S

Basi di Dati e Conoscenza

Progetto A.A. 2020/2021

SISTEMA DI GESTIONE DI TRASPORTO FERROVIARIO

0252956

Danilo D'Amico

Indice

1. Descrizione del Minimondo **2**

2. Analisi dei Requisiti **4**

3. Progettazione concettuale ..... Errore. Il segnalibro non è definito.

4. Progettazione logica **23**

5. Progettazione fisica **37**

Appendice: Implementazione **82**

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 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'agguinta 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
17, 18, 31	treni	corse	Nel primo paragrafo il termine "treno" viene utilizzato per intendere il veicolo composto da locomotrice e vagoni, mentre in questo caso si riferisce al numero di corse lungo una tratta.
23	macchinista	conducente	Unificazione sinonimi ambigui
24	Treno passeggeri	Corsa di un treno passeggeri	*Treno passeggeri* è un'entità che si riferisce all'oggetto treno, ma è chiaro che un capotreno non è associato al veicolo in sé ma ad una sua corsa
3	veicolo	treno	Per non confondere con le singole locomotrici e vagoni
8	carrozza	Vagone Passeggeri	Unificazione di sinonimi
9	vagone	Vagone Passeggeri	Disambiguazione tra vagone merci e vagone passeggeri
19	Amministratori del servizio	Gestori del servizio	Unificazione sinonimi
42	Vagone	Vagone Merci	Disambiguazione tra vagone merci e vagone passeggeri
43	Società	Azienda	Unificazione di sinonimi

Specifica disambiguata

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

(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 corse, la cui associazione viene gestita dai gestori del servizio. I gestori possono, su base periodica, modificare il numero di corse operanti su ciascuna tratta. I gestori 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 conducente non effettui più di 5 turni settimanali, per un massimo di 4 ore di lavoro. A ciascuna corsa di un 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 delle corse 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 merci 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	Un veicolo composto da una locomotrice ed un certo numero di vagoni (passeggeri o merci)		Treno passeggeri, Treno merci, locomotrice
Treno passeggeri	Una tipologia di treno dedicata al trasporto di passeggeri. Composto da una locomotrice e dei vagoni passeggeri		Vagone passeggeri, treno, locomotrice
Treno merci	Una tipologia di treno dedicata al trasporto di merci. Composto da una locomotrice e dei vagoni merci		Vagone merci, treno, locomotrice
Vagone	Ciò che viene trainato da un treno. Può essere per merci o passeggeri		Vagone merci, vagone passeggeri
Vagone passeggeri	Una tipologia di vagone, esclusiva dei treni passeggeri, adibita al trasporto di persone	carrozza	Vagone, Treno passeggeri
Vagone merci	Una tipologia di vagone, esclusiva dei treni merci, adibita al trasporto di merci di una società		Vagone, Treno merci
Locomotrice	Carrozza di guida di un		Treno

	treno		
Tratta	Insieme di fermate (di cui due capolinea) sulle quali avviene una corsa		Fermata, corsa
Fermata	Stop previsto di una corsa durante una tratta. Avviene in una stazione		Stazione, Tratta
Stazione	Luogo in cui avviene una fermata		Fermata
Corsa	Percorrenza effettiva di una tratta da parte di un treno		Tratta
Capolinea di partenza	Fermata in cui inizia una tratta		Fermata, Stazione
Capolinea di arrivo	Fermata in cui termina una tratta		Fermata, Stazione
Conducente	Impiegato che conduce un treno in una corsa	Macchinista	Turno, Treno
Capotreno	Impiegato capo del personale di un treno		Treno passeggeri, Turno
Turno	Turno lavorativo		Conducente, capotreno
Report sui turni di lavoro	Rapporto stilato da un lavoratore settimanalmente		Conducente, Capotreno
Acquirente	Cliente interessato all'acquisto di un biglietto per un treno passeggeri		Biglietto
Biglietto	Biglietto di accesso ad un treno passeggeri da parte di un acquirente		Tratta, Acquirente
Merce	Ciò che trasporta un Treno merci nei suoi vagoni merci		Azienda, Tratta, Vagone Merci
Azienda	Organizzazione che carica delle merci in un treno merci per spedirle ad un'altra azienda (o che lo riceve).	società	Vagone merci, merce

Raggruppamento dei requisiti in insiemi omogenei

Frasi di carattere generale
Si vuole realizzare un sistema informativo per la gestione dell'operatività di un'azienda di trasporto pubblico ferroviario ad alta velocità.

Frasi relative a Treno
I treni gestiti dal servizio sono caratterizzati da una matricola (codice univoco numerico di quattro cifre). Ogni treno è anche associato ad una data di acquisto e ad uno storico di manutenzione. Di ciascun treno è di interesse conoscere anche da quanti vagoni questo è composto.

Frasi relative a Treno Passeggeri
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. Per i treni adibiti al trasporto passeggeri sono memorizzati il numero di vagoni passeggeri di prima e seconda classe, il numero massimo di passeggeri che possono viaggiare in ciascun vagone passeggeri (con riferimento ai singoli posti). Si vuole anche realizzare, per quanto riguarda i treni passeggeri, una funzionalità di prenotazione di biglietto.

Frasi relative a Treno Merci
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.

Frasi relative a Vagone
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. 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 Vagone Passeggeri
Per i treni adibiti al trasporto passeggeri sono memorizzati il numero di vagoni passeggeri di prima e seconda classe, il numero massimo di passeggeri che possono viaggiare in ciascun vagone passeggeri (con riferimento ai singoli posti).

Frasi relative a Vagone Merci
I vagoni di un treno merci devono essere caratterizzati dalla portata massima (in tonnellate). 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.

Frasi relative a Locomotrice
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. 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 Tratta
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. Ogni tratta viene coperta da un numero predefinito di corse, la cui associazione viene gestita dai gestori del servizio

Frasi relative a Fermata
Ciascuna tratta ha un insieme di fermate identificate dal nome della stazione, dalla città e dalla provincia in cui si trova. 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.

Frasi relative a Stazione
Ciascuna tratta ha un insieme di fermate identificate dal nome della stazione, dalla città e dalla provincia in cui si trova.

Frasi relative a Corsa
Ogni tratta viene coperta da un numero predefinito di corse, la cui associazione viene gestita dai gestori del servizio. I gestori possono, su base periodica, modificare il numero di corse operanti su ciascuna tratta.

Frasi relative a Capolinea di Partenza
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.

Frasi relative a Capolinea di Arrivo
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.

Frasi relative a Conducente
I gestori 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 conducente non effettui più di 5 turni settimanali, per un massimo di 4 ore di lavoro.

Frasi relative a Capotreno
Nella gestione degli orari di lavoro, i gestori del servizio devono garantire che ciascun conducente non effettui più di 5 turni settimanali, per un massimo di 4 ore di lavoro. A ciascuna corsa di un treno passeggeri deve anche essere associato un capotreno, soggetto agli stessi vincoli orari.

Frasi relative a Turno
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.

Frasi relative a Report Turni di Lavoro
Ogni lavoratore ha la possibilità di generare un report sui propri turni di lavoro, su base settimanale, riportante l'indicazione degli orari e delle corse in cui esso è coinvolto.

Frasi relative a Acquirente
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.

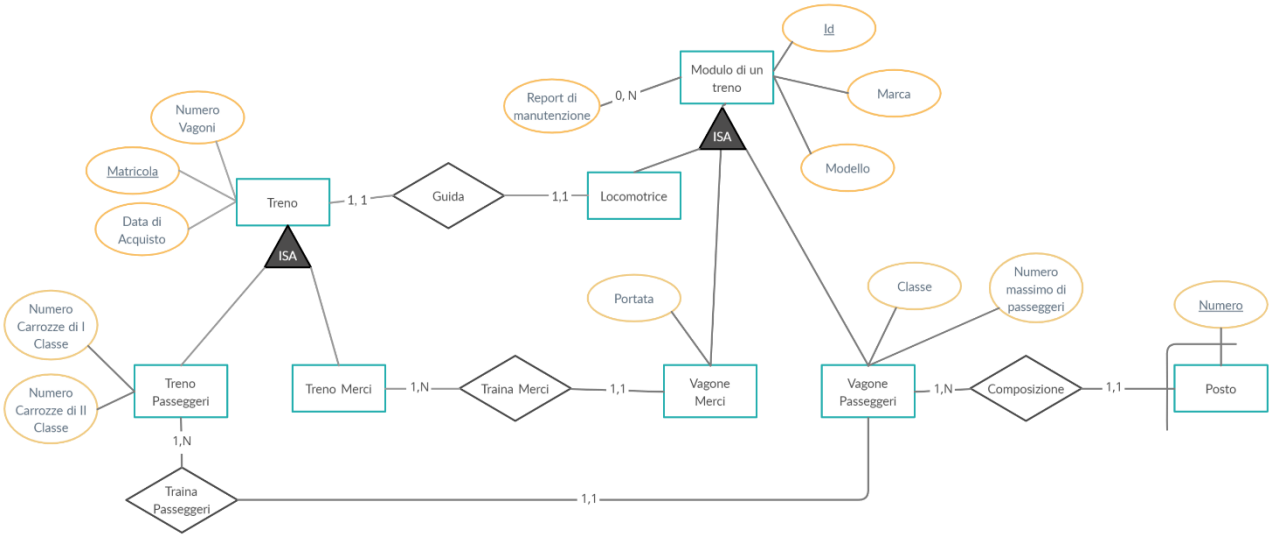
Frasi relative a Biglietto
Si vuole anche realizzare, per quanto riguarda i treni passeggeri, una funzionalità di prenotazione di biglietto. All'atto di acquisto di un biglietto (identificato per tratta e cui viene associato un posto disponibile sul treno con l'aggiunta di un codice di prenotazione univoco)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

Frasi relative a 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. 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 merci siano trasportate merci di una sola società, dirette ad una sola società.

Frasi relative a Azienda
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 merci siano trasportate merci di una sola società, dirette ad una sola società.

