AS Passeggero, TrattaViaggio, DataViaggio AS Data,
            VagonePosto AS Vagone, NumeroPosto AS Posto
        FROM Biglietto
        WHERE Codice_prenotazione = var_codice;
        SELECT Utilizzato FROM Biglietto WHERE Codice_prenotazione = var_codice
        INTO var_valido;
        UPDATE Biglietto SET Utilizzato = 1 WHERE Codice_prenotazione = var_codice;
    COMMIT;
END;
```

Procedura per l'inserimento di un nuovo report di manutenzione per un determinato veicolo:

```
CREATE PROCEDURE `insert_report`(IN var_testo varchar(2048), IN var_id INT)
BEGIN
    INSERT INTO Report_di_manutenzione VALUES (var_id, DATE(NOW()), var_testo);
END
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';
```

```
-----
-- Schema Ferrovie
-----
```

```
CREATE SCHEMA IF NOT EXISTS `Ferrovie` DEFAULT CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci ;
USE `Ferrovie` ;
```

```
-----
-- Table `Ferrovie`.`Azienda`
-----
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Azienda` (
  `PartitaIva` CHAR(12) NOT NULL,
  `Nome` VARCHAR(20) NOT NULL,
  `Tipo` VARCHAR(4) NOT NULL,
  `Indirizzo` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`PartitaIva`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-----
-- Table `Ferrovie`.`Passeggero`
-----
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Passeggero` (
  `CF_passeggero` CHAR(17) NOT NULL,
  `Nome_passeggero` VARCHAR(20) NOT NULL,
  `Cognome_passeggero` VARCHAR(20) NOT NULL,
  `Data_nascita_passeggero` DATE NULL DEFAULT NULL,
  PRIMARY KEY (`CF_passeggero`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-----
-- Table `Ferrovie`.`Veicolo`
-----
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Veicolo` (
  `Id_veicolo` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Marca` VARCHAR(20) NOT NULL,
```

```
`Modello` VARCHAR(20) NOT NULL,
`DataAcquisto` DATE NOT NULL,
PRIMARY KEY (`Id_veicolo`))
ENGINE = InnoDB
AUTO_INCREMENT = 14
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `Ferrovie`.`Tratta`
-----
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Tratta` (
  `Id_tratta` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`Id_tratta`))
ENGINE = InnoDB
AUTO_INCREMENT = 301
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `Ferrovie`.`Treno`
-----
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Treno` (
  `Matricola` INT UNSIGNED NOT NULL,
  `Tipo` TINYINT NOT NULL,
  `Tratta` INT UNSIGNED NULL DEFAULT NULL,
  PRIMARY KEY (`Matricola`),
  INDEX `Tratta` (`Tratta` ASC) VISIBLE,
  CONSTRAINT ``
  FOREIGN KEY (`Tratta`)
  REFERENCES `Ferrovie`.`Tratta` (`Id_tratta`)
  ON DELETE SET NULL)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `Ferrovie`.`Vagone_passeggeri`
-----
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Vagone_passeggeri` (
  `Id_vagone_passeggeri` INT UNSIGNED NOT NULL,
  `Classe` SMALLINT UNSIGNED NOT NULL,
  `Treno` INT UNSIGNED NULL DEFAULT NULL,
  `Posti` SMALLINT UNSIGNED NOT NULL,
  PRIMARY KEY (`Id_vagone_passeggeri`),
  INDEX `FK_VagonePTreno` (`Treno` ASC) VISIBLE,
  CONSTRAINT `FK_VagonePasseggeriVeicolo`
  FOREIGN KEY (`Id_vagone_passeggeri`)
  REFERENCES `Ferrovie`.`Veicolo` (`Id_veicolo`)
  ON DELETE CASCADE,
```



```
CONSTRAINT `FK_VagonePTreno`  
FOREIGN KEY (`Treno`)  
REFERENCES `Ferrovie`.`Treno` (`Matricola`)  
ON DELETE SET NULL)  
ENGINE = InnoDB  
DEFAULT CHARACTER SET = utf8mb4  
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----  
-- Table `Ferrovie`.`Viaggio`  
-- -----
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Viaggio` (  
  `Tratta` INT UNSIGNED NOT NULL,  
  `DataPartenza` DATE NOT NULL,  
  `OraPartenza` TIME NOT NULL,  
  `Treno` INT UNSIGNED NOT NULL,  
  PRIMARY KEY (`Tratta`, `DataPartenza`, `OraPartenza`),  
  INDEX `FK_ViaggioTreno` (`Treno` ASC) VISIBLE,  
  INDEX `Data` (`DataPartenza` ASC) VISIBLE,  
  CONSTRAINT `FK_ViaggioTratta`  
  FOREIGN KEY (`Tratta`)  
  REFERENCES `Ferrovie`.`Tratta` (`Id_tratta`)  
  ON DELETE CASCADE,  
  CONSTRAINT `FK_ViaggioTreno`  
  FOREIGN KEY (`Treno`)  
  REFERENCES `Ferrovie`.`Treno` (`Matricola`)  
  ON DELETE CASCADE)  
ENGINE = InnoDB  
DEFAULT CHARACTER SET = utf8mb4  
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----  
-- Table `Ferrovie`.`Posto`  
-- -----
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Posto` (  
  `Vagone` INT UNSIGNED NOT NULL,  
  `Numero` SMALLINT UNSIGNED NOT NULL,  
  `Occupato` TINYINT NOT NULL DEFAULT '0',  
  `Tratta` INT UNSIGNED NOT NULL,  
  `DataViaggio` DATE NOT NULL,  
  `OraViaggio` TIME NOT NULL,  
  PRIMARY KEY (`Vagone`, `Numero`, `Tratta`, `DataViaggio`, `OraViaggio`),  
  INDEX `Vagone` (`Vagone` ASC) VISIBLE,  
  INDEX `FK_PostoViaggio_idx` (`Tratta` ASC, `DataViaggio` ASC, `OraViaggio` ASC)  
  VISIBLE,  
  CONSTRAINT `FK_PostoVagone`  
  FOREIGN KEY (`Vagone`)  
  REFERENCES `Ferrovie`.`Vagone_passeggeri` (`Id_vagone_passeggeri`)  
  ON DELETE CASCADE,  
  CONSTRAINT `FK_PostoViaggio`
```

```

FOREIGN KEY (`Tratta`, `DataViaggio`, `OraViaggio`)
REFERENCES `Ferrovie`.`Viaggio` (`Tratta`, `DataPartenza`, `OraPartenza`)
ON DELETE CASCADE)

```

```
ENGINE = InnoDB
```

```
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;
```

```

-----
-- Table `Ferrovie`.`Biglietto`
-----

```

```

CREATE TABLE IF NOT EXISTS `Ferrovie`.`Biglietto` (
  `Codice_prenotazione` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `CF_passeggero` CHAR(17) NOT NULL,
  `NumeroCartaCredito` CHAR(17) NOT NULL,
  `Utilizzato` TINYINT NOT NULL DEFAULT '0',
  `VagonePosto` INT UNSIGNED NOT NULL,
  `NumeroPosto` SMALLINT UNSIGNED NOT NULL,
  `TrattaViaggio` INT UNSIGNED NOT NULL,
  `DataViaggio` DATE NOT NULL,
  `OraViaggio` TIME NOT NULL,
  PRIMARY KEY (`Codice_prenotazione`),
  INDEX `FK_BigliettoPasseggero_idx` (`CF_passeggero` ASC) VISIBLE,
  INDEX `BigliettoViaggio` (`TrattaViaggio` ASC, `DataViaggio` ASC, `OraViaggio` ASC)
  VISIBLE,
  INDEX `FK_BigliettoPosto` (`VagonePosto` ASC, `NumeroPosto` ASC, `TrattaViaggio`
  ASC, `DataViaggio` ASC, `OraViaggio` ASC) VISIBLE,
  CONSTRAINT `FK_BigliettoPasseggero`
  FOREIGN KEY (`CF_passeggero`)
  REFERENCES `Ferrovie`.`Passeggero` (`CF_passeggero`),
  CONSTRAINT `FK_BigliettoPosto`
  FOREIGN KEY (`VagonePosto`, `NumeroPosto`, `TrattaViaggio`, `DataViaggio`,
  `OraViaggio`)
  REFERENCES `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Tratta`, `DataViaggio`,
  `OraViaggio`)
  ON DELETE CASCADE)

```

```
ENGINE = InnoDB
```

```
AUTO_INCREMENT = 17
```

```
DEFAULT CHARACTER SET = utf8mb4
```

```
COLLATE = utf8mb4_0900_ai_ci;
```

```

-----
-- Table `Ferrovie`.`Fermata`
-----

```

```

CREATE TABLE IF NOT EXISTS `Ferrovie`.`Fermata` (
  `Provincia` VARCHAR(20) NOT NULL,
  `Citta` VARCHAR(20) NOT NULL,
  `Stazione` VARCHAR(30) NOT NULL,
  PRIMARY KEY (`Provincia`, `Citta`, `Stazione`))

```