3. Progettazione concettuale

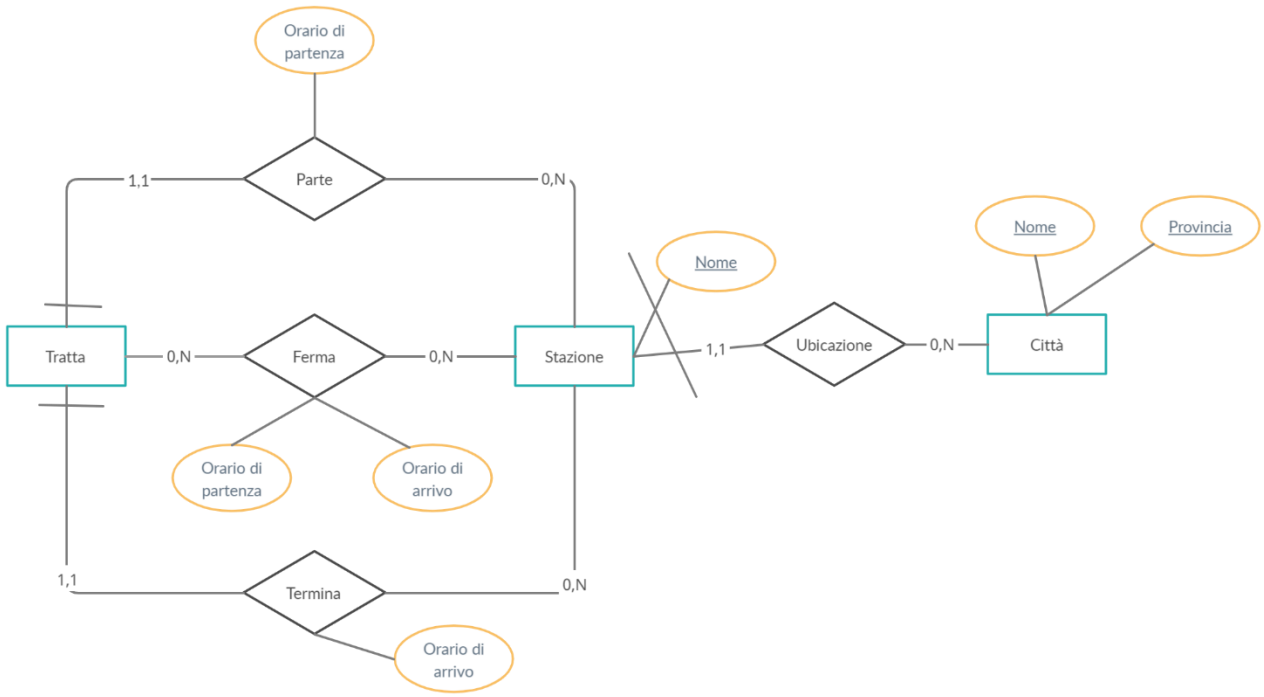
Costruzione dello schema E-R



Questo primo schema modella le tipologie di treno e le loro componenti. Dato che *Locomotrice*, *Vagone Passeggeri* e *Vagone Merci* posseggono gli stessi attributi si è deciso di generalizzarli mediante l'entità *Modulo di un Treno*. Per poter identificare le varie componenti si è aggiunto un attributo *Id* a questa entità, in quanto marca e modello non sono sufficienti per identificare un preciso vagone o una precisa locomotrice. Nell'ultimo paragrafo della specifica è richiesta la possibilità di inserire report di manutenzione per vagoni e locomotive, perciò è stato aggiunto un attributo multi-valore *Report di manutenzione* ad ogni modulo. Visualizzare lo storico di un treno sarà l'operazione sulla base di dati che consente di visualizzare tutti i report di manutenzione dei moduli di un treno.

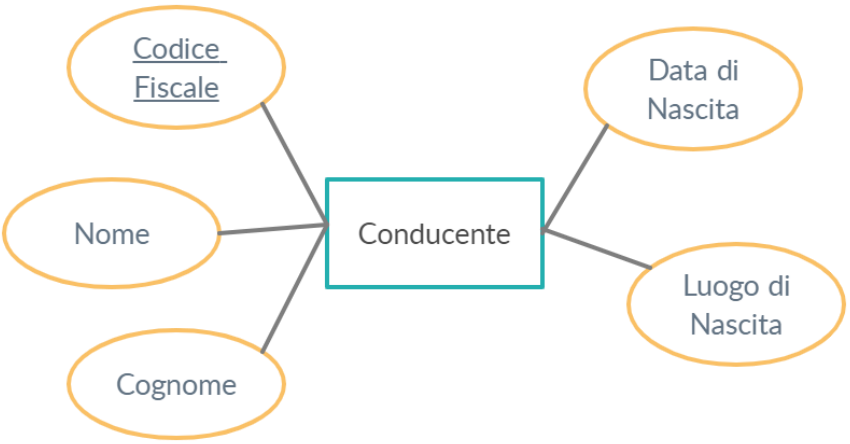
Affinché sia possibile controllare i singoli posti di un vagone, è stata anche aggiunta un'identità *Posto* identificata dal numero del posto all'interno del vagone e dal vagone stesso.

Per quanto riguarda i treni, viene specificato nel testo che questi si dividono in *Treni Passeggeri* e *Treni Merci* e che siano composti da vagoni di tipologie differenti. Per questo motivo si è deciso di associare le specializzazioni dei treni a quelle dei vagoni anziché creare un'associazione *Treno-Vagone* (considerando *Vagone* una generalizzazione di *Vagone Merci* e *Vagone Passeggeri* ed una specializzazione di *Modulo di un treno*).

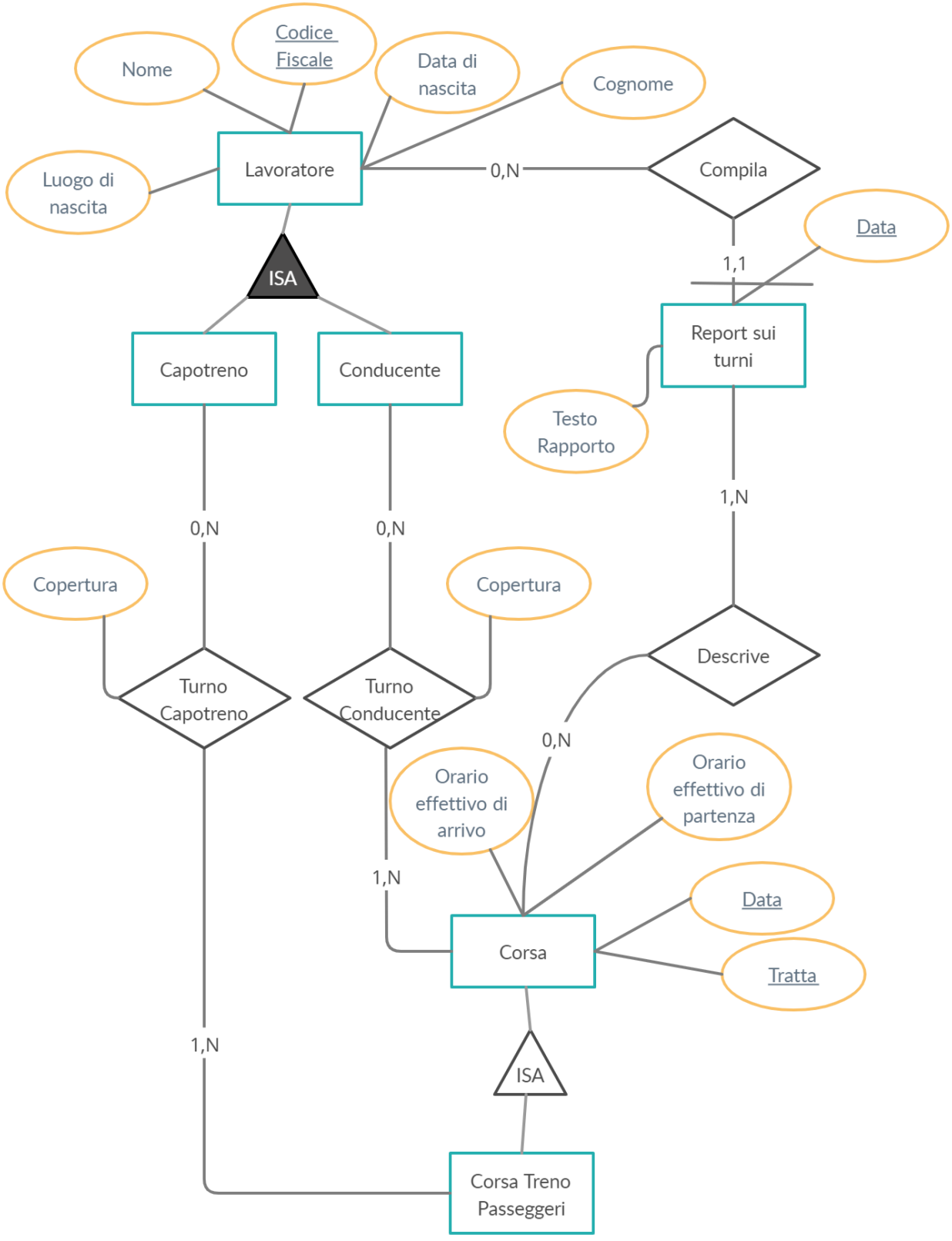


Lo schema in alto descrive la composizione delle tratte con gli orari di partenza ed arrivo previsti da ogni fermata. L'entità *Tratta* è identificata dai capolinea di partenza ed arrivo, cioè dalle associazioni *Parte* e *Termina* con una *Stazione*.

Segue l'entità *Conducente* così come descritta alla fine del secondo paragrafo.



Il paragrafo tre descrive la gestione dei turni di lavoro e dei report dei lavoratori.



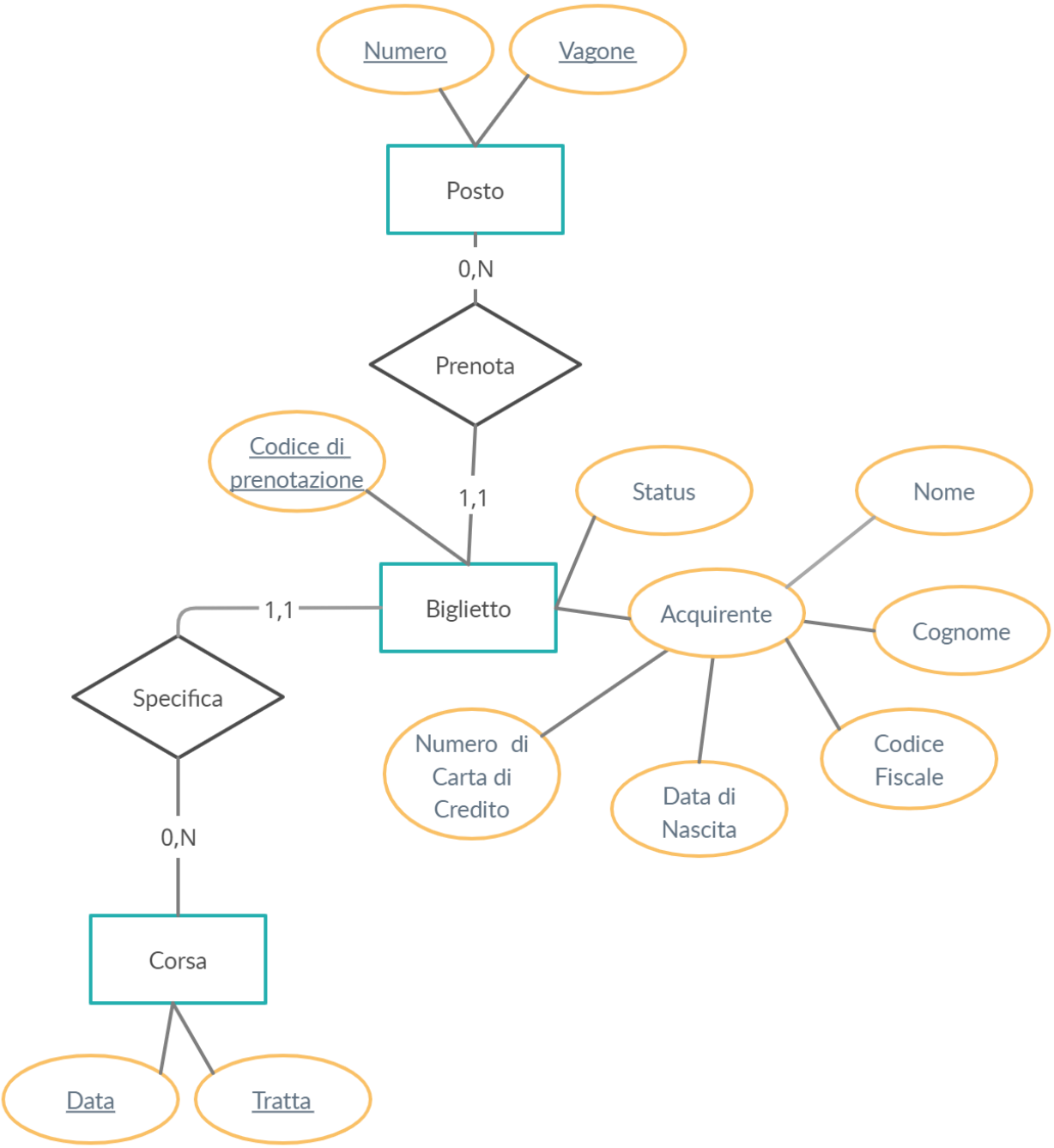
Si è identificato con l'entità *Lavoratore* il generico dipendente delle ferrovie, che va a sostituire l'entità *Conducente* definita in precedenza. Lavoratore viene specializzata in *Conducente* e *Capotreno*.