```
ENGINE = InnoDB
```

```
DEFAULT CHARACTER SET = utf8mb4
```

COLLATE = utf8mb4_0900_ai_ci;

 -- Table `Ferrovie`.`Composizione_tratta`

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Composizione_tratta` (
  `Tratta` INT UNSIGNED NOT NULL,
  `ProvinciaFermata` VARCHAR(20) NOT NULL,
  `CittaFermata` VARCHAR(20) NOT NULL,
  `StazioneFermata` VARCHAR(30) NOT NULL,
  `Ordine` SMALLINT UNSIGNED NOT NULL,
  PRIMARY KEY (`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `Tratta`),
  INDEX `FK_TrattaComposizione` (`Tratta` ASC) VISIBLE,
  INDEX `FK_FermataComposizione` (`ProvinciaFermata` ASC, `CittaFermata` ASC,
  `StazioneFermata` ASC) VISIBLE,
  CONSTRAINT `FK_FermataComposizione`
  FOREIGN KEY (`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`)
  REFERENCES `Ferrovie`.`Fermata` (`Provincia`, `Citta`, `Stazione`)
  ON DELETE CASCADE,
  CONSTRAINT `FK_TrattaComposizione`
  FOREIGN KEY (`Tratta`)
  REFERENCES `Ferrovie`.`Tratta` (`Id_tratta`)
  ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

 -- Table `Ferrovie`.`Fermare`

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Fermare` (
  `Tratta` INT UNSIGNED NOT NULL,
  `DataViaggio` DATE NOT NULL,
  `OraViaggio` TIME NOT NULL,
  `ProvinciaFermata` VARCHAR(20) NOT NULL,
  `CittaFermata` VARCHAR(20) NOT NULL,
  `StazioneFermata` VARCHAR(30) NOT NULL,
  `OraPartenza` TIME NULL DEFAULT NULL,
  `OraArrivo` TIME NULL DEFAULT NULL,
  PRIMARY KEY (`Tratta`, `DataViaggio`, `OraViaggio`, `ProvinciaFermata`, `CittaFermata`,
  `StazioneFermata`),
  UNIQUE INDEX `Tratta` (`Tratta` ASC, `DataViaggio` ASC, `ProvinciaFermata` ASC,
  `CittaFermata` ASC, `StazioneFermata` ASC, `OraPartenza` ASC) VISIBLE,
  INDEX `FK_FermareViaggio` (`Tratta` ASC, `DataViaggio` ASC, `OraViaggio` ASC)
  VISIBLE,
  INDEX `FK_FermareFermata` (`ProvinciaFermata` ASC, `CittaFermata` ASC,
  `StazioneFermata` ASC) VISIBLE,
  CONSTRAINT `FK_FermareFermata`
  FOREIGN KEY (`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`)
  REFERENCES `Ferrovie`.`Fermata` (`Provincia`, `Citta`, `Stazione`)
```

```
ON DELETE CASCADE,
CONSTRAINT `FK_FermareViaggio`
FOREIGN KEY (`Tratta`, `DataViaggio`, `OraViaggio`)
REFERENCES `Ferrovie`.`Viaggio` (`Tratta`, `DataPartenza`, `OraPartenza`)
ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-----
-- Table `Ferrovie`.`Lavoratore`
-----
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Lavoratore` (
  `CF` CHAR(17) NOT NULL,
  `Nome` VARCHAR(20) NOT NULL,
  `Cognome` VARCHAR(20) NOT NULL,
  `DataNascita` DATE NULL DEFAULT NULL,
  `LuogoNascita` VARCHAR(30) NULL DEFAULT NULL,
  `Ruolo` TINYINT NOT NULL,
  PRIMARY KEY (`CF`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-----
-- Table `Ferrovie`.`Locomotrice`
-----
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Locomotrice` (
  `Id_locomotrice` INT UNSIGNED NOT NULL,
  `Treno` INT UNSIGNED NULL DEFAULT NULL,
  PRIMARY KEY (`Id_locomotrice`),
  UNIQUE INDEX `Treno` (`Treno` ASC) VISIBLE,
  CONSTRAINT `FK_LocomotriceTreno`
  FOREIGN KEY (`Treno`)
  REFERENCES `Ferrovie`.`Treno` (`Matricola`)
  ON DELETE SET NULL,
  CONSTRAINT `FK_LocomotriceVeicolo`
  FOREIGN KEY (`Id_locomotrice`)
  REFERENCES `Ferrovie`.`Veicolo` (`Id_veicolo`)
  ON DELETE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-----
-- Table `Ferrovie`.`Vagone_merci`
-----
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Vagone_merci` (
  `Id_vagone_merci` INT UNSIGNED NOT NULL,
  `Portata` SMALLINT UNSIGNED NOT NULL,
```

```

`Treno` INT UNSIGNED NULL DEFAULT NULL,
PRIMARY KEY (`Id_vagone_merci`),
INDEX `FK_VagoneMTreno` (`Treno` ASC) VISIBLE,
CONSTRAINT `FK_VagoneMerciVeicolo`
FOREIGN KEY (`Id_vagone_merci`)
REFERENCES `Ferrovie`.`Veicolo` (`Id_veicolo`)
ON DELETE CASCADE,
CONSTRAINT `FK_VagoneMTreno`
FOREIGN KEY (`Treno`)
REFERENCES `Ferrovie`.`Treno` (`Matricola`)
ON DELETE SET NULL)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `Ferrovie`.`Merce`
-----
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Merce` (
  `Id_merce` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Tipo` VARCHAR(30) NOT NULL,
  `Massa` INT UNSIGNED NOT NULL,
  `Vagone` INT UNSIGNED NULL DEFAULT NULL,
  `Provenienza` CHAR(12) NULL DEFAULT NULL,
  `Direzione` CHAR(12) NULL DEFAULT NULL,
  `Viaggio` INT UNSIGNED NULL DEFAULT NULL,
  `DataViaggio` DATE NULL DEFAULT NULL,
  `OraViaggio` TIME NULL DEFAULT NULL,
  PRIMARY KEY (`Id_merce`),
  INDEX `FK_ViaggioMerce` (`Viaggio` ASC, `DataViaggio` ASC, `OraViaggio` ASC)
  VISIBLE,
  INDEX `FK_AziendaMerce1_idx` (`Provenienza` ASC) VISIBLE,
  INDEX `FK_AziendaMerce2_idx` (`Direzione` ASC) VISIBLE,
  INDEX `FK_VagoneMerce_idx` (`Vagone` ASC) VISIBLE,
  CONSTRAINT `FK_AziendaMerce1`
  FOREIGN KEY (`Provenienza`)
  REFERENCES `Ferrovie`.`Azienda` (`PartitaIva`)
  ON DELETE SET NULL,
  CONSTRAINT `FK_AziendaMerce2`
  FOREIGN KEY (`Direzione`)
  REFERENCES `Ferrovie`.`Azienda` (`PartitaIva`)
  ON DELETE SET NULL,
  CONSTRAINT `FK_VagoneMerce`
  FOREIGN KEY (`Vagone`)
  REFERENCES `Ferrovie`.`Vagone_merci` (`Id_vagone_merci`)
  ON DELETE SET NULL,
  CONSTRAINT `FK_ViaggioMerce`
  FOREIGN KEY (`Viaggio`, `DataViaggio`, `OraViaggio`)
  REFERENCES `Ferrovie`.`Viaggio` (`Tratta`, `DataPartenza`, `OraPartenza`)
  ON DELETE CASCADE)

```

```
ENGINE = InnoDB
AUTO_INCREMENT = 4
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- Table `Ferrovie`.`Report_di_manutenzione`
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Report_di_manutenzione` (
  `Id_veicolo` INT UNSIGNED NOT NULL,
  `DataManutenzione` DATE NOT NULL,
  `Testo` VARCHAR(2048) NOT NULL,
  PRIMARY KEY (`Id_veicolo`, `DataManutenzione`),
  CONSTRAINT `FK_ReportVeicolo`
  FOREIGN KEY (`Id_veicolo`)
  REFERENCES `Ferrovie`.`Veicolo` (`Id_veicolo`)
  ON DELETE CASCADE)
```

```
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- Table `Ferrovie`.`Turno`
```

```
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Turno` (
  `LavoratoreAssegnato` CHAR(17) NOT NULL,
  `DataTurno` DATE NOT NULL,
  `OraInizio` TIME NOT NULL,
  `OraFine` TIME NOT NULL,
  `Treno` INT UNSIGNED NOT NULL,
  `LavoratoreSvolto` CHAR(17) NULL DEFAULT NULL,
  PRIMARY KEY (`LavoratoreAssegnato`, `DataTurno`),
  INDEX `FK_TurnoTreno` (`Treno` ASC) VISIBLE,
  INDEX `Data` (`DataTurno` ASC) VISIBLE,
  INDEX `FK_LavoratoreTurno1` (`LavoratoreAssegnato` ASC) VISIBLE,
  INDEX `FK_TurnoLavoratore2_idx` (`LavoratoreSvolto` ASC) VISIBLE,
  CONSTRAINT `FK_TurnoLavoratore1`
  FOREIGN KEY (`LavoratoreAssegnato`)
  REFERENCES `Ferrovie`.`Lavoratore` (`CF`)
  ON DELETE CASCADE
  ON UPDATE RESTRICT,
  CONSTRAINT `FK_TurnoLavoratore2`
  FOREIGN KEY (`LavoratoreSvolto`)
  REFERENCES `Ferrovie`.`Lavoratore` (`CF`)
  ON DELETE SET NULL
  ON UPDATE RESTRICT,
  CONSTRAINT `FK_TurnoTreno`
  FOREIGN KEY (`Treno`)
  REFERENCES `Ferrovie`.`Treno` (`Matricola`)
  ON DELETE CASCADE)
```

```
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-----
-- Table `Ferrovie`.`Utenti`
-----
CREATE TABLE IF NOT EXISTS `Ferrovie`.`Utenti` (
  `Username` VARCHAR(20) NOT NULL,
  `U_password` VARCHAR(20) NOT NULL,
  `Ruolo` ENUM('amministratore', 'macchinista', 'capotreno', 'manutentore') NOT NULL,
  PRIMARY KEY (`Username`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

CREATE USER 'LOGIN' IDENTIFIED BY 'login';
CREATE ROLE 'role_login';
GRANT 'role_login' TO 'LOGIN';
SET DEFAULT ROLE 'role_login' TO 'LOGIN';
GRANT EXECUTE ON PROCEDURE login TO 'role_login';

CREATE USER 'PASSEGGERO' IDENTIFIED BY 'passeggero';
CREATE ROLE 'role_passeggero';
GRANT 'role_passeggero' TO 'PASSEGGERO';
SET DEFAULT ROLE 'role_passeggero' TO 'PASSEGGERO';
GRANT EXECUTE ON PROCEDURE trova_viaggio TO 'role_passeggero';
GRANT EXECUTE ON PROCEDURE prenota_biglietto TO 'role_passeggero';

CREATE USER 'MACCHINISTA' IDENTIFIED BY 'macchinista';
CREATE ROLE 'role_macchinista';
GRANT 'role_macchinista' TO 'MACCHINISTA';
SET DEFAULT ROLE 'role_macchinista' TO 'MACCHINISTA';
GRANT EXECUTE ON PROCEDURE visualizza_turni TO 'role_macchinista';

CREATE USER 'CAPOTRENO' IDENTIFIED BY 'capotreno';
CREATE ROLE 'role_capotreno';
GRANT 'role_capotreno' TO 'CAPOTRENO';
SET DEFAULT ROLE 'role_capotreno' TO 'CAPOTRENO';
GRANT EXECUTE ON PROCEDURE visualizza_turni TO 'role_capotreno';
GRANT EXECUTE ON PROCEDURE controllo_biglietti TO 'role_capotreno';

CREATE USER 'MANUTENTORE' IDENTIFIED BY 'manutentore';
CREATE ROLE 'role_manutentore';
GRANT 'role_manutentore' TO 'MANUTENTORE';
SET DEFAULT ROLE 'role_manutentore' TO 'MANUTENTORE';
GRANT EXECUTE ON PROCEDURE insert_report TO 'role_manutentore';

CREATE USER 'AMMINISTRATORE' IDENTIFIED BY 'amministratore';
CREATE ROLE 'role_amministratore';
```

```
GRANT 'role_amministratore' TO 'AMMINISTRATORE';
SET DEFAULT ROLE 'role_amministratore' TO 'AMMINISTRATORE';
GRANT EXECUTE ON PROCEDURE insert_treno_merci TO 'role_amministratore';
GRANT EXECUTE ON PROCEDURE insert_treno_passeggeri TO 'role_amministratore';
GRANT EXECUTE ON PROCEDURE insert_turno TO 'role_amministratore';
GRANT EXECUTE ON PROCEDURE sostituisci_turno TO 'role_amministratore';
GRANT EXECUTE ON PROCEDURE insert_lavoratore TO 'role_amministratore';
GRANT EXECUTE ON PROCEDURE insert_utente TO 'role_amministratore';
GRANT EXECUTE ON PROCEDURE assign_train TO 'role_amministratore';
GRANT EXECUTE ON PROCEDURE insert_merce TO 'role_amministratore';
GRANT EXECUTE ON PROCEDURE insert_viaggio TO 'role_amministratore';
GRANT EXECUTE ON PROCEDURE num_fermate TO 'role_amministratore';
```

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

```
-- -----
-- Data for table `Ferrovie`.`Azienda`
-- -----
```

```
START TRANSACTION;
  USE `Ferrovie`;
  INSERT INTO `Ferrovie`.`Azienda` (`PartitaIva`, `Nome`, `Tipo`, `Indirizzo`) VALUES
    ('00001234567', 'Spedizioni', 'srl', 'Via Roma 120');
  INSERT INTO `Ferrovie`.`Azienda` (`PartitaIva`, `Nome`, `Tipo`, `Indirizzo`) VALUES
    ('00009876543', 'Spedizioni 2.0', 'srl', 'Via Firenze 10');
COMMIT;
```

```
-- -----
-- Data for table `Ferrovie`.`Passeggero`
-- -----
```

```
START TRANSACTION;
  USE `Ferrovie`;
  INSERT INTO `Ferrovie`.`Passeggero` (`CF_passeggero`, `Nome_passeggero`,
    `Cognome_passeggero`, `Data_nascita_passeggero`) VALUES ('FRCVRD78B11H501K',
    'Francesco', 'Verdi', '1978-02-11');
  INSERT INTO `Ferrovie`.`Passeggero` (`CF_passeggero`, `Nome_passeggero`,
    `Cognome_passeggero`, `Data_nascita_passeggero`) VALUES ('CRLGLL89H43H678L',
    'Carla', 'Gialli', '1989-06-03');
COMMIT;
```

```
-- -----
-- Data for table `Ferrovie`.`Veicolo`
-- -----
```

```
START TRANSACTION;
USE `Ferrovie`;
  INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
  VALUES (1, 'Frecciarossa', '1000', '2020-09-23');
```



```

INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (2, 'Frecciarossa', '1000', '2020-09-23');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (3, 'Frecciarossa', '1000', '2020-09-23');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (4, 'Frecciarossa', 'ETR 500', '2016-08-01');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (5, 'Frecciarossa', 'ETR 500', '2016-08-01');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (6, 'Frecciarossa', 'ETR 500', '2016-08-01');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (7, 'Frecciarossa', '1000', '2019-05-05');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (8, 'Frecciarossa', '1000', '2019-05-05');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (9, 'Frecciargento', 'ETR 200', '2005-09-09');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (10, 'Frecciargento', 'ETR 200', '2005-09-09');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (11, 'Frecciargento', '600', '2008-07-09');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (12, 'Frecciargento', '600', '2008-07-09');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (13, 'Frecciargento', '600', '2008-07-09');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (14, 'ADME', '550', '2001-07-16');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (15, 'ADME', '550', '2001-07-16');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (16, 'ADME', '320', '2010-12-06');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (17, 'ADME', '320', '2010-12-06');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (18, 'ADME', '320', '2010-12-06');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (19, 'WTR', 'ETR 400', '2012-11-11');
INSERT INTO `Ferrovie`.`Veicolo` (`Id_veicolo`, `Marca`, `Modello`, `DataAcquisto`)
VALUES (20, 'WTR', 'ETR 400', '2012-11-11');

```

COMMIT;

```

-----
-- Data for table `Ferrovie`.`Tratta`
-----

```

START TRANSACTION;

USE `Ferrovie`;

INSERT INTO `Ferrovie`.`Tratta` (`Id_tratta`) VALUES (100);

INSERT INTO `Ferrovie`.`Tratta` (`Id_tratta`) VALUES (200);

INSERT INTO `Ferrovie`.`Tratta` (`Id_tratta`) VALUES (300);

COMMIT;

```
-----  
-- Data for table `Ferrovie`.`Treno`  
-----
```

```
START TRANSACTION;
```

```
    USE `Ferrovie`;
```

```
    INSERT INTO `Ferrovie`.`Treno` (`Matricola`, `Tipo`, `Tratta`) VALUES (2727, 1, 100);
```

```
    INSERT INTO `Ferrovie`.`Treno` (`Matricola`, `Tipo`, `Tratta`) VALUES (1212, 0, 100);
```

```
    INSERT INTO `Ferrovie`.`Treno` (`Matricola`, `Tipo`, `Tratta`) VALUES (4343, 1, 100);
```

```
    INSERT INTO `Ferrovie`.`Treno` (`Matricola`, `Tipo`, `Tratta`) VALUES (5656, 1, 200);
```

```
    INSERT INTO `Ferrovie`.`Treno` (`Matricola`, `Tipo`, `Tratta`) VALUES (3434, 0, 200);
```

```
    INSERT INTO `Ferrovie`.`Treno` (`Matricola`, `Tipo`, `Tratta`) VALUES (9898, 1, 300);
```

```
    INSERT INTO `Ferrovie`.`Treno` (`Matricola`, `Tipo`, `Tratta`) VALUES (2323, 1, 300);
```

```
    INSERT INTO `Ferrovie`.`Treno` (`Matricola`, `Tipo`, `Tratta`) VALUES (7676, 0, 300);
```

```
COMMIT;
```

```
-----  
-- Data for table `Ferrovie`.`Vagone_passeggeri`  
-----
```

```
START TRANSACTION;
```

```
    USE `Ferrovie`;
```

```
    INSERT INTO `Ferrovie`.`Vagone_passeggeri` (`Id_vagone_passeggeri`, `Classe`, `Treno`,  
    `Posti`) VALUES (2, 1, 2727, 10);
```

```
    INSERT INTO `Ferrovie`.`Vagone_passeggeri` (`Id_vagone_passeggeri`, `Classe`, `Treno`,  
    `Posti`) VALUES (3, 2, 2727, 12);
```

```
    INSERT INTO `Ferrovie`.`Vagone_passeggeri` (`Id_vagone_passeggeri`, `Classe`, `Treno`,  
    `Posti`) VALUES (5, 2, 4343, 10);
```

```
    INSERT INTO `Ferrovie`.`Vagone_passeggeri` (`Id_vagone_passeggeri`, `Classe`, `Treno`,  
    `Posti`) VALUES (6, 1, 4343, 5);
```

```
    INSERT INTO `Ferrovie`.`Vagone_passeggeri` (`Id_vagone_passeggeri`, `Classe`, `Treno`,  
    `Posti`) VALUES (8, 1, 5656, 7);
```

```
    INSERT INTO `Ferrovie`.`Vagone_passeggeri` (`Id_vagone_passeggeri`, `Classe`, `Treno`,  
    `Posti`) VALUES (10, 2, 9898, 10);
```

```
    INSERT INTO `Ferrovie`.`Vagone_passeggeri` (`Id_vagone_passeggeri`, `Classe`, `Treno`,  
    `Posti`) VALUES (12, 2, 2323, 12);
```

```
    INSERT INTO `Ferrovie`.`Vagone_passeggeri` (`Id_vagone_passeggeri`, `Classe`, `Treno`,  
    `Posti`) VALUES (13, 1, 2323, 9);
```

```
COMMIT;
```

```
-----  
-- Data for table `Ferrovie`.`Viaggio`  
-----
```

```
START TRANSACTION;
```

```
USE `Ferrovie`;
```

```
    INSERT INTO `Ferrovie`.`Viaggio` (`Tratta`, `DataPartenza`, `OraPartenza`, `Treno`)  
    VALUES (100, '2021-09-13', '08:00:00', 1212);
```

```
    INSERT INTO `Ferrovie`.`Viaggio` (`Tratta`, `DataPartenza`, `OraPartenza`, `Treno`)  
    VALUES (100, '2021-09-07', '12:00:00', 4343);
```

```
    INSERT INTO `Ferrovie`.`Viaggio` (`Tratta`, `DataPartenza`, `OraPartenza`, `Treno`)  
    VALUES (200, '2021-09-08', '17:00:00', 5656);
```

```

INSERT INTO `Ferrovie`.`Viaggio` (`Tratta`, `DataPartenza`, `OraPartenza`, `Treno`)
VALUES (200, '2021-09-14', '09:00:00', 3434);
INSERT INTO `Ferrovie`.`Viaggio` (`Tratta`, `DataPartenza`, `OraPartenza`, `Treno`)
VALUES (300, '2021-09-15', '13:00:00', 9898);
COMMIT;

```

```

-----
-- Data for table `Ferrovie`.`Posto`
-----

```

```

START TRANSACTION;

```

```

USE `Ferrovie`;
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 1, 1, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 2, 0, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 3, 0, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 4, 0, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 5, 1, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 6, 0, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 7, 0, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 8, 0, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 9, 0, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (5, 10, 0, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (6, 1, 0, 100, '2021-09-07', '12:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (8, 1, 1, 200, '2021-09-08', '17:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (8, 2, 0, 200, '2021-09-08', '17:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (8, 3, 0, 200, '2021-09-08', '17:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (10, 1, 0, 300, '2021_09-15', '13:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (10, 2, 0, 300, '2021_09-15', '13:00:00');
INSERT INTO `Ferrovie`.`Posto` (`Vagone`, `Numero`, `Occupato`, `Tratta`, `DataViaggio`,
`OraViaggio`) VALUES (10, 3, 0, 300, '2021_09-15', '13:00:00');

```

```

COMMIT;