Conducente e *Capotreno* possono, a loro volta, essere assegnati a dei turni, modellati come associazioni tra l’impiegato in questione e la corsa (solo passeggeri nel caso del capotreno) su cui lavoreranno. Dato che deve essere possibile indicare se un *Lavoratore* non ha compiuto il suo turno per malattia, è stato aggiunto un attributo *Copertura* ai due tipi di turno. Dato che poi è necessario assegnare un nuovo lavoratore alla corsa in questione, a *Corsa* e *Corsa Treno Passeggeri* è stata data una cardinalità nell’associazione per i turni di 1, N.

L’entità *Corsa*, specializzata mediante una generalizzazione parziale in *Corsa Treno Passeggeri* per i turni da *Capotreno*, è identificata dalla *Tratta* percorsa e dalla *Data* in cui tale corsa avverrà. L’entità *Tratta* è stata definita in precedenza e verrà integrata con *Corsa* nell’integrazione finale, ma per questo schema parziale si è deciso di semplificarla in un attributo per migliorare la leggibilità.

L’entità *Report sui turni* è stata creata per consentire al *Lavoratore* di crearne sulle corse in cui è coinvolto. È identificata da *Data* e *Lavoratore* e, come richiesto dalla specifica, dovrà essere specificato nelle regole aziendali che non se ne può aggiungere più di una per lavoratore per settimana.

Il numero massimo di ore di lavoro e turni settimanali verrà definito mediante delle regole aziendali.



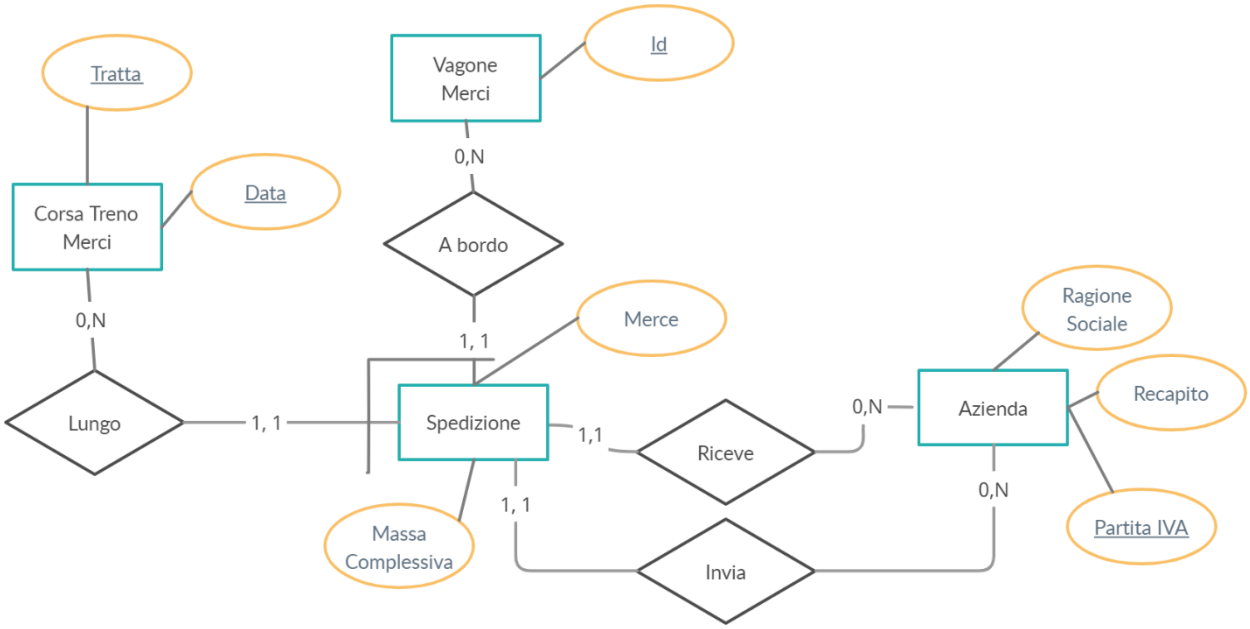
Lo schema soprastante è modella una funzione di prenotazione biglietto così come descritta nel quinto paragrafo della specifica. All'entità biglietto è stato aggiunto un attributo *status* per soddisfare la richiesta, avanzata nel settimo paragrafo, di consentire ai controllori di contrassegnare un biglietto come "valido ed utilizzato".

Nelle specifiche viene richiesto che un biglietto sia identificato per tratta, ma è anche richiesto che un controllore possa identificare un biglietto usando il *Codice di Prenotazione*. Si è pertanto scelto come attributo identificativo il *Codice di prenotazione*. Non si è associato il biglietto alla *Tratta* cui

fa riferimento, perché un biglietto non prenota un posto su di una *Tratta* ma a bordo di un *Treno Passeggeri* che percorre una *Corsa*, la quale a sua volta realizza una *Tratta*. Per questo motivo un *Biglietto* ha una associazione 1, 1 con la *Corsa* relativa.

Ad ogni *Biglietto* è poi legato un acquirente. Si è deciso di non implementare un acquirente come entità ma come attributo composto dell'entità *Biglietto* perché nella specifica è richiesto che l'acquirente inserisca i suoi dati ogni volta che prenota un biglietto, rendendo tali dati parte dell'entità Biglietto.

Le entità *Corsa* e *Posto*, già definite in precedenza, sono state qui semplificate indicando *Tratta* e *Vagone* come attributi per migliorare la leggibilità.



Questo schema modella l'invio delle merci così come descritto nel sesto paragrafo.

L'entità *Spedizione* è identificata dal *Vagone* sul quale la merce sarà trasportata e dalla *Corsa Merci* nella quale questo accadrà.

Le relazioni *Invia* e *Riceve* sono instaurate tra l'entità *Azienda* e *Vagone Merci* con cardinalità 1,1 perché il contenuto di un vagone è inviato da una sola società ad una sola società.

Le entità già modellate sono state semplificate per migliorare la leggibilità dello schema.

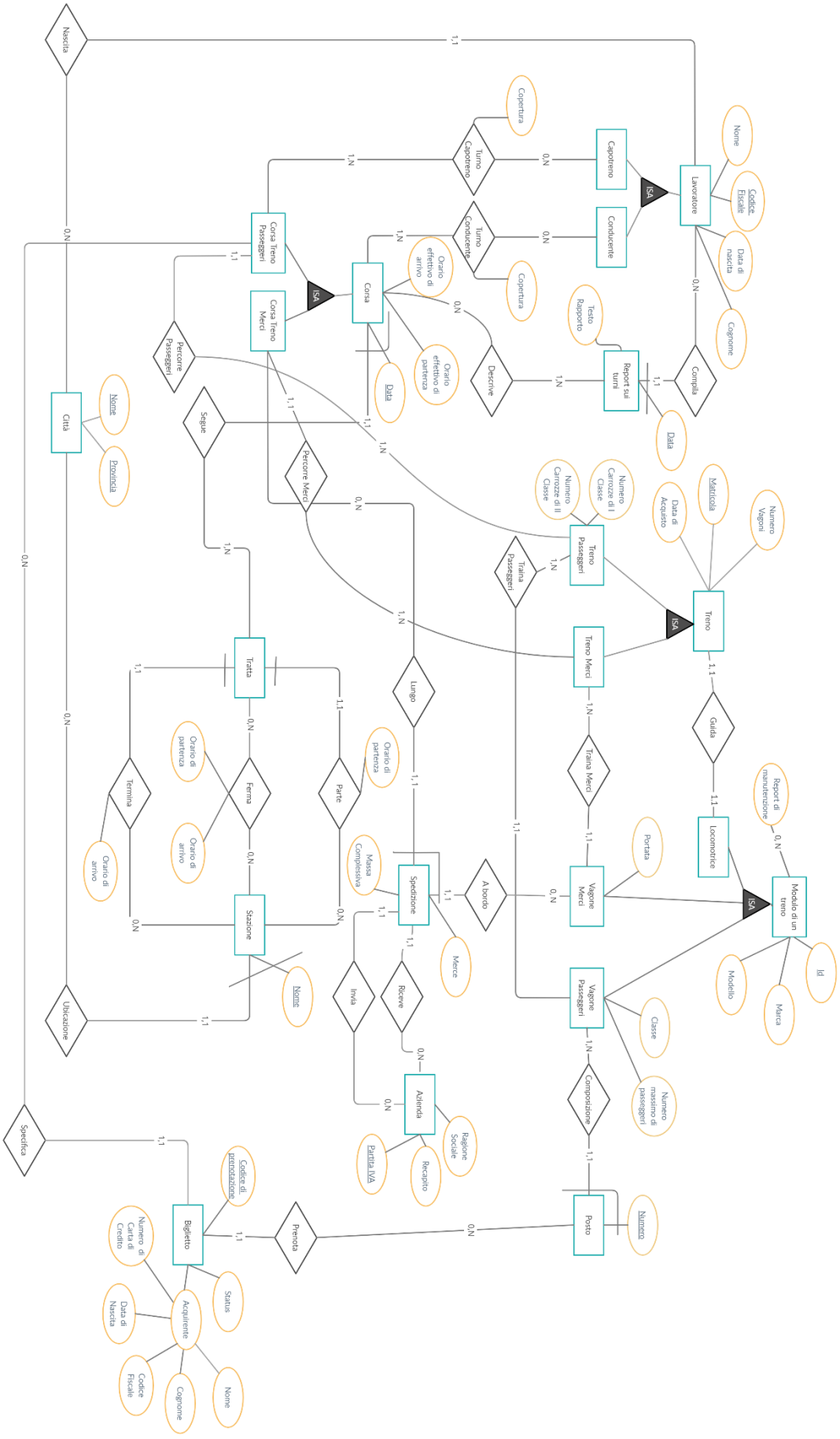
Gli schemi inseriti fino ad ora devono ora essere uniti nello schema finale.

Integrazione finale

Sono state sostituite le semplificazioni degli schemi parziali.

Sono state aggiunte le associazioni tra *Corsa Treno Passeggeri* e *Corsa Treno Merci* e, rispettivamente, *Treno Passeggeri* e *Treno Merci*.

L’entità *Lavoratore* aveva un attributo luogo di nascita. Dato che tale attributo non è altro che la città in cui è nato e che tale dato veniva già modellato per la posizione delle stazioni, è stato sostituito con una associazione *Nascita* con l’entità *Città* per migliorare la minimalità dello schema.


Nascita
1,1
0,N
Copertura
Turno Capotreno
1,N
0,N
Capotreno
ISA
Lavoratore
Nome
Codice Fiscale
Data di nascita
Cognome
Turno Conducente
Conducente
Corsa Treno Passeggeri
Corsa Treno Merci
Corsa
Orario effettivo di arrivo
Orario effettivo di partenza
Data
Descrive
Testo Rapporto
Report sui turni
Compila
Percorre Passeggeri
Percorre Merci
Segue
Città
Nome
Provincia
Numero Carrozze di I Classe
Numero Carrozze di II Classe
Data di Acquisto
Matricola
Numero Vagoni
Treno Passeggeri
Tratta
Lungo
Treno
Treno Merci
Guida
Locomotrice
Report di manutenzione
Modulo di un treno
Id
Modello
Marca
Termina
Ferma
Parte
Orario di partenza
Orario di arrivo
Massa Complessiva
Spedizione
A bordo
Vagone Merci
Portata
Vagone Passeggeri
Classe
Numero massimo di passeggeri
Composizione
Posto
Numero
Stazione
Nome
Ubicazione
Invia
Riceve
Merce
Azienda
Ragione Sociale
Partita IVA
Recapito
Specifica
Biglietto
Codice di prenotazione
Prenota
Numero di Carta di Credito
Acquirente
Status
Data di Nascita
Codice Fiscale
Cognome
Nome

Regole aziendali

Regole di vincolo:

- Un lavoratore non deve svolgere più di cinque turni settimanali.
- Un lavoratore non deve compiere più di quattro ore di lavoro per settimana.
- Un lavoratore può aggiungere al massimo un report sui turni a settimana
- Il report sui turni deve indicare corse a cui il lavoratore ha partecipato
- Una Corsa partecipa ad un solo Turno Conducente coperto
- Una Corsa Treno Passeggeri partecipa ad un solo Turno Capotreno coperto

Regole di derivazione:

- L'attributo *Numero di carrozza di I classe* si ricava dal numero di *Vagoni Passeggeri* con attributo classe uguale a I associati al *Treno Passeggeri* in questione.
- L'attributo *Numero di carrozza di II classe* si ricava dal numero di *Vagoni Passeggeri* con attributo classe uguale a II associati al *Treno Passeggeri* in questione.
- L'attributo *Numero di vagoni* di un Treno si ricava dal numero di vagoni merci associati nel caso si tratti di un treno merci, dal numero di vagoni passeggeri associati nel caso si tratti di un treno passeggeri.
- L'attributo *Numero massimo di passeggeri* si ricava dal numero di entità *posto* associate al vagone passeggeri.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Lavoratore	Impiegato delle ferrovie.	Codice Fiscale, Nome, Cognome, Data di nascita	Codice Fiscale
Conducente	Specializzazione che specifica il ruolo assunto dal Lavoratore.	Codice Fiscale, Nome, Cognome, Data di nascita	Codice Fiscale
Capotreno	Specializzazione che specifica il ruolo assunto dal Lavoratore.	Codice Fiscale, Nome, Cognome, Data di nascita	Codice Fiscale

Corsa	Effettiva percorrenza di quanto specificato in Tratta.	Data, Orario effettivo di partenza, Orario effettivo di arrivo	Data, Tratta (Segue)
Corsa Treno Passeggeri	Specializzazione di corsa che indica che il treno utilizzato per la corsa è un treno adibito al trasporto passeggeri.	Data, Orario effettivo di partenza, Orario effettivo di arrivo	Data, Tratta (Segue)
Corsa Treno Merci	Specializzazione di corsa che indica che il treno utilizzato per la corsa è un treno adibito al trasporto merci.	Data, Orario effettivo di partenza, Orario effettivo di arrivo	Data, Tratta (Segue)
Report sui turni	Report settimanale che un lavoratore può compilare sulle corse in cui è coinvolto	Data, testo rapporto	Data, Lavoratore (identificatore esterno)
Tratta	Insieme di fermate legate ad un orario che indicano un percorso che può essere svolto da una corsa.		Stazione (Parte), orario di partenza (Parte), Stazione (Termina), orario di arrivo (Termina)
Stazione	La stazione di una città in cui un treno può fermarsi durante la sua corsa	Nome	Nome, Città (Ubicazione)
Città	Può contenere delle stazioni ed è utile per identificarle.	Nome, Provincia	Nome, Provincia
Biglietto	Biglietto valido per un posto a bordo di un treno che percorre una data tratta	Status, Codice di Prenotazione, Acquirente	Codice di Prenotazione
Azienda	Società che invia o riceve merci tramite un vagone passeggeri	Partita IVA, Recapito, Ragione Sociale	Partita IVA
Spedizione	Invio da un'azienda ad un'altra tramite un vagone merci	Merce, massa complessiva	Vagone Merci (A bordo), Corsa Treno Merci (lungo)
Posto	Posto a sedere a bordo di una carrozza	Numero	Numero, Vagone Passeggeri (composizione)
Modulo di un treno	Generalizzazione dei concetti di locomotrice e treno	Id, Marca, Modello, report di manutenzione	id
Locomotrice	Specializzazione di un modulo di un	Id, Marca,	id

	treno. Elemento trainante di un treno	Modello	
Vagone Merci	Specializzazione di un modulo di un treno. Vagone adibito al trasporto merci. Parte di un treno merci	Id, Marca, Modello, portata	id
Vagone Passeggeri	Specializzazione di un modulo di un treno. Vagone adibito al trasporto passeggeri. Parte di un treno passeggeri	Id, Marca, Modello, classe, numero massimo di passeggeri	Id
Treno	Insieme di vagoni trainati da una locomotrice che percorre una tratta per mezzo di una corsa	Matricola, Numero vagoni, Data di acquisto	Matricola
Treno passeggeri	Specializzazione di un treno. Adibito al trasporto passeggeri	Matricola, Numero vagoni, Data di acquisto, Numero carrozze di prima classe, numero carrozze di seconda classe	Matricola
Treno merci	Specializzazione di un treno. Adibito al trasporto passeggeri	Matricola, Numero vagoni, Data di acquisto	Matricola

4. Progettazione logica

Volume dei dati

I dati sono calcolati sul volume atteso in un quattro settimane (un mese).

Concetto nello schema	Tipo[1]	Volume atteso
Lavoratore	E	2.016 (stimati per eccesso di circa il 20% rispetto al numero di corse settimanali considerando che un lavoratore può partecipare ad un massimo di 5 corse)
Turno Conducente	R	27.720 (turni da conducente aspettati in quattro settimane in base al numero di corse più circa un 10% di turni sostitutivi)
Turno Capotreno	R	9240 (turni da capotreno aspettati in quattro settimane in base al numero di corse più circa un 10% di turni sostitutivi)
Conducente	E	1.344
Capotreno	E	672
Corsa	E	25.200 (stimando che un treno possa compiere in un giorno in media 3 corse)
Corsa Treno Passeggeri	E	6.300
Corsa Treno Merci	E	18.900
Compila	R	8064 (ha un rapport 1:1 con *Report sui turni* e pertanto ha la stessa cardinalità)
Report sui turni	E	8.064 (stimando che al massimo ogni lavoratore stili un report a settimana per 4 settimane (dati di un mese))
Descrive	R	33.600 (stimando che tutte le corse di un mese vengano descritte in un report da conducente e capotreno)
Segue	R	25.200 (ha un rapport 1:1 con *Corsa* e pertanto ha la stessa cardinalità)
Percorre Passeggeri	R	6.300 (ha un rapport 1:1 con *Corsa Treno Passeggeri* e pertanto ha la

[1] Indicare con E le entità, con R le relazioni

		stessa cardinalità)
Percorre Merci	R	18.900 (ha un rapport 1:1 con *Corsa treno merci* e pertanto ha la stessa cardinalità)
Tratta	E	40
Parte	R	40 (ha un rapport 1:1 con *Tratta* e pertanto ha la stessa cardinalità)
Ferma	R	400 (stimando che un treno faccia in media 10 fermate)
Termina	R	40 (ha un rapport 1:1 con *Tratta* e pertanto ha la stessa cardinalità)
Stazione	E	200 (approssimazione sul numero di stazioni in italia)
Ubicazione	R	200 (ha un rapport 1:1 con *Stazione* e pertanto ha la stessa cardinalità)
Città	E	7.904 (numero di comuni italiani)
Nascita	R	2.016 (ha un rapport 1:1 con *Lavoratore* e pertanto ha la stessa cardinalità)
Specifica	R	3.360.000 (ha un rapport 1:1 con *Biglietto* e pertanto ha la stessa cardinalità)
Biglietto	E	3.360.000 (stimando un riempimento di circa il 70% delle *corse treno passeggeri*)
Prenota	R	3.360.000 (ha un rapport 1:1 con *Biglietto* e pertanto ha la stessa cardinalità)
Posto	E	40.000 (considerando 50 posti per *vagone passeggeri*)
Composizione	R	40.000 (ha un rapport 1:1 con *Posto* e pertanto ha la stessa cardinalità)
Vagone Passeggeri	E	800 (stimando 8 vagoni per treno passeggeri)
Vagone Merci	E	2.000 (stimando 10 vagoni per treno merci)
Lungo	R	117.600 (ha un rapport 1:1 con *Spedizione* e pertanto ha la stessa cardinalità)

A bordo	R	117.600 (ha un rapport 1:1 con *Spedizione* e pertanto ha la stessa cardinalità)
Spedizione	E	117.600 (stimando un 70% di riempimento)
Invia	R	117.600 (ha un rapport 1:1 con *Invio* e pertanto ha la stessa cardinalità)
Riceve	R	117.600 (ha un rapport 1:1 con *Invio* e pertanto ha la stessa cardinalità)
Azienda	E	235.200 (stimando, nel peggiore dei casi, che un'azienda o invii merci o le riceva e partecipi ad una sola spedizione)
Traina Passeggeri	R	800 (ha un rapport 1:1 con *Vagone Passeggeri* e pertanto ha la stessa cardinalità)
Traina Merci	R	2.000 (ha un rapport 1:1 con *Vagone Merci* e pertanto ha la stessa cardinalità)
Locomotrice	E	300 (ha un rapport 1:1 con *Treno* e pertanto ha la stessa cardinalità)
Modulo di un Treno	E	3.100 (somma di *locomotrice* e vagoni)
Treno	E	300 (somma di treno passeggeri e treno merci)
Treno Passeggeri	E	100
Treno Merci	E	200
Guida	R	300 (ha un rapport 1:1 con *Treno* e pertanto ha la stessa cardinalità)

Tavola delle operazioni

Tutte le frequenze vengono calcolate approssimando per eccesso il risultato ottenuto dai volumi attesi

Cod.	Descrizione	Frequenza attesa
01	Acquista un biglietto	120.00 al giorno
02	Aggiungi spedizione	4.200 al giorno
03	Aggiungi report manutenzione	110 al giorno

04	Aggiungi corsa treno merci	675 al giorno
05	Aggiungi una corsa treno passeggeri	225 al giorno
06	Assegna turno conducente	27720 il primo del mese
07	Assegna turno capotreno	9240 il primo del mese
08	Sostituisci conducente	50 al giorno (5%)
09	Sostituisci capotreno	17 al giorno (5%)
10	Genera report sui turni di lavoro	288 al giorno
11	Convalida biglietto	120.000 al giorno
12	Aggiungi un'azienda	84 al giorno
13	Aggiungi un conducente	5 al mese
14	Aggiungi un capotreno	2 al mese
15	Aggiungi un treno passeggeri	1 al mese
16	Aggiungi un treno merci	1 al mese
17	Visualizza storico di manutenzione treno merci	1 al mese
18	Visualizza storico di manutenzione treno passeggeri	1 al mese
19	Verifica biglietto	120.000 al giorno

Costo delle operazioni

Operazione 01: Acquista un biglietto. Costo: 9

Concetto	Costrutto	Accessi	Tipo
Biglietto	E	1	S
Specifica	R	1	S
Corsa Treno Passeggeri	E	1	L
Corsa	E	1	L
Prenota	R	1	S
Posto	E	1	L