```

```

-----
-- Data for table `Ferrovie`.`Biglietto`
-----

```

```
START TRANSACTION;
```

```
USE `Ferrovie`;
```

```
INSERT INTO `Ferrovie`.`Biglietto` (`Codice_prenotazione`, `CF_passeggero`,  
`NumeroCartaCredito`, `Utilizzato`, `VagonePosto`, `NumeroPosto`, `TrattaViaggio`,  
`DataViaggio`, `OraViaggio`) VALUES (1, 'FRCVRD78B11H501K', '3333444455556666',  
0, 5, 1, 100, '2021-09-07', '12:00:00');
```

```
INSERT INTO `Ferrovie`.`Biglietto` (`Codice_prenotazione`, `CF_passeggero`,  
`NumeroCartaCredito`, `Utilizzato`, `VagonePosto`, `NumeroPosto`, `TrattaViaggio`,  
`DataViaggio`, `OraViaggio`) VALUES (2, 'FRCVRD78B11H501K', '7777666655554444',  
1, 5, 5, 100, '2021-09-07', '12:00:00');
```

```
INSERT INTO `Ferrovie`.`Biglietto` (`Codice_prenotazione`, `CF_passeggero`,  
`NumeroCartaCredito`, `Utilizzato`, `VagonePosto`, `NumeroPosto`, `TrattaViaggio`,  
`DataViaggio`, `OraViaggio`) VALUES (3, 'CRLGLL89H43H678L', '8888444422220000',  
0, 8, 1, 200, '2021-09-08', '17:00:00');
```

```
COMMIT;
```

```
-----  
-- Data for table `Ferrovie`.`Fermata`  
-----
```

```
START TRANSACTION;
```

```
USE `Ferrovie`;
```

```
INSERT INTO `Ferrovie`.`Fermata` (`Provincia`, `Citta`, `Stazione`) VALUES ('Roma',  
'Roma', 'Termini');
```

```
INSERT INTO `Ferrovie`.`Fermata` (`Provincia`, `Citta`, `Stazione`) VALUES ('Firenze',  
'Firenze', 'Rifredi');
```

```
INSERT INTO `Ferrovie`.`Fermata` (`Provincia`, `Citta`, `Stazione`) VALUES ('Milano',  
'Milano', 'Centrale');
```

```
INSERT INTO `Ferrovie`.`Fermata` (`Provincia`, `Citta`, `Stazione`) VALUES ('Milano',  
'Rho', 'Fiera');
```

```
INSERT INTO `Ferrovie`.`Fermata` (`Provincia`, `Citta`, `Stazione`) VALUES ('Venezia',  
'Venezia', 'Santa Lucia');
```

```
INSERT INTO `Ferrovie`.`Fermata` (`Provincia`, `Citta`, `Stazione`) VALUES ('Venezia',  
'Mestre', 'Mestre');
```

```
INSERT INTO `Ferrovie`.`Fermata` (`Provincia`, `Citta`, `Stazione`) VALUES ('Roma',  
'Roma', 'Tiburtina');
```

```
COMMIT;
```

```
-----  
-- Data for table `Ferrovie`.`Composizione_tratta`  
-----
```

```
START TRANSACTION;
```

```
USE `Ferrovie`;
```

```
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,  
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (100, 'Roma', 'Roma', 'Termini', 1);
```

```
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,  
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (100, 'Roma', 'Roma', 'Tiburtina', 2);
```

```
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,  
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (100, 'Firenze', 'Firenze', 'Rifredi', 3);
```

```

INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (100, 'Milano', 'Milano', 'Centrale',
4);
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (200, 'Venezia', 'Venezia', 'Santa
Lucia', 1);
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (200, 'Venezia', 'Mestre', 'Mestre', 2);
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (200, 'Milano', 'Rho', 'Fiera', 3);
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (200, 'Milano', 'Milano', 'Centrale',
4);
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (300, 'Milano', 'Milano', 'Centrale',
1);
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (300, 'Firenze', 'Firenze', 'Rifredi', 2);
INSERT INTO `Ferrovie`.`Composizione_tratta` (`Tratta`, `ProvinciaFermata`,
`CittaFermata`, `StazioneFermata`, `Ordine`) VALUES (300, 'Roma', 'Roma', 'Termini', 3);
COMMIT;

```

```

-----
-- Data for table `Ferrovie`.`Fermare`
-----

```

```

START TRANSACTION;

```

```

USE `Ferrovie`;
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (100, '2021-09-13', '08:00:00', 'Roma', 'Roma', 'Termini', '08:00:00', NULL);
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (100, '2021-09-13', '08:00:00', 'Roma', 'Roma', 'Tiburtina', '08:25:00', '08:15:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (100, '2021-09-13', '08:00:00', 'Firenze', 'Firenze', 'Rifredi', '12:00:00', '11:30:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (100, '2021-09-13', '08:00:00', 'Milano', 'Milano', 'Centrale', NULL, '15:00:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (100, '2021-09-07', '12:00:00', 'Roma', 'Roma', 'Termini', '12:00:00', NULL);
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (100, '2021-09-07', '12:00:00', 'Roma', 'Roma', 'Tiburtina', '12:40:00', '12:25:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (100, '2021-09-07', '12:00:00', 'Firenze', 'Firenze', 'Rifredi', '13:30:00', '13:10:00');

```

```

INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (100, '2021-09-07', '12:00:00', 'Milano', 'Milano', 'Centrale', NULL, '16:00:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (200, '2021-09-08', '17:00:00', 'Venezia', 'Venezia', 'Santa Lucia', '17:00:00',
NULL);
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (200, '2021-09-08', '17:00:00', 'Venezia', 'Mestre', 'Mestre', '17:45:00', '17:30:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (200, '2021-09-08', '17:00:00', 'Milano', 'Rho', 'Fiera', '18:35:00', '18:30:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (200, '2021-09-08', '17:00:00', 'Milano', 'Milano', 'Centrale', NULL, '19:00:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (200, '2021-09-14', '09:00:00', 'Venezia', 'Venezia', 'Santa Lucia', '09:00:00',
NULL);
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (200, '2021-09-14', '09:00:00', 'Venezia', 'Mestre', 'Mestre', '10:00:00', '09:25:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (200, '2021-09-14', '09:00:00', 'Milano', 'Rho', 'Fiera', '12:15:00', '12:00:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (200, '2021-09-14', '09:00:00', 'Milano', 'Milano', 'Centrale', NULL, '13:00:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (300, '2021-09-15', '13:00:00', 'Milano', 'Milano', 'Centrale', '13:00:00', NULL);
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (300, '2021-09-15', '13:00:00', 'Firenze', 'Firenze', 'Rifredi', '15:00:00', '14:30:00');
INSERT INTO `Ferrovie`.`Fermare` (`Tratta`, `DataViaggio`, `OraViaggio`,
`ProvinciaFermata`, `CittaFermata`, `StazioneFermata`, `OraPartenza`, `OraArrivo`)
VALUES (300, '2021-09-15', '13:00:00', 'Roma', 'Roma', 'Termini', NULL, '16:00:00');
COMMIT;

```

```

-----
-- Data for table `Ferrovie`.`Lavoratore`
-----

```

```

START TRANSACTION;
USE `Ferrovie`;
INSERT INTO `Ferrovie`.`Lavoratore` (`CF`, `Nome`, `Cognome`, `DataNascita`,
`LuogoNascita`, `Ruolo`) VALUES ('DCNSRA99H59H501S', 'Sara', 'Da Canal', '1999-06-
19', 'Roma (RM)', 0);

```

```
INSERT INTO `Ferrovie`.`Lavoratore` (`CF`, `Nome`, `Cognome`, `DataNascita`,
`LuogoNascita`, `Ruolo`) VALUES ('RSSMRO67L02H501R', 'Mario', 'Rossi', '1967-08-02',
'Roma(RM)', 1);
INSERT INTO `Ferrovie`.`Lavoratore` (`CF`, `Nome`, `Cognome`, `DataNascita`,
`LuogoNascita`, `Ruolo`) VALUES ('BNCMRA79M53H501F', 'Maria', 'Bianchi', '1979-09-
13', 'Roma(RM)', 0);
COMMIT;
```

```
-----
-- Data for table `Ferrovie`.`Locomotrice`
-----
```

```
START TRANSACTION;
USE `Ferrovie`;
INSERT INTO `Ferrovie`.`Locomotrice` (`Id_locomotrice`, `Treno`) VALUES (1, 2727);
INSERT INTO `Ferrovie`.`Locomotrice` (`Id_locomotrice`, `Treno`) VALUES (4, 4343);
INSERT INTO `Ferrovie`.`Locomotrice` (`Id_locomotrice`, `Treno`) VALUES (7, 5656);
INSERT INTO `Ferrovie`.`Locomotrice` (`Id_locomotrice`, `Treno`) VALUES (9, 9898);
INSERT INTO `Ferrovie`.`Locomotrice` (`Id_locomotrice`, `Treno`) VALUES (11, 2323);
INSERT INTO `Ferrovie`.`Locomotrice` (`Id_locomotrice`, `Treno`) VALUES (14, 1212);
INSERT INTO `Ferrovie`.`Locomotrice` (`Id_locomotrice`, `Treno`) VALUES (16, 3434);
INSERT INTO `Ferrovie`.`Locomotrice` (`Id_locomotrice`, `Treno`) VALUES (19, 7676);
COMMIT;
```

```
-----
-- Data for table `Ferrovie`.`Vagone_merci`
-----
```

```
START TRANSACTION;
USE `Ferrovie`;
INSERT INTO `Ferrovie`.`Vagone_merci` (`Id_vagone_merci`, `Portata`, `Treno`) VALUES
(15, 300, 1212);
INSERT INTO `Ferrovie`.`Vagone_merci` (`Id_vagone_merci`, `Portata`, `Treno`) VALUES
(17, 4000, 3434);
INSERT INTO `Ferrovie`.`Vagone_merci` (`Id_vagone_merci`, `Portata`, `Treno`) VALUES
(18, 3200, 3434);
INSERT INTO `Ferrovie`.`Vagone_merci` (`Id_vagone_merci`, `Portata`, `Treno`) VALUES
(20, 950, 7676);
COMMIT;
```

```
-----
-- Data for table `Ferrovie`.`Merce`
-----
```

```
START TRANSACTION;
USE `Ferrovie`;
INSERT INTO `Ferrovie`.`Merce` (`Id_merce`, `Tipo`, `Massa`, `Vagone`, `Provenienza`,
`Direzione`, `Viaggio`, `DataViaggio`, `OraViaggio`) VALUES (1, 'Patate', 200, 15,
'00001234567', '00009876543', 100, '2021-09-13', '08:00:00');
INSERT INTO `Ferrovie`.`Merce` (`Id_merce`, `Tipo`, `Massa`, `Vagone`, `Provenienza`,
`Direzione`, `Viaggio`, `DataViaggio`, `OraViaggio`) VALUES (2, 'Pere', 1000, 17,
'00009876543', '00001234567', 200, '2021-09-14', '09:00:00');
COMMIT;
```

```
-----
-- Data for table `Ferrovie`.`Turno`
-----
```

```
START TRANSACTION;
```

```
USE `Ferrovie`;
```

```
INSERT INTO `Ferrovie`.`Turno` (`LavoratoreAssegnato`, `DataTurno`, `OraInizio`,
`OraFine`, `Treno`, `LavoratoreSvolto`) VALUES ('DCNSRA99H59H501S', '2021-09-13',
'08:00:00', '12:00:00', 1212, NULL);
```

```
INSERT INTO `Ferrovie`.`Turno` (`LavoratoreAssegnato`, `DataTurno`, `OraInizio`,
`OraFine`, `Treno`, `LavoratoreSvolto`) VALUES ('DCNSRA99H59H501S', '2021-09-07',
'12:00:00', '16:00:00', 4343, NULL);
```

```
INSERT INTO `Ferrovie`.`Turno` (`LavoratoreAssegnato`, `DataTurno`, `OraInizio`,
`OraFine`, `Treno`, `LavoratoreSvolto`) VALUES ('DCNSRA99H59H501S', '2021-09-08',
'17:00:00', '20:00:00', 5656, NULL);
```

```
INSERT INTO `Ferrovie`.`Turno` (`LavoratoreAssegnato`, `DataTurno`, `OraInizio`,
`OraFine`, `Treno`, `LavoratoreSvolto`) VALUES ('DCNSRA99H59H501S', '2021-09-14',
'09:00:00', '13:00:00', 3434, 'BNCMRA79M53H501F');
```

```
INSERT INTO `Ferrovie`.`Turno` (`LavoratoreAssegnato`, `DataTurno`, `OraInizio`,
`OraFine`, `Treno`, `LavoratoreSvolto`) VALUES ('RSSMRO67L02H501R', '2021-09-07',
'12:00:00', '16:00:00', 4343, NULL);
```

```
INSERT INTO `Ferrovie`.`Turno` (`LavoratoreAssegnato`, `DataTurno`, `OraInizio`,
`OraFine`, `Treno`, `LavoratoreSvolto`) VALUES ('RSSMRO67L02H501R', '2021-09-08',
'17:00:00', '21:00:00', 5656, NULL);
```

```
INSERT INTO `Ferrovie`.`Turno` (`LavoratoreAssegnato`, `DataTurno`, `OraInizio`,
`OraFine`, `Treno`, `LavoratoreSvolto`) VALUES ('RSSMRO67L02H501R', '2021-09-15',
'13:00:00', '16:00:00', 9898, NULL);
```

```
INSERT INTO `Ferrovie`.`Turno` (`LavoratoreAssegnato`, `DataTurno`, `OraInizio`,
`OraFine`, `Treno`, `LavoratoreSvolto`) VALUES ('BNCMRA79M53H501F', '2021-09-15',
'13:00:00', '16:00:00', 9898, NULL);
```

```
COMMIT;
```

```
-----
-- Data for table `Ferrovie`.`Utenti`
-----
```

```
START TRANSACTION;
```

```
USE `Ferrovie`;
```

```
INSERT INTO `Ferrovie`.`Utenti` (`Username`, `U_password`, `Ruolo`) VALUES ('Admin',
'admin', 'amministratore');
```

```
INSERT INTO `Ferrovie`.`Utenti` (`Username`, `U_password`, `Ruolo`) VALUES
('Manutentore', 'admin', 'manutentore');
```

```
INSERT INTO `Ferrovie`.`Utenti` (`Username`, `U_password`, `Ruolo`) VALUES
('DCNSRA99H59H501S', 'admin', 'macchinista');
```

```
INSERT INTO `Ferrovie`.`Utenti` (`Username`, `U_password`, `Ruolo`) VALUES
('RSSMRO67L02H501R', 'admin', 'capotreno');
```

```
INSERT INTO `Ferrovie`.`Utenti` (`Username`, `U_password`, `Ruolo`) VALUES
('BNCMRA79M53H501F', 'admin', 'macchinista');
```

```
COMMIT;
```