Operazione 02: Aggiungi spedizione. Costo: 14

Concetto	Costrutto	Accessi	Tipo
Azienda	E	2	L
Invia	R	1	S
Riceve	R	1	S
Spedizione	E	1	S
Lungo	R	1	S
Corsa Treno Merci	E	1	L
Corsa	E	1	L
A bordo	R	1	S
Vagone Merci	E	1	L

Operazione 03: Aggiungi report manutenzione. Costo: 2

Concetto	Costrutto	Accessi	Tipo
Modulo di un treno	E	1	S

Operazione 04: Aggiungi una corsa treno merci. Costo: 11

Concetto	Costrutto	Accessi	Tipo
Corsa	E	1	S
Corsa treno merci	E	1	S
Percorre Merci	R	1	S
Treno	E	1	L
Treno merci	E	1	L
Segue	R	1	S
Tratta	E	1	L

Operazione 05: Aggiungi una corsa treno passeggeri. Costo: 11

Concetto	Costrutto	Accessi	Tipo

Corsa	E	1	S
Corsa treno passeggeri	E	1	S
Percorre Passeggeri	R	1	S
Treno	E	1	L
Treno passeggeri	E	1	L
Segue	R	1	S
Tratta	E	1	L

Operazione 06: Assegna turno conducente. Costo: 5

Concetto	Costrutto	Accessi	Tipo
Turno Conducente	R	1	S
Lavoratore	E	1	L
Conducente	E	1	L
Corsa	E	1	L

Operazione 07: Assegna turno capotreno. Costo: 6

Concetto	Costrutto	Accessi	Tipo
Turno Capotreno	R	1	S
Lavoratore	E	1	L
Capotreno	E	1	L
Corsa	E	1	L
Corsa Treno Passeggeri	E	1	L

Operazione 08: Sostituisci conducente. Costo: 9

Concetto	Costrutto	Accessi	Tipo

Lavoratore	E	2	L
Conducente	E	2	L
Turno Conducente	E	2	S (segno il precedente come malato e creo il nuovo turno sostitutivo)
Corsa	E	1	L

Operazione 09: Sostituisci capotreno. Costo: 10

Concetto	Costrutto	Accessi	Tipo
Lavoratore	E	2	L
Capotreno	E	2	L
Turno Capotreno	E	2	S (segno il precedente come malato e creo il nuovo turno sostitutivo)
Corsa	E	1	L
Corsa Treno passeggeri	E	1	L

Operazione 10: Genera report sui turni di lavoro. Costo: 20

Concetto	Costrutto	Accessi	Tipo
Lavoratore	E	1	L
Compila	R	1	S
Report sui turni	E	1	S
Descrive	R	5	S
Corsa	E	5	L

Operazione 11: Convalida biglietto. Costo: 2

Concetto	Costrutto	Accessi	Tipo
Biglietto	E	1	S

Operazione 12: Aggiungi un’azienda. Costo: 2

Concetto	Costrutto	Accessi	Tipo
Azienda	E	1	S

Operazione 13: Aggiungi un conducente. Costo: 4

Concetto	Costrutto	Accessi	Tipo
Lavoratore	E	1	S
Conducente	E	1	S

Operazione 14: Aggiungi un capotreno. Costo: 4

Concetto	Costrutto	Accessi	Tipo
Lavoratore	E	1	S
Capotreno	E	1	S

Operazione 15: Aggiungi un treno passeggeri. Costo: 2.058

Concetto	Costrutto	Accessi	Tipo
Treno	E	1	S
Guida	R	1	S
Modulo di un treno	E	9	S
Locomotrice	E	1	S
Treno passeggeri	E	1	S
Traina Passeggeri	R	8	S
Vagone Passeggeri	E	8	S
Composizione	R	500	S
Posto	E	500	S

Operazione 16: Aggiungi un treno merci. Costo: 70

Concetto	Costrutto	Accessi	Tipo
Treno	E	1	S
Guida	R	1	S
Modulo di un treno	E	11	S
Locomotrice	E	1	S
Treno merci	E	1	S
Traina merci	R	10	S
Vagone merci	E	10	S

Operazione 17: Visualizza storico manutenzioni treno merci. Costo: 35

Concetto	Costrutto	Accessi	Tipo
Treno	E	1	L
Treno Merci	E	1	L
Guida	R	1	L
Modulo di un treno	E	11	L
Locomotrice	E	1	L
Traina merci	R	10	L
Vagone Merci	E	10	L

Operazione 18: Visualizza storico manutenzioni treno passeggeri. Costo: 29

Concetto	Costrutto	Accessi	Tipo
Treno	E	1	L
Treno Passeggeri	E	1	L
Guida	R	1	L
Modulo di un treno	E	9	L
Locomotrice	E	1	L

Traina Passeggeri	R	8	L
Vagone Passeggeri	E	8	L

Operazione 19: Verifica Biglietto. Costo: 6

Concetto	Costrutto	Accessi	Tipo
Biglietto	E	1	L
Prenota	R	1	L
Posto	E	1	L
Specifica	R	1	L
Corsa	E	1	L
Corsa Treno Passeggeri	E	1	L

Ristrutturazione dello schema E-R

Le ridondanze presenti nella progettazione logica sono il numero di vagoni (Treno), il numero di carrozze di prima e seconda classe (Treno passeggeri) ed il numero massimo di passeggeri in un vagone (vagone passeggeri). Si tratta di ridondanze che vengono create all'inizializzazione delle entità ad esse collegate e che non vengono aggiornate. Consumano dello spazio nella base di dati ma, essendo esplicitamente richieste nella specifica, è opportuno lasciarle in quanto non sono onerose da mantenere.

Bisogna eliminare le generalizzazioni Lavoratore, Corsa, Modulo di un Treno e Treno.

Nel primo caso si sceglie di assorbire i figli nel genitore con l'aggiunta dell'attributo ruolo. Una volta deciso di mantenere la generalizzazione *Lavoratore*, si possono unire le entità *Turno Capotreno* e *Turno Conducente* in un'entità *Turno*.

L'entità corsa viene legata ai figli mediante un'associazione, creando due entità deboli.

Nel caso di Modulo di un treno e dei suoi figli, si assorbe la generalizzazione nei figli, mantenendo dunque solo Locomotrice, Vagone Passeggeri e Vagone Merci. Di conseguenza si crea un tipo diverso di *Report di manutenzione* per ognuna delle entità figlio.

L'attributo multi-valore Report di manutenzione viene trasformato in una nuova entità debole per ognuno dei figli di modulo di un treno.

L'attributo multi-valore Acquirente viene invece mantenuto all'interno dell'entità biglietto, diviso negli attributi *nome, cognome, codice fiscale, data di nascita, numero cc*. L'accesso all'entità Biglietto, infatti, è fatta in caso di acquisto, verifica e convalida. In due di questi tre casi, acquisto e verifica, è necessario scrivere o reperire anche le informazioni relative ai dati dell'acquirente, pertanto non si ritiene opportuno partizionare l'entità.

Trasformazione di attributi e identificatori

L'entità *Città* opera da identificatore esterno per l'entità Stazione ed è legata tramite l'associazione 1, 1 *Nascita* all'entità *Lavoratore*. *Città* contiene solo due attributi identificativi, i quali andranno ripetuti nelle entità a cui è collegata. Questo significa che, dato che i dati contenuti nell'entità vanno duplicati in entrambi i casi, si può eliminare completamente l'entità per risparmiare spazio aggiungendo gli attributi *Città* e *Provincia* a *Stazione* e gli attributi *Città di nascita* e *Provincia di nascita* a *Lavoratore*.

Dato che *Report sui Turni* è un'entità debole, le si aggiunge l'attributo codice fiscale del lavoratore collegato con il nome *Lavoratore*.

All'entità *Posto* si aggiunge l'Id del vagone passeggeri di riferimento con il nome *Vagone Passeggeri.*

L'entità *Stazione* è identificata da tre attributi, il che implica che per dover aggiungere tre attributi in ogni istanza di *Ferma* e sei in ogni *Tratta*. Per risparmiare memoria di decide di aggiungere un attributo identificativo numerico chiamato *Codice*.

Ogni *Tratta*, nonostante l'aggiunta di un id per le stazioni, è identificata da quattro attributi (id e orario dei capolinea di partenza e arrivo), che andrebbero aggiunti in *Ferma* ed in *Corsa*. Per questo motivo anche in *Tratta* si aggiunge un nuovo attributo identificativo *Codice*.

Nell'entità *Corsa* si aggiunge ora un attributo *Tratta* contenente il codice della Tratta relativa.

Nell'entità *Spedizione* si aggiungono gli attributi identificativi della *Corsa Treno Merci* relativa con i nomi *Data Corsa* e *Tratta*.

Traduzione di entità e associazioni

AZIENDA (Partita IVA, Recapito, Ragione Sociale)

BIGLIETTO (Codice Di Prenotazione, Numero Posto, Vagone Passeggeri, Status, Data Corsa, Tratta, Codice Fiscale, Nome, Cognome, Data Di Nascita, Numero CC)

CORSA (Data, Tratta, Orario Effettivo di Partenza, Orario Effettivo di Arrivo)

CORSA REPORT (Lavoratore, Data Report, Numero, Data Corsa, Tratta)

CORSA TRENO MERCI (Data, Tratta, Treno Merci)

CORSA TRENO PASSEGGERI (Data, Tratta, Treno Passeggeri)

FERMA (Orario partenza, Orario arrivo, Tratta, Stazione)

LAVORATORE (Codice fiscale, Nome, Cognome, Data di Nascita, Città di Nascita, Provincia di Nascita, Ruolo)

LOCOMOTRICE (Id, Marca, Modello);

POSTO (Numero, Vagone Passeggeri);

REPORT MANUTENZIONE LOCOMOTRICE (Locomotrice, Timestamp, Testo)

REPORT MANUTENZIONE VAGONE PASSEGGERI (Vagone Passeggeri, Timestamp, Testo)

REPORT MANUTENZIONE VAGONE MERCI (Vagone Merci, Timestamp, Testo)

REPORT SUI TURNI (Lavoratore, Data, Testo Rapporto)

SPEDIZIONE (Azienda Mittente, Azienda Destinataria, Merce, Massa Complessiva, Vagone Merci, Data Corsa, Tratta)

STAZIONE (Codice, Nome, Città, Provincia)

TRATTA (Codice, Capolinea Partenza, Capolinea Arrivo, Orario Partenza, Orario Arrivo)

TRENO PASSEGGERI (Matricola, Numero Vagoni, Data Di Acquisto, Numero Carrozze I Classe, Numero Carrozze II Classe, Locomotrice)

TRENO MERCI (Matricola, Numero Vagoni, Data Di Acquisto, Locomotrice)

TURNO (Lavoratore, Data Corsa, Tratta, Copertura)

VAGONE MERCI (Id, Marca, Modello, Portata, Treno)

VAGONE PASSEGGERI (Id, Marca, Modello, Classe, Numero Massimo di Passeggeri, Treno)

Vincoli di integrità referenziale:

- BIGLIETTO (Numero posto, Vagone Passeggeri) dipende da POSTO (Numero, Vagone Passeggeri)
- BIGLIETTO (Data Corsa, Tratta) dipende da CORSA TRENO PASSEGGERI (Data, Tratta)
- CORSA(Tratta) dipende da TRATTA (Codice)
- CORSA REPORT (Lavoratore, Data Report) dipende da REPORT SUI TURNI (Lavoratore, Data)
- CORSA REPORT (Data Corsa, Tratta) dipende da CORSA (Data, Tratta)
- CORSA TRENO MERCI (Data, Tratta) dipende da CORSA (Data, Tratta)
- CORSA TRENO MERCI (Treno Merci) dipende da TRENO MERCI (Matricola)
- CORSA TRENO PASSEGGERI (Data, Tratta) dipende da CORSA (Data, Tratta)
- CORSA TRENO PASSEGGERI (Treno Passeggeri) dipende da TRENO PASSEGGERI (Matricola)
- FERMA(Tratta) dipende da TRATTA(Codice)
- FERMA(Stazione) dipende da STAZIONE(Codice)
- POSTO (Vagone Passeggeri) dipende da VAGONE PASSEGGERI (Id)
- REPORT DI MANUTENZIONE LOCOMOTRICE (Locomotrice) dipende da LOCOMOTRICE(Id)
- REPORT DI MANUTENZIONE VAGONE PASSEGGERI (Vagone Passeggeri) dipende da VAGONE PASSEGGERI(Id)
- REPORT DI MANUTENZIONE VAGONE MERCI (Vagone Merci) dipende da VAGONE MERCI(Id)
- REPORT SUI TURNI(Lavoratore) dipende da LAVORATORE (Codice Fiscale)
- SPEDIZIONE (Azienda Mittente) dipende da AZIENDA (Partita IVA)
- SPEDIZIONE (Azienda Destinataria) dipende da AZIENDA (Partita IVA)
- SPEDIZIONE (Vagone Merci) dipende da VAGONE MERCI (Id)
- SPEDIZIONE (Data Corsa, Tratta) dipende da CORSA TRENO MERCI (Data, Tratta)
- TRATTA (Capolinea Partenza) dipende da STAZIONE (Codice)
- TRATTA (Capolinea Arrivo) dipende da STAZIONE (Codice)
- TRENO MERCI (Locomotrice) dipende da LOCOMOTRICE(Id)
- TRENO PASSEGGERI (Locomotrice) dipende da LOCOMOTRICE(Id)

- TURNO (Lavoratore) dipende da LAVORATORE (Codice Fiscale)
- TURNO (Data Corsa, Tratta) dipende da CORSA (Data, Tratta)
- VAGONE PASSEGGERI(Treno) dipende da TRENO PASSEGGERI (Matricola)
- VAGONE MERCI(Treno) dipende da TRENO MERCI (Matricola)

Normalizzazione del modello relazionale

Il modello relazionale è già in forma normale 3NF.

5. Progettazione fisica

Utenti e privilegi

Utenti:

- Lavoratore
- Gestore del servizio
- Acquirente
- Controllore
- Addetto alla manutenzione

Privilegi:

- Lavoratore:
  - Può generare un report sui turni di lavoro
- Gestore del servizio:
  - Aggiungi spedizione
  - Aggiungi corsa treno merci
  - Aggiungi corsa treno passeggeri
  - Sostituisci lavoratore
  - Aggiungi un'azienda
  - Aggiungi un treno passeggeri
  - Aggiungi un treno merci
  - Aggiungi lavoratore
  - Crea utente
  - Inserisci orari corsa
  - Visualizza storico di manutenzione treno merci
  - Visualizza storico di manutenzione treno passeggeri
- Acquirente:
  - Può acquistare un biglietto
- Controllore:
  - Può verificare le informazioni di un biglietto
  - Può convalidare un biglietto
- Addetto alla manutenzione:

- Può aggiungere un report di manutenzione per locomotive, vagoni merci e vagoni passeggeri

Strutture di memorizzazione

Tabella Azienda		
Attributo	**Tipo di dato**	**Attributi[2]**
Partita_IVA	VARCHAR(11)	PK NN
Recapito	VARCHAR(45)	NN UQ
Ragione_Sociale	VARCHAR(45)	NN UQ

Tabella Biglietto		
Attributo	**Tipo di dato**	**Attributi[3]**
Codice_di_prenotazione	INT	PK NN UN AI
Numero_Posto	INT	NN UN
Vagone_Passeggeri	INT	NN UN
Data_Corsa	DATE	NN
Tratta	INT	NN UN
Status	ENUM('non-convalidato', 'convalidato')	
Codice_Fiscale	VARCHAR(16)	NN
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
Data_di_nascita	DATE	NN
Numero_CC	VARCHAR(16)	

[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.

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

Tabella Corsa		
Attributo	**Tipo di dato**	**Attributi[4]**
Data	DATE	PK NN
Tratta	INT	PK NN UN
Orario_effettivo_di_partenza	DATETIME	
Orario_effettivo_di_arrivo	DATETIME	

Tabella Corsa_Report		
Attributo	**Tipo di dato**	**Attributi[5]**
Lavoratore	VARCHAR(16)	PK NN
Data_Report	DATE	PK NN
Numero	INT	PK NN UN
Data_Corsa	DATE	NN
Tratta	INT	NN UN

Tabella Corsa_Treno_Merci		
Attributo	**Tipo di dato**	**Attributi[6]**
Data_Corsa	DATE	PK NN
Tratta	INT	PK NN UN
Treno_Merci	VARCHAR(4)	NN

Tabella Corsa_Treno_Passeggeri		
Attributo	**Tipo di dato**	**Attributi[7]**

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

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

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

Data_Corsa	DATE	PK NN
Tratta	INT	PK NN UN
Treno_Passeggeri	VARCHAR(4)	NN

Tabella Ferma		
Attributo	**Tipo di dato**	**Attributi[8]**
Orario_Partenza	TIME	NN
Orario_Arrivo	TIME	NN
Tratta	INT	PK NN UN
Stazione	INT	PK NN UN

Tabella Lavoratore		
Attributo	**Tipo di dato**	**Attributi[9]**
Codice_Fiscale	VARCHAR(16)	PK NN
Nome	VARCHAR(45)	NN
Cognome	VARCHAR(45)	NN
Data_di_nascita	DATE	NN
Città_di_nascita	VARCHAR(45)	NN
Ruolo	ENUM('capotreno', 'conducente')	NN
Provincia_di_Nascita	VARCHAR(2)	NN

Tabella Locomotrice

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

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

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

Attributo	Tipo di dato	Attributi[10]
Id	INT	PK NN UN AI
Marca	VARCHAR(45)	NN
Modello	VARCHAR(45)	NN

Tabella Posto		
Attributo	**Tipo di dato**	**Attributi[11]**
Numero	INT	PK NN UN AI
Vagone_Passeggeri	INT	PK NN UN

Tabella Report_Manutenzione_Locomotrice		
Attributo	**Tipo di dato**	**Attributi[12]**
Locomotrice	INT	PK NN UN
Testo	VARCHAR(200)	NN
Timestamp	TIMESTAMP	PK NN

Tabella Report_Manutenzione_Vagone_Merci		
Attributo	**Tipo di dato**	**Attributi[13]**
Vagone_Merci	INT	PK NN UN
Testo	VARCHAR(200)	NN
Timestamp	TIMESTAMP	PK NN

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

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

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

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

Tabella Report_Manutenzione_Vagone_Passeggeri		
Attributo	**Tipo di dato**	**Attributi[14]**
Vagone_Paseggeri	INT	PK NN UN
Testo	VARCHAR(200)	NN
Timestamp	TIMESTAMP	PK NN

Tabella Report_sui_turni		
Attributo	**Tipo di dato**	**Attributi[15]**
Lavoratore	VARCHAR(16)	PK NN
Data	DATE	PK NN
Testo_Rapporto	VARCHAR(500)	NN

Tabella Spedizione		
Attributo	**Tipo di dato**	**Attributi[16]**
Azienda_Mittente	VARCHAR(11)	NN
Azienda_Destinataria	VARCHAR(11)	NN
Merce	VARCHAR(45)	NN
Massa_Complessiva	INT	NN UN
Vagone_Merci	INT	PK NN UN
Data_Corsa	DATE	PK NN
Tratta	INT	PK NN UN

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

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

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

Tabella Stazione		
Attributo	**Tipo di dato**	**Attributi**[17]
Codice	INT	PK NN UN AI
Nome	VARCHAR(45)	NN
Città	VARCHAR(45)	NN
Provincia	VARCHAR(2)	NN

Tabella Tratta		
Attributo	**Tipo di dato**	**Attributi**[18]
Codice	INT	PK NN UN AI
Capolinea_di_Partenza	INT	NN UN
Capolinea_di_Arrivo	INT	NN UN
Orario_Partenza	TIME	NN
Orario_Arrivo	TIME	NN

Tabella Treno_Merci		
Attributo	**Tipo di dato**	**Attributi**[19]
Matricola	VARCHAR(4)	PK NN
Numero_Vagoni	INT	NN UN
Data_di_Acquisto	DATE	NN
Locomotrice	INT	NN UN

Tabella Treno_Passeggeri

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

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

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

Attributo	Tipo di dato	Attributi[20]
Matricola	VARCHAR(4)	PK NN
Numero_Vagoni	INT	NN UN
Data_di_Acquisto	DATE	NN
Locomotrice	INT	NN UN
Numero_Carrozze_di_I_Classe	INT	NN UN
Numero_Carrozze_di_II_Classe	INT	NN UN

Tabella Turno		
Attributo	**Tipo di dato**	**Attributi[21]**
Lavoratore	VARCHAR(16)	PK NN
Data_Corsa	DATE	PK NN
Tratta	INT	PK NN UN
Copertura	ENUM('coperto', 'malattia')	

Tabella Utenti		
Attributo	**Tipo di dato**	**Attributi[22]**
Username	VARCHAR(45)	PK NN
Password	Char(32)	NN
Ruolo	ENUM('Gestore_del_servizio', 'Lavoratore', 'Addetto_alla_manutenzione', 'Controllore', 'Acquirente')	NN

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

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

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

Tabella Vagone_Merci		
Attributo	**Tipo di dato**	**Attributi[23]**
Id	INT	PK NN UN AI
Marca	VARCHAR(45)	NN
Modello	VARCHAR(45)	NN
Portata	INT	NN UN
Treno	VARCHAR(4)	NN

Tabella Vagone_Passeggeri		
Attributo	**Tipo di dato**	**Attributi[24]**
Id	INT	PK NN UN AI
Marca	VARCHAR(45)	NN
Modello	VARCHAR(45)	NN
Classe	ENUM('1', '2')	NN
Numero_Massimo_di_Passeggeri	INT	NN UN
Treno	VARCHAR(4)	NN

Indici

Tabella Azienda	
Indice PRIMARY	**Tipo[25]:**
Partita_IVA	PR

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

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

[25] IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella Azienda	
Indice Recapito_UNIQUE	**Tipo[26]:**
Recapito	UQ

Tabella Azienda	
Indice Ragione_Sociale_UNIQUE	**Tipo[27]:**
Ragione_Sociale	UQ

Ragione Sociale e Recapito sono elementi distintivi di un'azienda

Tabella Biglietto	
Indice PRIMARY	**Tipo[28]:**
Codice_di_prenotazione	PR

Tabella Biglietto	
Indice fk_Biglietto_1_idx	**Tipo[29]:**
Numero_Posti	IDX
Vagone_Passeggeri	IDX

Tabella Biglietto	
Indice fk_Biglietto_2_idx	**Tipo[30]:**
Data_Corsa	IDX
Tratta	IDX

[26] IDX = index, UQ = unique, FT = full text, PR = primary.

[27] IDX = index, UQ = unique, FT = full text, PR = primary.

[28] IDX = index, UQ = unique, FT = full text, PR = primary.

[29] IDX = index, UQ = unique, FT = full text, PR = primary.

[30] IDX = index, UQ = unique, FT = full text, PR = primary.

I due indici soprastanti sono usati per le foreign key della tabella Biglietto

Tabella Corsa	
Indice PRIMARY	**Tipo[31]:**
Data	PR
Tratta	PR

Tabella Corsa	
Indice fk_Corsa_1_idx	**Tipo[32]:**
Tratta	IDX

Indice usato per la foreign key della tabella Corsa

Tabella Corsa_Report	
Indice PRIMARY	**Tipo[33]:**
Lavoratore	PR
Data Report	PR
Numero	PR

Tabella Corsa_Report	
Indice fk_corsa_2_idx	**Tipo[34]:**

[31] IDX = index, UQ = unique, FT = full text, PR = primary.

[32] IDX = index, UQ = unique, FT = full text, PR = primary.

[33] IDX = index, UQ = unique, FT = full text, PR = primary.

[34] IDX = index, UQ = unique, FT = full text, PR = primary.

Tratta	IDX
Data_Corsa	

Indice usato per la foreign key della tabella Corsa_Report

Tabella Corsa_Treno_Merci	
Indice PRIMARY	**Tipo[35]:**
Data_Corsa	PR
Tratta	PR

Tabella Corsa_Treno_Merci	
Indice fk_Corsa_Treno_Merci_1_idx	**Tipo[36]:**
Data_Corsa	IDX
Tratta	IDX

Tabella Corsa_Treno_Merci	
Indice fk_Corsa_Treno_Merci_2_idx	**Tipo[37]:**
Treno_Merci	IDX

Indice per la Foreign Key per la tabella Corsa_Treno_Merci

Tabella Corsa_Treno_Passeggeri	
Indice PRIMARY	**Tipo[38]:**

[35] IDX = index, UQ = unique, FT = full text, PR = primary.

[36] IDX = index, UQ = unique, FT = full text, PR = primary.

[37] IDX = index, UQ = unique, FT = full text, PR = primary.

[38] IDX = index, UQ = unique, FT = full text, PR = primary.

Data_Corsa	PR
Tratta	PR

Tabella Corsa_Treno_Passeggeri	
Indice fk_Corsa_Treno_Passeggeri_2_idx	**Tipo[39]:**
Treno_Passeggeri	IDX

Indice per la Foreign Key per la tabella Corsa_Treno_Passeggeri

Tabella Ferma	
Indice PRIMARY	**Tipo[40]:**
Tratta	PR
Stazione	PR

Tabella Ferma	
Indice fk_Ferma_2_idx	**Tipo[41]:**
Stazione	IDX

Indice per la Foreign Key per la tabella Stazione

Tabella Lavoratore	
Indice PRIMARY	**Tipo[42]:**
Codice_Fiscale	PR

[39] IDX = index, UQ = unique, FT = full text, PR = primary.

[40] IDX = index, UQ = unique, FT = full text, PR = primary.

[41] IDX = index, UQ = unique, FT = full text, PR = primary.

[42] IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella Locomotrice	
Indice PRIMARY	**Tipo[43]:**
Id	PR

Tabella Posto	
Indice PRIMARY	**Tipo[44]:**
Numero	PR
Vagone_Passeggeri	PR

Tabella Posto	
Indice fk_Posto_1_idx	**Tipo[45]:**
Vagone_Passeggeri	IDX

Indice per la Foreign Key per la tabella Posto

Tabella Report_Manutenzione_Locomotrice	
Indice PRIMARY	**Tipo[46]:**
Locomotrice	PR
Timestamp	PR

Tabella Report_Manutenzione_Vagone_Merci	
Indice PRIMARY	**Tipo[47]:**

[43] IDX = index, UQ = unique, FT = full text, PR = primary.

[44] IDX = index, UQ = unique, FT = full text, PR = primary.

[45] IDX = index, UQ = unique, FT = full text, PR = primary.

[46] IDX = index, UQ = unique, FT = full text, PR = primary.

[47] IDX = index, UQ = unique, FT = full text, PR = primary.

Vagone_Merci	PR
Timestamp	PR

Tabella Report_Manutenzione_Vagone_Passeggeri	
Indice PRIMARY	**Tipo[48]:**
Vagone_Passeggeri	PR
Timestamp	PR

Tabella Report_sui_turni	
Indice PRIMARY	**Tipo[49]:**
Lavoratore	PR
Data	PR

Tabella Spedizione	
Indice PRIMARY	**Tipo[50]:**
Vagone_Merci	PR
Data_Corsa	PR
Tratta	PR

Tabella Spedizione	
Indice fk_Spedizione_1_idx	**Tipo[51]:**
Azienda_Mittente	IDX

[48] IDX = index, UQ = unique, FT = full text, PR = primary.

[49] IDX = index, UQ = unique, FT = full text, PR = primary.

[50] IDX = index, UQ = unique, FT = full text, PR = primary.

[51] IDX = index, UQ = unique, FT = full text, PR = primary.

Tabella Spedizione	
Indice fk_Spedizione_2_idx	**Tipo[52]:**
Azienda_Destinataria	IDX

Tabella Spedizione	
Indice fk_Spedizione_3_idx	**Tipo[53]:**
Vagone_Merci	IDX

Indici per le Foreign Key per la tabella Spedizione

Tabella Stazione	
Indice PRIMARY	**Tipo[54]:**
Codice	PR

Tabella Tratta	
Indice PRIMARY	**Tipo[55]:**
Codice	PR

Tabella Tratta	
Indice fk_Tratta_1_idx	**Tipo[56]:**
Capolinea_di_Partenza	IDX

Tabella Tratta

[52] IDX = index, UQ = unique, FT = full text, PR = primary.

[53] IDX = index, UQ = unique, FT = full text, PR = primary.

[54] IDX = index, UQ = unique, FT = full text, PR = primary.

[55] IDX = index, UQ = unique, FT = full text, PR = primary.

[56] IDX = index, UQ = unique, FT = full text, PR = primary.

Indice fk_Tratta_2_idx	**Tipo[57]:**
Capolinea_di_Arrivo	IDX

Indice per la Foreign Key per la tabella Tratta

Tabella Treno_Merci	
Indice PRIMARY	**Tipo[58]:**
Matricola	PR

Tabella Treno_Merci	
Indice fk_Treno_Merci_1_idx	**Tipo[59]:**
Locomotrice	IDX

Indice per la Foreign Key per la tabella Treno_Merci

Tabella Treno_Passeggeri	
Indice PRIMARY	**Tipo[60]:**
Matricola	PR

Tabella Treno_Passeggeri	
Indice fk_Treno_Passeggeri_1_idx	**Tipo[61]:**

[57] IDX = index, UQ = unique, FT = full text, PR = primary.

[58] IDX = index, UQ = unique, FT = full text, PR = primary.

[59] IDX = index, UQ = unique, FT = full text, PR = primary.

[60] IDX = index, UQ = unique, FT = full text, PR = primary.

[61] IDX = index, UQ = unique, FT = full text, PR = primary.

Locomotrice	IDX

Indice per la Foreign Key per la tabella Treno_Passeggeri

Tabella Turno	
Indice PRIMARY	**Tipo[62]:**
Data_Corsa	PR
Lavoratore	PR
Tratta	PR

Tabella Turno	
Indice fk_Turno_2_idx	**Tipo[63]:**
Data_Corsa	IDX
Tratta	IDX

Indice per la Foreign Key per la tabella Turno

Tabella Utenti	
Indice PRIMARY	**Tipo[64]:**
Username	PR

Tabella Vagone_Merci	
Indice PRIMARY	**Tipo[65]:**

[62] IDX = index, UQ = unique, FT = full text, PR = primary.

[63] IDX = index, UQ = unique, FT = full text, PR = primary.

[64] IDX = index, UQ = unique, FT = full text, PR = primary.

[65] IDX = index, UQ = unique, FT = full text, PR = primary.

Id	PR

Tabella Vagone_Merci	
Indice fk_Vagone_Merci_1_idx	**Tipo[66]:**
Treno	IDX

Indice per la Foreign Key per la tabella Vagone_Merci

Tabella Vagone_Passeggeri	
Indice PRIMARY	**Tipo[67]:**
Id	PR

Tabella Vagone_Passeggeri	
Indice fk_Vagone_Passeggeri_1_idx	**Tipo[68]:**
Treno	IDX

Indice per la Foreign Key per la tabella Vagone_Passeggeri

Trigger

CREATE DEFINER = CURRENT_USER TRIGGER
`Trasporto_Ferroviario_Alta_Velocita`.`Corsa_BEFORE_INSERT` BEFORE INSERT ON `Corsa`
FOR EACH ROW

BEGIN

    -- controlla la data degli orari

[66] IDX = index, UQ = unique, FT = full text, PR = primary.

[67] IDX = index, UQ = unique, FT = full text, PR = primary.