Codice del Front-End

defines.h

```
#pragma once
```

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

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

```
struct configuration {  
    char *host;  
    char *db_username;  
    char *db_password;  
    unsigned int port;  
    char *database;  
    char username[128];  
    char password[128];  
};
```

```
extern struct configuration conf;
```

```
extern int parse_config(char *path, struct configuration *conf);  
extern char *getInput(unsigned int lung, char *stringa, bool hide);  
extern bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive);  
extern char multiChoice(char *domanda, char choices[], int num);  
extern void print_error(MYSQL *conn, char *message);  
extern void print_stmt_error(MYSQL_STMT *stmt, char *message);  
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool  
close_stmt);  
extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);  
extern void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title, int *num_result);  
extern void run_as_passenger(MYSQL *conn);  
extern void run_as_driver(MYSQL *conn);  
extern void run_as_controller(MYSQL *conn);  
extern void run_as_administrator(MYSQL *conn);  
extern void run_as_maintainer(MYSQL *conn);  
extern void mostra_turni(MYSQL *conn);  
extern int parse_date(char *date, MYSQL_TIME *parsed);  
extern char *parse_time(char *time);  
extern int date_compare(char *s);  
extern int time_compare(char *s);  
extern int int_compare(char *s);
```

main.c

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

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

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

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

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

```
typedef enum {  
    AMMINISTRATORE = 1,
```

```
MACCHINISTA = 2,
CAPOTRENO = 3,
MANUTENTORE,
FAILED_LOGIN,
PASSEGGERO
} role_t;

struct configuration conf;

static MYSQL *conn;

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

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

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

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

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

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

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

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

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

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

```

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

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

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

mysql_stmt_close(login_procedure);
printf("%d\n", role);
return role;

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

int main(void) {
    role_t role;

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

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

    if (mysql_real_connect(conn, conf.host, conf.db_username, conf.db_password, conf.database,
conf.port, NULL, CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS) == NULL) {
        fprintf (stderr, "mysql_real_connect() failed\n");
        mysql_close (conn);
        exit(EXIT_FAILURE);
    }
    if(yesOrNo("Need to book a ticket? ", 'y', 'n', true, true)){
        role = PASSEGGERO;
    }
    else{
        printf("Username: ");

```

```

    getInput(128, conf.username, false);
    printf("Password: ");
    getInput(128, conf.password, true);
    role = attempt_login(conn, conf.username, conf.password);
}
switch(role) {
    case MACCHINISTA:
        run_as_driver(conn);
        break;

    case CAPOTRENO:
        run_as_controller(conn);
        break;

    case AMMINISTRATORE:
        run_as_administrator(conn);
        break;
    case MANUTENTORE:
        run_as_maintainer(conn);
        break;
    case FAILED_LOGIN:
        fprintf(stderr, "Invalid credentials\n");
        exit(EXIT_FAILURE);
        break;
    case PASSEGGERO:
        run_as_passenger(conn);
        break;
    default:
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
        abort();
}

printf("Bye!\n");

mysql_close (conn);
return 0;
}

```

passaggero.c

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mysql.h>
#include "defines.h"

```

```

static void find_route(MYSQL *conn, char *p_partenza, char *c_partenza, char *p_arrivo, char
*c_arrivo, MYSQL_TIME date, bool *connect) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[5];
    int status;

```

```
char header[512];
int results;

// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call trova_viaggio(?,?,?,?)", conn)) {
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize search\n", false);
    *connect = false;
}

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

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

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

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

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

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

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

// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
    print_stmt_error(prepared_stmt, "An error occurred while retrieving the route.");
    *connect = false;
    goto out;
}
// We have multiple result sets here!
do {
    // Skip OUT variables (although they are not present in the procedure...)
    if(conn->server_status & SERVER_PS_OUT_PARAMS) {
        goto next;
    }
    sprintf(header, "\nPossible routes:\n");
```

```

    dump_result_set(conn, prepared_stmt, header, &results);
    // more results? -1 = no, >0 = error, 0 = yes (keep looking)
    next:
    status = mysql_stmt_next_result(prepared_stmt);
    if (status > 0){
        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
        *connect = false;
    }
    if(results == 0){
        printf("No routes found\n");
        *connect = false;
    }
} while (status == 0);

out:
mysql_stmt_close(prepared_stmt);
}

static int select_route(MYSQL *conn, int tratta, MYSQL_TIME data, char *ora, char *provincia,
char *citta, char *stazione, int classe, char *cf, char *nome,
char *cognome, MYSQL_TIME nascita, char *numero){
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[13];
    int ret;
    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call prenota_biglietto(?,?,?,?,?,?,?,?,?,?)", conn)){
        finish_with_stmt_error(conn, prepared_stmt, "Unable to book ticket\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);

    param[1].buffer_type = MYSQL_TYPE_DATE;
    param[1].buffer = &data;
    param[1].buffer_length = sizeof(MYSQL_TIME);

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

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

    param[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[4].buffer = stazione;

```

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

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

param[6].buffer_type = MYSQL_TYPE_SHORT;
param[6].buffer = &classe;
param[6].buffer_length = sizeof(classe);

param[7].buffer_type = MYSQL_TYPE_STRING;
param[7].buffer = cf;
param[7].buffer_length = strlen(cf);

param[8].buffer_type = MYSQL_TYPE_VAR_STRING;
param[8].buffer = nome;
param[8].buffer_length = strlen(nome);

param[9].buffer_type = MYSQL_TYPE_VAR_STRING;
param[9].buffer = cognome;
param[9].buffer_length = strlen(cognome);

param[10].buffer_type = MYSQL_TYPE_DATE;
param[10].buffer = (char *)&nascita;
param[10].buffer_length = sizeof(MYSQL_TIME);

param[11].buffer_type = MYSQL_TYPE_STRING;
param[11].buffer = numero;
param[11].buffer_length = strlen(numero);

param[12].buffer_type = MYSQL_TYPE_LONG; //OUT
param[12].buffer = &ret;
param[12].buffer_length = sizeof(ret);

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

// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
    print_stmt_error(prepared_stmt, "An error occurred while searching a ticket.");
    goto out;
}
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
param[0].buffer = &ret;
param[0].buffer_length = sizeof(ret);

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

```
    goto out;
}

// Retrieve output parameter
if(mysql_stmt_fetch(prepared_stmt)) {
    print_stmt_error(prepared_stmt, "Could not buffer results");
    goto out;
}
out:
mysql_stmt_close(prepared_stmt);
return ret;
}

void run_as_passenger(MYSQL *conn)
{
    char options[2] = {'1','2'};
    char op;
    bool connect = true;

    //Input for find_route routine
    char p_partenza[20];
    char c_partenza[20];
    char p_arrivo[20];
    char c_arrivo[20];
    char date[11];
    char tratta[10];
    char stazione[30];
    char orario[6];
    char classe;
    MYSQL_TIME parsed_date;
    char *parsed_time;
    char cf[17];
    char nome[20];
    char cognome[20];
    char nascita[11];
    MYSQL_TIME parsed_nascita;
    char numero[17];
    int codice;

    printf("Redircting to booking page...\n");

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

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



```

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

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

    switch(op) {
        case '1':
            printf("FROM:\nprovincia\t");
            getInput(20, p_partenza, false);
            printf("città\t");
            getInput(20, c_partenza, false);
            printf("\nTO:\nprovincia\t");
            getInput(20, p_arrivo, false);
            printf("città\t");
            getInput(20, c_arrivo, false);
            printf("\nDAY(yyyy-mm-dd):\n");
            while(true){
                getInput(11, date, false);
                if(date_compare(date)) break;
                else printf("Wrong format, please try again: ");
            }
            parse_date(date, &parsed_date);
            find_route(conn, p_partenza, c_partenza, p_arrivo, c_arrivo, parsed_date, &connect);
            if(connect){
                printf("Select your favorite Tratta: ");
                while(true){
                    getInput(10, tratta, false);
                    if(int_compare(tratta)) break;
                    else printf("Wrong format, please try again: ");
                }
                printf("From which station do you wish to leave? ");
                getInput(30, stazione, false);
                printf("Select your favorite time(hh:mm): ");
                while(true){
                    getInput(6, orario, false);
                    if(time_compare(orario)) break;
                    else printf("Wrong format, please try again: ");
                }
                parsed_time = parse_time(orario);
                classe = multiChoice("Select class: ", options, 2);
                printf("\033[2J\033[H");

                //inserire le informazioni del passeggero
                printf("Please insert passenger informations\n");
                printf("CF: ");
                getInput(17, cf, false);
            }
        }
    }
}

```

```

    printf("Name: ");
    getInput(20, nome, false);
    printf("Surname: ");
    getInput(20, cognome, false);
    printf("Date of birth(yyyy-mm-dd): ");
    while(true){
        getInput(11, nascita, false);
        if(date_compare(nascita)) break;
        else printf("Wrong format, please try again: ");
    }
    parse_date(nascita, &parsed_nascita);
    printf("Credit card number: ");
    getInput(17, numero, false);
    codice = select_route(conn, atoi(tratta), parsed_date, parsed_time, p_partenza,
        c_partenza, stazione, classe-'0', cf, nome, cognome, parsed_nascita, numero);
    printf("This is your ticket number: %d!", codice);
}
break;

case '2':
    return;

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

```

macchinista.c

```

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

void mostra_turni(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];
    int status;
    char header[512];
    int results;

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

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

```

```

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

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

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

// We have multiple result sets here!
do {
    // Skip OUT variables (although they are not present in the procedure...)
    if(conn->server_status & SERVER_PS_OUT_PARAMS) {
        goto next;
    }
    sprintf(header, "\nShifts:\n");
    dump_result_set(conn, prepared_stmt, header, &results);

    // more results? -1 = no, >0 = error, 0 = yes (keep looking)
    next:
    status = mysql_stmt_next_result(prepared_stmt);
    if (status > 0){
        finish_with_stmt_error(conn, prepared_stmt, "Unexpected condition", true);
    }
    if(results == 0) printf("You don't have any shift this week yet\n");

} while (status == 0);

mysql_stmt_close(prepared_stmt);
}

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

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

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

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

```

```

    exit(EXIT_FAILURE);
}

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

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

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

        case '2':
            return;

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

```

capotreno.c

```

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

static void controlla_biglietti(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];
    MYSQL_BIND param1;
    char header[512];
    char codice[10];
    int valid;
    int results;

    printf("Inserire il codice della prenotazione: ");
    getInput(20, codice, false);
    if(!setup_prepared_stmt(&prepared_stmt, "call controllo_biglietti(?,?)", conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize check_ticket statement\n",
            false);
    }
    int i_codice = atoi(codice);
    // Prepare parameters

```

```

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

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

param[1].buffer_type = MYSQL_TYPE_SHORT;
param[1].buffer = &valid;
param[1].buffer_length = sizeof(valid);

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

// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
    finish_with_stmt_error(conn, prepared_stmt, "Could not check ticket list\n", true);
}
if(conn->server_status & SERVER_PS_OUT_PARAMS) {
    mysql_stmt_next_result(prepared_stmt);
}
sprintf(header, "\nInformation:\n");
dump_result_set(conn, prepared_stmt, header, &results);
if(results == 0){
    printf("Non existent ticket\n");
    goto err;
}
    mysql_stmt_next_result(prepared_stmt);
// Prepare output parameters
memset(&param1, 0, sizeof(param1));
param1.buffer_type = MYSQL_TYPE_SHORT; // OUT
param1.buffer = &valid;
param1.buffer_length = sizeof(valid);

if(mysql_stmt_bind_result(prepared_stmt, &param1)) {
    print_stmt_error(prepared_stmt, "Could not retrieve output parameter");
    goto err;
}

// Retrieve output parameter
if(mysql_stmt_fetch(prepared_stmt)) {
    print_stmt_error(prepared_stmt, "Could not buffer results");
    goto err;
}
printf("%c\n", valid);
if(valid == 1){
    printf("Ticket already used\n");
}
else{

```

```
    printf("Valid ticket\n");
}

err:
mysql_stmt_close(prepared_stmt);
}

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

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

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

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

    while(true) {
        printf("\033[2J\033[H");
        printf("**** What should I do for you? ****\n\n");
        printf("1) Show my shift\n");
        printf("2) Check ticket\n");
        printf("3) Quit\n");

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

        switch(op) {
            case '1':
                mostra_turni(conn);
                break;
            case '2':
                controlla_biglietti(conn);
                break;
            case '3':
                return;

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

manutentore.c

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

void insert_report(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];
    char char_id[20];
    int id;
    char report[2048];

    printf("Vehicle id: ");
    while(true){
        getInput(20, char_id, false);
        if(int_compare(char_id)) break;
        else printf("Wrong format, try again: ");
    }
    printf("Report: \n");
    getInput(2048, report, false);

    id = atoi(char_id);

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

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

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

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

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

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not insert report\n", true);
    }
    else printf("Report correctly inserted...\n");
}
```

```

    mysql_stmt_close(prepared_stmt);
}

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

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

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

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

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

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

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

            case '2':
                return;

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

```

ammistratore.c

```

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

```



```
#include "defines.h"

char *create_string(int max, char *title, int buffer_size){
    char *s=malloc(buffer_size);
    if(s == NULL) {
        printf("Error: malloc has failed\n");
        return NULL;
    }
    int nextIndex=0;
    for(int i=0; i<max; i++){
        printf("%s %d :", title, i+1);
        getInput(buffer_size, s+nextIndex, false);
        nextIndex = strlen(s);
        memcpy(s+nextIndex, "$", 1);
        nextIndex++;
    }
    memcpy(s+nextIndex, "\\0", 1);
    return s;
}

static void add_train(MYSQL *conn)
{
    MYSQL_STMT *prepared_stmt;

    // Input for the registration routine
    char char_matricola[5];
    int matricola;
    char marca[20];
    char modello[20];
    char char_num_vagoni[10];
    int num_vagoni;
    char *portata;
    char *posti;
    char *classe;

    char options[2] = {'1','2'};
    char op;

    // Get the required information
    printf("\nRegistration number: ");
    while(true){
        getInput(5, char_matricola, false);
        if(int_compare(char_matricola)) break;
        else printf("Wrong format, please try again: ");
    }
    matricola = atoi(char_matricola);
    printf("Brand: ");
    getInput(20, marca, false);
    printf("Model: ");
    getInput(20, modello, false);
```

```
printf("Wagon number: ");
while(true){
    getInput(10, char_num_vagoni, false);
    if(int_compare(char_num_vagoni)) break;
    else printf("Wrong format, please try again: ");
}
num_vagoni = atoi(char_num_vagoni);

printf("Train type:\n1) Goods\n2) Passengers\n");
op = multiChoice("Select: ", options, 2);
switch(op){
    case '1':
        portata = create_string(num_vagoni, "Max carrying capacity for wagon", 150);

        MYSQL_BIND param[5];
        // Prepare stored procedure call

        if(!setup_prepared_stmt(&prepared_stmt, "call insert_treno_merci(?, ?, ?, ?, ?)", conn)) {
            finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize train insertion
statement\n", false);
        }
        // Prepare parameters
        memset(param, 0, sizeof(param));

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

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

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

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

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

        if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
            finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for train
insertion\n", true);
        }
        break;
    case '2':
```

```

    posti = create_string(num_vagoni, "Seats number for wagon", 50);
    classe = create_string(num_vagoni, "Class for wagon", 50);
    printf("%s\n", classe);
    MYSQL_BIND param1[6];
    // Prepare stored procedure call

    if(!setup_prepared_stmt(&prepared_stmt, "call insert_treno_passeggeri(?, ?, ?, ?, ?, ?)",
conn)) {
        finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize train insertion
statement\n", false);
    }
    // Prepare parameters
    memset(param1, 0, sizeof(param1));

    param1[0].buffer_type = MYSQL_TYPE_LONG;
    param1[0].buffer = &matricola;
    param1[0].buffer_length = sizeof(matricola);

    param1[1].buffer_type = MYSQL_TYPE_VAR_STRING;
    param1[1].buffer = marca;
    param1[1].buffer_length = strlen(marca);

    param1[2].buffer_type = MYSQL_TYPE_VAR_STRING;
    param1[2].buffer = modello;
    param1[2].buffer_length = strlen(modello);

    param1[3].buffer_type = MYSQL_TYPE_LONG;
    param1[3].buffer = &num_vagoni;
    param1[3].buffer_length = sizeof(num_vagoni);

    param1[4].buffer_type = MYSQL_TYPE_VAR_STRING;
    param1[4].buffer = posti;
    param1[4].buffer_length = strlen(posti);

    param1[5].buffer_type = MYSQL_TYPE_VAR_STRING;
    param1[5].buffer = classe;
    param1[5].buffer_length = strlen(classe);

    if (mysql_stmt_bind_param(prepared_stmt, param1) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for train
insertion\n", true);
    }
    break;
default:
    fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
    abort();
}
// Run procedure
if (mysql_stmt_execute(prepared_stmt) != 0) {
    print_stmt_error (prepared_stmt, "An error occurred while adding the train.");
}

```

```

    } else {
        printf("Train correctly added...\n");
    }

    mysql_stmt_close(prepared_stmt);
}

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

    char option[2] = {'1', '2'};
    // Input for the registration routine
    char cf[17];
    char name[20];
    char surname[20];
    char char_data[11];
    MYSQL_TIME data;
    char luogo[30];
    char char_ruolo;
    int ruolo;

    // Get the required information
    printf("\nEmployee cf: ");
    getInput(17, cf, false);
    printf("Employee name: ");
    getInput(20, name, false);
    printf("Employee surname: ");
    getInput(20, surname, false);
    printf("Employee birth date(yyyy-mm-dd): ");
    while(true){
        getInput(11, char_data, false);
        if(date_compare(char_data)) break;
        else printf("Wrong format, try again: ");
    }
    parse_date(char_data, &data);
    printf("Employee birth place: ");
    getInput(30, luogo, false);
    printf("Employee role:\n1)Macchinista\n2)Capotreno\n");
    char_ruolo=multiChoice("Select:", option, 2);
    if (char_ruolo == '1') ruolo = 0;
    else if(char_ruolo == '2') ruolo = 1;
    else {
        printf("Invalid option\n");
        abort();
    }

    // Prepare stored procedure call
    if(!setup_prepared_stmt(&prepared_stmt, "call insert_lavoratore(?, ?, ?, ?, ?, ?)", conn)) {

```

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

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

param[0].buffer_type = MYSQL_TYPE_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 = &data;
param[3].buffer_length = sizeof(MYSQL_TIME);

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

param[5].buffer_type = MYSQL_TYPE_SHORT;
param[5].buffer = &ruolo;
param[5].buffer_length = sizeof(ruolo);

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

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

printf("Employee correctly added...\n");

out:
mysql_stmt_close(prepared_stmt);
}

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

```
MYSQL_STMT *prepared_stmt;
MYSQL_BIND param[3];
char options[4] = {'1', '2', '3', '4'};
char r;

// Input for the registration routine
char username[17];
char password[20];
char ruolo[20];

// Get the required information
printf("\nUsername: ");
getInput(17, username, false);
printf("password: ");
getInput(20, password, true);
printf("Assign a possible role:\n");
printf("\t1) Driver\n");
printf("\t2) Controller\n");
printf("\t3) Administrator\n");
printf("\t4) Maintainer\n");
r = multiChoice("Select role", options, 4);

// Convert role into enum value
switch(r) {
    case '1':
        strcpy(ruolo, "MACCHINISTA");
        break;
    case '2':
        strcpy(ruolo, "CAPOTRENO");
        break;
    case '3':
        strcpy(ruolo, "AMMINISTRATORE");
        break;
    case '4':
        strcpy(ruolo, "MANUTENTORE");
        break;
    default:
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
        abort();
}

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

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

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

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

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

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

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

mysql_stmt_close(prepared_stmt);
}

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

    // Input for the registration routine
    char char_matricola[5];
    char char_id_tratta[20];
    int matricola;
    int tratta;

    // Get the required information
    printf("\nTrain: ");
    while(true){
        getInput(5, char_matricola, false);
        if(int_compare(char_matricola)) break;
        else printf("Wrong format, try again: ");
    }
    matricola = atoi(char_matricola);
    printf("Route: ");
    while(true){
        getInput(20, char_id_tratta, false);
        if(int_compare(char_id_tratta)) break;
```

```

    else printf("Wrong format, try again: ");
}
tratta = atoi(char_id_tratta);

// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call assign_train(?, ?)", conn)) {
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize train assignment
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);

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

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

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

printf("Train correctly assigned...\n");

out:
mysql_stmt_close(prepared_stmt);
}

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

    // Input for the registration routine
    char cf[17];
    char char_data[11];
    MYSQL_TIME data;
    char char_inizio[6];
    char *inizio;
    char char_fine[6];

```



```
char *fine;
char char_matricola[5];
int matricola;

// Get the required information
printf("\nEmployee cf: ");
getInput(17, cf, false);
printf("Date(yyyy-mm-dd): ");
while(true){
    getInput(11, char_data, false);
    if(date_compare(char_data)) break;
    else printf("Wrong format, try again: ");
}
printf("Start time(hh:mm): ");
while(true){
    getInput(6, char_inizio, false);
    if(time_compare(char_inizio)) break;
    else printf("Wrong format, try again: ");
}
printf("End time(hh-mm): ");
while(true){
    getInput(6, char_fine, false);
    if(time_compare(char_fine)) break;
    else printf("Wrong format, try again: ");
}
printf("Train: ");
while(true){
    getInput(5, char_matricola, false);
    if(int_compare(char_matricola)) break;
    else printf("Wrong format, try again: ");
}
// Convert values
parse_date(char_data, &data);
inizio = parse_time(char_inizio);
fine = parse_time(char_fine);
matricola = atoi(char_matricola);

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

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

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

```

param[1].buffer_type = MYSQL_TYPE_DATE;
param[1].buffer = &data;
param[1].buffer_length = sizeof(MYSQL_TIME);

param[2].buffer_type = MYSQL_TYPE_STRING;
param[2].buffer = inizio;
param[2].buffer_length = strlen(inizio);

param[3].buffer_type = MYSQL_TYPE_STRING;
param[3].buffer = fine;
param[3].buffer_length = strlen(fine);

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

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

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

mysql_stmt_close(prepared_stmt);
}

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

    // Input for the registration routine
    char cf[17];
    char char_data[11];
    MYSQL_TIME data;

    // Get the required information
    printf("\nEmployee that needs replacement cf: ");
    getInput(17, cf, false);
    printf("Date(yyyy-mm-dd): ");
    while(true){
        getInput(11, char_data, false);
        if(date_compare(char_data)) break;
        else printf("Wrong format, try again: ");
    }
}

```

```
// Convert values
parse_date(char_data, &data);

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

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

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

param[1].buffer_type = MYSQL_TYPE_DATE;
param[1].buffer = &data;
param[1].buffer_length = sizeof(MYSQL_TIME);

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

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

mysql_stmt_close(prepared_stmt);
}

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

    // Input for the registration routine
    char char_tratta[20];
    int tratta;
    char char_data[11];
    MYSQL_TIME data;
    char char_treno[5];
    int treno;
    int num_fermate;
    char *oraPartenza;
```

```
char *oraArrivo;

// Get the required information
printf("\nRoute: ");
while(true){
    getInput(5, char_tratta, false);
    if(int_compare(char_tratta)) break;
    else printf("Wrong format, try again: ");
}
printf("Date(yyyy-mm-dd): ");
while(true){
    getInput(11, char_data, false);
    if(date_compare(char_data)) break;
    else printf("Wrong format, try again: ");
}
printf("Train: ");
while(true){
    getInput(20, char_treno, false);
    if(int_compare(char_treno)) break;
    else printf("Wrong format, try again: ");
}

// Convert values
tratta = atoi(char_tratta);
parse_date(char_data, &data);
treno = atoi(char_treno);

//ricava il numero di fermate
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &tratta;
param[0].buffer_length = sizeof(tratta);

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

if(!setup_prepared_stmt(&prepared_stmt, "call num_fermate(?, ?)", conn)) {
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize retrieve stops number
statement\n", false);
}

if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
    finish_with_stmt_error(conn, prepared_stmt, "Could not bind parameters for stops number\n",
true);
}
if (mysql_stmt_execute(prepared_stmt) != 0) {
    print_stmt_error(prepared_stmt, "An error occurred while retrieving stops number.");
    goto out;
}
```

```
memset(param, 0, sizeof(param));
param[0].buffer_type = MYSQL_TYPE_LONG;
param[0].buffer = &num_fermate;
param[0].buffer_length = sizeof(num_fermate);
if(mysql_stmt_bind_result(prepared_stmt, param)) {
    print_stmt_error(prepared_stmt, "Could not retrieve output parameter");
    goto out;
}
if(mysql_stmt_fetch(prepared_stmt)) {
    print_stmt_error(prepared_stmt, "Could not buffer results");
    goto out;
}
mysql_stmt_close(prepared_stmt);

oraPartenza = create_string(num_fermate, "Departure time for stop", 150);
oraArrivo = create_string(num_fermate, "Arrival time for stop", 150);

// Prepare stored procedure call
if(!setup_prepared_stmt(&prepared_stmt, "call insert_viaggio(?, ?, ?, ?, ?)", conn)) {
    finish_with_stmt_error(conn, prepared_stmt, "Unable to initialize travel insert 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);

param[1].buffer_type = MYSQL_TYPE_DATE;
param[1].buffer = &data;
param[1].buffer_length = sizeof(MYSQL_TIME);

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

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

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

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

```

    }

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        print_stmt_error (prepared_stmt, "An error occurred while inserting the travel.");
    } else {
        printf("Travel correctly inserted...\n");
    }
    out:
    mysql_stmt_close(prepared_stmt);
}

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

    // Input for the registration routine
    char tipo[30];
    char char_massa[6];
    int massa;
    char provenienza[12];
    char direzione[12];
    char char_tratta[20];
    int tratta;
    char char_data[11];
    MYSQL_TIME data;
    char char_ora[6];
    char *ora;

    // Get the required information
    printf("\nGoods type: ");
    getInput(30, tipo, false);
    printf("Goods mass: ");
    while(true){
        getInput(6, char_massa, false);
        if(int_compare(char_massa)) break;
        else printf("Wrong format, try again: ");
    }
    printf("From: ");
    getInput(12, provenienza, false);
    printf("To: ");
    getInput(12, direzione, false);
    printf("Route: ");
    while(true){
        getInput(20, char_tratta, false);
        if(int_compare(char_tratta)) break;
        else printf("Wrong format, try again: ");
    }
    printf("Date(yyyy-mm-dd): ");

```

```
while(true){
    getInput(11, char_data, false);
    if(date_compare(char_data)) break;
    else printf("Wrong format, try again: ");
}
printf("Time(hh:mm): ");
while(true){
    getInput(6, char_ora, false);
    if(time_compare(char_ora)) break;
    else printf("Wrong format, try again: ");
}

// Convert values
massa = atoi(char_massa);
tratta = atoi(char_tratta);
parse_date(char_data, &data);
ora = parse_time(char_ora);

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

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

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

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

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

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

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

param[5].buffer_type = MYSQL_TYPE_DATE;
param[5].buffer = &data;
param[5].buffer_length = sizeof(MYSQL_TIME);
```

```

param[6].buffer_type = MYSQL_TYPE_STRING;
param[6].buffer = ora;
param[6].buffer_length = strlen(ora);

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

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

mysql_stmt_close(prepared_stmt);
}

void run_as_administrator(MYSQL *conn)
{
    char options[9] = {'1','2', '3', '4', '5', '6', '7', '8', '9'};
    char op;

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

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

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

    while(true) {
        printf("\033[2J\033[H");
        printf("*** What should I do for you? ***\n\n");
        printf("1) Add new train\n");
        printf("2) Add new employee\n");
        printf("3) Create new user\n");
        printf("4) Assign train to route\n");
        printf("5) Insert shift\n");
        printf("6) Replace shift\n");
        printf("7) Add new travel\n");
        printf("8) Add goods and assign them to a travel\n");
        printf("9) Quit\n");
    }
}

```



```

op = multiChoice("Select an option", options, 9);

switch(op) {
    case '1':
        add_train(conn);
        break;
    case '2':
        add_employee(conn);
        break;
    case '3':
        create_user(conn);
        break;
    case '4':
        assign_train(conn);
        break;
    case '5':
        insert_shift(conn);
        break;
    case '6':
        replace_shift(conn);
        break;
    case '7':
        insert_travel(conn);
        break;
    case '8':
        insert_goods(conn);
        break;
    case '9':
        return;
    default:
        fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
        abort();
}
getchar();
}
}

```

utils.c

```

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

#include "defines.h"

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

```

```

        mysql_stmt_sqlstate(stmt),
        mysql_stmt_error (stmt));
    }
}

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

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

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

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

    mysql_stmt_attr_set(*stmt, STMT_ATTR_UPDATE_MAX_LENGTH, &update_length);

    return true;
}

void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message, bool
close_stmt)
{
    print_stmt_error(stmt, message);
    if(close_stmt) mysql_stmt_close(stmt);
    mysql_close(conn);
    exit(EXIT_FAILURE);
}

static void print_dashes(MYSQL_RES *res_set)

```

```

{
    MYSQL_FIELD *field;
    unsigned int i, j;

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

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

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

    mysql_field_seek (res_set, 0);

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

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

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

    print_dashes(res_set);
}

void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title, int *num_result)

```

```

{
    int i;
    int status;
    int num_fields; /* number of columns in result */
    MYSQL_FIELD *fields; /* for result set metadata */
    MYSQL_BIND *rs_bind; /* for output buffers */
    MYSQL_RES *rs_metadata;
    MYSQL_TIME *date;
    size_t attr_size;
    *num_result=-1;

    /* 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) {
    *num_result = *num_result + 1;
    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_TIME:
            date = (MYSQL_TIME *)rs_bind[i].buffer;
            printf(" %02d:%02d:%02d |", date->hour, date->minute, date->second);
        case MYSQL_TYPE_STRING:
            printf(" %-*s |", (int)fields[i].max_length, (char *)rs_bind[i].buffer);
            break;

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

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

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

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

```

    }
    putchar('\n');
    print_dashes(rs_metadata);
}

mysql_free_result(rs_metadata); /* free metadata */

/* free output buffers */
for (i = 0; i < num_fields; i++) {
    free(rs_bind[i].buffer);
}
free(rs_bind);
}
}
int date_compare(char *s){
    if(strlen(s)!=10) return 0;
    for(int i=0; i<10; i++){
        if(i!= 4 && i!= 7 && (s[i]<48 || s[i]>57)) return 0;
    }
    if(s[4]!='-' || s[7]!='-') return 0;
    return 1;
}
int time_compare(char *s){
    if (strlen(s)!=5) return 0;
    for(int i=0; i<4; i++){
        if(i!=2 && (s[i]<48 || s[i]>57)) return 0;
    }
    if(s[2]!=':') return 0;
    return 1;
}
int int_compare(char *s){
    for(unsigned long i=0;i<strlen(s);i++){
        if(s[i]<48||s[i]>57) return 0;
    }
    return 1;
}

```

inout.c

```

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

```

```

#include "defines.h"

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

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

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

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

        // Cattura i segnali che altrimenti potrebbero far terminare il programma, lasciando l'utente
        // senza output sulla shell
        sigemptyset(&sa.sa_mask);
        sa.sa_flags = 0; // Per non resettare le system call
        sa.sa_handler = handler;
        (void) sigaction(SIGALRM, &sa, &savealrm);
        (void) sigaction(SIGINT, &sa, &saveint);
        (void) sigaction(SIGHUP, &sa, &savehup);
        (void) sigaction(SIGQUIT, &sa, &savequit);
        (void) sigaction(SIGTERM, &sa, &saveterm);
        (void) sigaction(SIGTSTP, &sa, &savetstp);
        (void) sigaction(SIGTTIN, &sa, &savettin);
        (void) sigaction(SIGTTOU, &sa, &savettou);

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

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

```



```

    stringa[i] = '\0';
    break;
} else
    stringa[i] = c;

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

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

// Se sono stati digitati 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 stringa;
}

```

```

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

bool yesOrNo(char *domanda, char yes, char no, bool predef, bool insensitive)
{
    // I caratteri 'yes' e 'no' devono essere minuscoli
    yes = tolower(yes);
    no = tolower(no);

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

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

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

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

char multiChoice(char *domanda, char choices[], int num)
{
    // Genera la stringa delle possibilità


```

```

char *possib = malloc(2 * num * sizeof(char));
int i, j = 0;
for(i = 0; i < num; i++) {
    possib[j++] = choices[i];
    possib[j++] = '/';
}
possib[j-1] = '\0'; // Per eliminare l'ultima '/'

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

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

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

```

parse.c

```

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

#include "defines.h"

#define BUFF_SIZE 4096

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

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

```

```

} jsmntype_t;

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

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

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

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

/**
 * Fills token type and boundaries.

```

```

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

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

    start = parser->pos;

    for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
        switch (js[parser->pos]) {
            /* In strict mode primitive must be followed by ",", " or "]" */
            case ':':
            case '\t': case '\r': case '\n': case ' ':
            case ',': case ']': case '}':
                goto found;
        }
        if (js[parser->pos] < 32 || js[parser->pos] >= 127) {
            parser->pos = start;
            return JSMN_ERROR_INVALID;
        }
    }

found:
    if (tokens == NULL) {
        parser->pos--;
        return 0;
    }
    token = jsmn_alloc_token(parser, tokens, num_tokens);
    if (token == NULL) {
        parser->pos = start;
        return JSMN_ERROR_NOMEM;
    }
    jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
    parser->pos--;
    return 0;
}

/**
 * Fills next token with JSON string.

```

```

*/
static int jsmn_parse_string(jsmn_parser *parser, const char *js,
    size_t len, jsmntok_t *tokens, size_t num_tokens) {
    jsmntok_t *token;

    int start = parser->pos;

    parser->pos++;

    /* Skip starting quote */
    for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
        char c = js[parser->pos];

        /* Quote: end of string */
        if (c == '"') {
            if (tokens == NULL) {
                return 0;
            }
            token = jsmn_alloc_token(parser, tokens, num_tokens);
            if (token == NULL) {
                parser->pos = start;
                return JSMN_ERROR_NOMEM;
            }
            jsmn_fill_token(token, JSMN_STRING, start+1, parser->pos);
            return 0;
        }

        /* Backslash: Quoted symbol expected */
        if (c == '\\' && parser->pos + 1 < len) {
            int i;
            parser->pos++;
            switch (js[parser->pos]) {
                /* Allowed escaped symbols */
                case '\"': case '/' : case '\\': case 'b' :
                case 'f' : case 'r' : case 'n' : case 't' :
                    break;
                /* Allows escaped symbol \uXXXX */
                case 'u':
                    parser->pos++;
                    for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0'; i++) {
                        /* If it isn't a hex character we have an error */
                        if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) || /* 0-9 */
                            (js[parser->pos] >= 65 && js[parser->pos] <= 70) || /* A-F */
                            (js[parser->pos] >= 97 && js[parser->pos] <= 102)))) { /* a-f */
                            parser->pos = start;
                            return JSMN_ERROR_INVALID;
                        }
                    }
                    parser->pos++;
                }
            }
            parser->pos--;
        }
    }

```

```

        break;
    /* Unexpected symbol */
    default:
        parser->pos = start;
        return JSMN_ERROR_INVALID;
    }
}
}
parser->pos = start;
return JSMN_ERROR_PART;
}

/**
 * Parse JSON string and fill tokens.
 */
static int jsmn_parse(jsmn_parser *parser, const char *js, size_t len, jsmntok_t *tokens, unsigned int
num_tokens) {
    int r;
    int i;
    jsmntok_t *token;
    int count = parser->toknext;

    for (; parser->pos < len && js[parser->pos] != '\0'; parser->pos++) {
        char c;
        jsmntype_t type;

        c = js[parser->pos];
        switch (c) {
            case '{': case '[':
                count++;
                if (tokens == NULL) {
                    break;
                }
                token = jsmn_alloc_token(parser, tokens, num_tokens);
                if (token == NULL)
                    return JSMN_ERROR_NOMEM;
                if (parser->toksuper != -1) {
                    tokens[parser->toksuper].size++;
                }
                token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);
                token->start = parser->pos;
                parser->toksuper = parser->toknext - 1;
                break;
            case '}': case ']':
                if (tokens == NULL)
                    break;
                type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);
                for (i = parser->toknext - 1; i >= 0; i--) {
                    token = &tokens[i];
                    if (token->start != -1 && token->end == -1) {

```

```

        if (token->type != type) {
            return JSMN_ERROR_INVALID;
        }
        parser->toksuper = -1;
        token->end = parser->pos + 1;
        break;
    }
}
/* Error if unmatched closing bracket */
if (i == -1) return JSMN_ERROR_INVALID;
for (; i >= 0; i--) {
    token = &tokens[i];
    if (token->start != -1 && token->end == -1) {
        parser->toksuper = i;
        break;
    }
}
break;
case '"':
    r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
    if (r < 0) return r;
    count++;
    if (parser->toksuper != -1 && tokens != NULL)
        tokens[parser->toksuper].size++;
    break;
case '\t' : case '\r' : case '\n' : case ' ':
    break;
case ':':
    parser->toksuper = parser->toknext - 1;
    break;
case ',':
    if (tokens != NULL && parser->toksuper != -1 &&
        tokens[parser->toksuper].type != JSMN_ARRAY &&
        tokens[parser->toksuper].type != JSMN_OBJECT) {
        for (i = parser->toknext - 1; i >= 0; i--) {
            if (tokens[i].type == JSMN_ARRAY || tokens[i].type == JSMN_OBJECT) {
                if (tokens[i].start != -1 && tokens[i].end == -1) {
                    parser->toksuper = i;
                    break;
                }
            }
        }
    }
}
break;
/* In non-strict mode every unquoted value is a primitive */
default:
    r = jsmn_parse_primitive(parser, js, len, tokens, num_tokens);
    if (r < 0) return r;
    count++;
    if (parser->toksuper != -1 && tokens != NULL)

```



```

        tokens[parser->toksuper].size++;
        break;

    }
}

if (tokens != NULL) {
    for (i = parser->toknext - 1; i >= 0; i--) {
        /* Unmatched opened object or array */
        if (tokens[i].start != -1 && tokens[i].end == -1) {
            return JSMN_ERROR_PART;
        }
    }
}

return count;
}

/**
 * Creates a new parser based over a given buffer with an array of tokens
 * available.
 */
static void jsmn_init(jsmn_parser *parser) {
    parser->pos = 0;
    parser->toknext = 0;
    parser->toksuper = -1;
}

static int jsoneq(const char *json, jsmntok_t *tok, const char *s)
{
    if (tok->type == JSMN_STRING
        && (int) strlen(s) == tok->end - tok->start
        && strncmp(json + tok->start, s, tok->end - tok->start) == 0) {
        return 0;
    }
    return -1;
}

static size_t load_file(char *filename)
{
    FILE *f = fopen(filename, "rb");
    if(f == NULL) {
        fprintf(stderr, "Unable to open file %s\n", filename);
        exit(1);
    }

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

```

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

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

config[fsize] = 0;
return fsize;
}

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

    load_file(path);

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

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

    /* Loop over all keys of the root object */
    for (i = 1; i < r; i++) {
        if (jsoneq(config, &t[i], "host") == 0) {
            /* We may use strdup() to fetch string value */
            conf->host = strdup(config + t[i+1].start, t[i+1].end-t[i+1].start);
            i++;
        } else if (jsoneq(config, &t[i], "username") == 0) {
            conf->db_username = strdup(config + t[i+1].start, t[i+1].end-t[i+1].start);
            i++;
        } else if (jsoneq(config, &t[i], "password") == 0) {
            conf->db_password = strdup(config + t[i+1].start, t[i+1].end-t[i+1].start);
            i++;
        } else if (jsoneq(config, &t[i], "port") == 0) {
            conf->port = strtol(config + t[i+1].start, NULL, 10);
            i++;
        } else if (jsoneq(config, &t[i], "database") == 0) {
```

```
        conf->database = strdup(config + t[i+1].start, t[i+1].end-t[i+1].start);
        i++;
    } else {
        printf("Unexpected key: %.*s\n", t[i].end-t[i].start, config + t[i].start);
    }
}
return 1;
}
int parse_date(char *date, MYSQL_TIME *parsed){
    char d[3];
    char m[3];
    char y[5];
    memcpy(y, date, 4);
    memcpy(m, date+5, 2);
    memcpy(d, date+8, 2);
    y[4]= '\0';
    m[2]= '\0';
    d[2]= '\0';
    parsed->year = atoi(y);
    parsed->month = atoi(m);
    parsed->day = atoi(d);
    return 0;
}

char *parse_time(char *time){
    char *parsed = malloc(10);
    memcpy(parsed, time, 5);
    memcpy(parsed+5, ":", 1);
    memcpy(parsed+6, "00\0", 2);
    return parsed;
}
```