[68] IDX = index, UQ = unique, FT = full text, PR = primary.

```
   if ((not (date(NEW.Orario_Effettivo_di_Partenza) = NEW.`Data` and
date(NEW.Orario_Effettivo_di_Arrivo) >= NEW.`Data`)) and (NEW.Orario_Effettivo_di_Partenza
<> null or NEW.Orario_Effettivo_di_Partenza <> null))
                then
                signal sqlstate '45000' set message_text = "La data inserita negli orari non corrisponde
a quella della corsa";
        end if;
END

CREATE DEFINER = CURRENT_USER TRIGGER
`Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Report_BEFORE_INSERT` BEFORE INSERT ON
`Corsa_Report` FOR EACH ROW
BEGIN
        -- controlla che il turno scelto esista

   if not exists (select *
                                from `Turno`
            where Turno.Lavoratore = NEW.Lavoratore and Turno.Data_Corsa = NEW.Data_Corsa
and Turno.Tratta = NEW.Tratta)
                then
                signal sqlstate '45000' set message_text = "Il turno selezionato non esiste";
        end if;
END

CREATE DEFINER = CURRENT_USER TRIGGER
`Trasporto_Ferroviario_Alta_Velocita`.`Ferma_BEFORE_INSERT` BEFORE INSERT ON `Ferma`
FOR EACH ROW
```

```
BEGIN
	-- controlla che gli orari siano validi
   if not ((NEW.Orario_Partenza between '00:00:00' and '24:00:00') or (NEW.Orario_Arrivo between '00:00:00' and '24:00:00'))
		then
		signal sqlstate '45000' set message_text = "Gli orari selezionati non sono validi";
	end if;
END

CREATE DEFINER = CURRENT_USER TRIGGER `Trasporto_Ferroviario_Alta_Velocita`.`Report_sui_turni_BEFORE_INSERT` BEFORE INSERT ON `Report_sui_turni` FOR EACH ROW
BEGIN
	if (select week(Report_sui_turni.`Data`, 1)
		from `Report_sui_turni`
	where Report_sui_turni.Lavoratore = NEW.Lavoratore
	and year(Report_sui_turni.`Data`) = year(NEW.`Data`)) = week(NEW.`Data`, 1)
	then
		signal sqlstate '45000' set message_text = "Non e' possibile aggiungere piu' di un report sui turni per settimana";
	end if;
END

CREATE DEFINER = CURRENT_USER TRIGGER `Trasporto_Ferroviario_Alta_Velocita`.`Tratta_BEFORE_INSERT` BEFORE INSERT ON `Tratta` FOR EACH ROW
```

```
BEGIN
	-- controlla che gli orari siano validi
   if not ((NEW.Orario_Partenza between '00:00:00' and '24:00:00') or (NEW.Orario_Arrivo between '00:00:00' and '24:00:00'))
		then
		signal sqlstate '45000' set message_text = "Gli orari selezionati non sono validi";
	end if;
END

CREATE DEFINER = CURRENT_USER TRIGGER `Trasporto_Ferroviario_Alta_Velocita`.`Turno_BEFORE_INSERT` BEFORE INSERT ON `Turno` FOR EACH ROW
BEGIN
	declare var_Nuovo_Orario_Partenza time;
   declare var_Nuovo_Orario_Arrivo time;
   declare var_new_shift_length int default 0;

   select Tratta.Orario_Partenza, Tratta.Orario_Arrivo
   from `Tratta`
   where NEW.Tratta = Tratta.Codice
   into var_Nuovo_Orario_Partenza, var_Nuovo_Orario_Arrivo;

	-- due turni non si possono sovrapporre
	if exists (select *
	from `Turno` join `Tratta` on Turno.Tratta = Tratta.Codice
```

```
        where Turno.Lavoratore = NEW.Lavoratore
        and Turno.Data_Corsa = NEW.Data_Corsa
        and ((Tratta.Orario_Partenza between var_Nuovo_Orario_Partenza and
var_Nuovo_Orario_Arrivo) or
                (Tratta.Orario_Arrivo between var_Nuovo_Orario_Partenza and
var_Nuovo_Orario_Arrivo)))
    then
                signal sqlstate '45000' set message_text = "Non e' possibile aggiungere un turno che si
sovrapponga ad un altro";
        end if;

        -- al lavoratore non possono essere assegnati piu' di cinque turni per settimana
        if (select count(*)
                from `Turno`
                where Turno.Lavoratore = NEW.Lavoratore
    and week(Turno.Data_Corsa, 1) = week(NEW.Data_Corsa, 1)
    and year(Turno.Data_Corsa) = year(NEW.Data_Corsa)) >= 5
    then
                signal sqlstate '45000' set message_text = "Non e' possibile assegnare ad un lavoratore
piu' di cinque turni in una settimana";
        end if;

        -- un lavoratore non puo' effettuare più di quattro ore di lavoro a settimana
```

```
    select if(timestampdiff(second, Orario_Partenza, Orario_Arrivo) >0, timestampdiff(second, Orario_Partenza, Orario_Arrivo), timestampdiff(second, Orario_Partenza, Orario_Arrivo) + 24*3600)
        from `Tratta`
        where `Tratta`.`Codice` = NEW.Tratta
    into var_new_shift_length;

        if(select sum(if(timestampdiff(second, Orario_Partenza, Orario_Arrivo) >0, timestampdiff(second, Orario_Partenza, Orario_Arrivo), timestampdiff(second, Orario_Partenza, Orario_Arrivo) + 24*3600))
                from `Turno` join `Tratta` on `Turno`.Tratta = `Tratta`.`Codice`
                where Turno.Lavoratore = NEW.Lavoratore
                        and Turno.Copertura = 'coperto'
                        and week(Turno.Data_Corsa, 1) = week(NEW.Data_Corsa, 1)
                        and year(Turno.Data_Corsa) = year(NEW.Data_Corsa)) >= 4*3600 - var_new_shift_length then
                signal sqlstate '45000' set message_text = "Un lavoratore non puo' effettuare piu' di quattro ore di lavoro a settimana";
        end if;
END
```

Eventi

Non sono previsti eventi.

Viste

Non è stata necessaria l’introduzione di viste.

Stored Procedures e transazioni

CREATE PROCEDURE `acquista_biglietto` (IN var_Data_Corsa date, IN var_Tratta int, IN var_Classe int, IN var_Codice_Fiscale varchar(16), IN var_Nome varchar(45), IN var_Cognome varchar(45), IN var_Data_di_nascita date, in var_Numero_CC varchar(16))

BEGIN

declare var_treno varchar(4);

declare var_biglietto_id int;

set transaction isolation level repeatable read;

start transaction;

-- troviamo il treno che percorre la corsa specificata

select `Treno_Passeggeri`

from `Corsa_Treno_Passeggeri`

where Corsa_Treno_Passeggeri.Data_Corsa = var_Data_Corsa

and Corsa_Treno_Passeggeri.Tratta = var_Tratta

into var_treno;

select @var_Posto := Posto.`Numero`, @var_Vagone := Posto.`Vagone_Passeggeri`

from `Posto` join Vagone_Passeggeri on Posto.Vagone_Passeggeri = Vagone_Passeggeri.Id join Treno_Passeggeri on Vagone_Passeggeri.Treno = Treno_Passeggeri.Matricola

where Treno_Passeggeri.Matricola = var_treno and `Vagone_Passeggeri`.Classe = var_Classe and (Posto.`Numero`, Posto.`Vagone_Passeggeri`) not in (select `Numero_Posto`, `Vagone_Passeggeri` from `Biglietto`

where Biglietto.Data_Corsa = var_Data_Corsa

and Biglietto.Tratta = var_Tratta)

```
    limit 1;

    insert into `Biglietto` (`Numero_Posto`, `Vagone_Passeggeri`, `Data_Corsa`, `Tratta`, `Codice_Fiscale`, `Nome`, `Cognome`, `Data_di_Nascita`, `Numero_CC`) values (@var_Posto, @var_Vagone, var_Data_Corsa, var_Tratta, var_Codice_Fiscale, var_Nome, var_Cognome, var_Data_di_nascita, var_Numero_CC);

    set var_biglietto_id = last_insert_id();
    select var_biglietto_id, @var_Posto, @var_Vagone;

    commit;
END

CREATE PROCEDURE `aggiungi_azienda` (IN var_Partita_IVA varchar(11), IN var_Recapito varchar(45), IN var_Ragione_Sociale varchar(45))
BEGIN
        set transaction isolation level repeatable read;
    start transaction;
        insert into `Azienda` (`Partita_IVA`, `Recapito`, `Ragione_Sociale`) values (var_Partita_IVA, var_Recapito, var_Ragione_Sociale);
        commit;
END

CREATE PROCEDURE `aggiungi_corsa_treno_merci` (IN var_Data date, IN var_Tratta int, IN var_Treno_Merci varchar(4), IN var_Conducente varchar(16))
BEGIN
```

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


        set transaction isolation level repeatable read;
        start transaction;
    insert into `Corsa` (`Data`, `Tratta`)  values (var_Data, var_Tratta);
    insert into `Corsa_Treno_Merci` (`Data_Corsa`, `Tratta`, `Treno_Merci`)  values (var_Data, var_Tratta, var_Treno_Merci);


        if((select `Ruolo` from `Lavoratore` where `Codice_Fiscale` = var_Conducente)<>'conducente') then
                signal sqlstate '45001' set message_text = "Il lavoratore selezionato non è un conducente";
        end if;


    insert into `Turno` (`Data_Corsa`, `Lavoratore`, `Tratta`) values (var_Data, var_Conducente, var_Tratta);
        commit;
END
```

```
CREATE PROCEDURE `aggiungi_corsa_treno_passeggeri` (IN var_Data date, IN var_Tratta int, IN var_Treno_Passeggeri varchar(4), IN var_Conducente varchar(16), IN var_Capotreno varchar(16))
BEGIN
	declare exit handler for sqlexception
  begin
    rollback;  -- rollback any changes made in the transaction
    resignal;  -- raise again the sql exception to the caller
  end;

	set transaction isolation level repeatable read;
	start transaction;
  insert into `Corsa` (`Data`, `Tratta`)  values (var_Data, var_Tratta);
  insert into `Corsa_Treno_Passeggeri` (`Data_Corsa`, `Tratta`, `Treno_Passeggeri`) values (var_Data, var_Tratta, var_Treno_Passeggeri);

	if((select `Ruolo` from `Lavoratore` where `Codice_Fiscale` = var_Conducente)<>'conducente') then
		signal sqlstate '45001' set message_text = "Il lavoratore selezionato non e' un conducente";
	end if;

	if((select `Ruolo` from `Lavoratore` where `Codice_Fiscale` = var_Capotreno)<>'capotreno') then
		signal sqlstate '45001' set message_text = "Il lavoratore selezionato non e' un capotreno";
```

```
    end if;

  insert into `Turno` (`Data_Corsa`, `Lavoratore`, `Tratta`) values (var_Data, var_Conducente, var_Tratta);

  insert into `Turno` (`Data_Corsa`, `Lavoratore`, `Tratta`) values (var_Data, var_Capotreno, var_Tratta);

    commit;

END


CREATE PROCEDURE `aggiungi_lavoratore` (IN var_Codice_Fiscale varchar(11), IN var_Nome varchar(45), IN var_Cognome varchar(45), IN var_Data_di_Nascita date, IN var_Citta_di_Nascita varchar(45), IN var_Ruolo varchar(45), IN var_Provincia_di_nascita varchar(2))

BEGIN

    set transaction isolation level repeatable read;

    start transaction;

    insert into `Lavoratore` (`Codice_Fiscale`, `Nome`, `Cognome`, `Data_di_Nascita`, `Citta_di_Nascita`, `Ruolo`, `Provincia_di_Nascita`)  values (var_Codice_Fiscale, var_Nome, var_Cognome, var_Data_di_Nascita, var_Citta_di_Nascita, var_Ruolo, var_Provincia_di_nascita);

    commit;

END


CREATE PROCEDURE `aggiungi_report_manutenzione_locomotrice` (IN var_Locomotrice int, IN var_Testo varchar(200))

BEGIN

    set transaction isolation level repeatable read;

    start transaction;
```

```
        insert into `Report_Manutenzione_Locomotrice` (`Locomotrice`, `Testo`, `Timestamp`) values (var_Locomotrice, var_Testo, current_timestamp());

        commit;

END


CREATE PROCEDURE `aggiungi_report_manutenzione_vagone_merci` (IN var_Vagone_Merci int, IN var_Testo varchar(200))

BEGIN

        set transaction isolation level repeatable read;

        start transaction;

        insert into `Report_Manutenzione_Vagone_Merci` (`Vagone_Merci`, `Testo`, `Timestamp`) values (var_Vagone_Merci, var_Testo, current_timestamp());

        commit;

END


CREATE PROCEDURE `aggiungi_report_manutenzione_vagone_passeggeri` (IN var_Vagone_Passeggeri int, IN var_Testo varchar(200))

BEGIN

        set transaction isolation level repeatable read;

        start transaction;

        insert into `Report_Manutenzione_Vagone_Passeggeri` (`Vagone_Passeggeri`, `Testo`, `Timestamp`)  values (var_Vagone_Passeggeri, var_Testo, current_timestamp());

        commit;

END
```

```
CREATE PROCEDURE `aggiungi_spedizione` (in var_Azienda_Mittente varchar(11), in var_Azienda_Destinataria varchar(11), in var_Merce varchar(45), in var_Massa_Complessiva int, in var_Vagone_Merci int, in var_Data_Corsa date, in var_Tratta int)
BEGIN
	declare var_treno_corsa varchar(4);
  declare var_treno_vagone varchar(4);

	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 Treno_Merci
  from `Corsa_Treno_Merci`
  where Corsa_Treno_Merci.Data_Corsa = var_Data_Corsa & Corsa_Treno_Merci.Tratta = var_Tratta
  into var_treno_corsa;

  select Treno
  from `Vagone_Merci`
  where Vagone_Merci.Id = var_Vagone_Merci
```

```
    into var_treno_vagone;


    if(var_treno_corsa<>var_treno_vagone) then
            signal sqlstate '45001' set message_text = "Il vagone selezionato non e' parte del treno che percorre la corsa richiesta";
        end if;


    insert into `Spedizione` (`Azienda_Mittente`, `Azienda_Destinataria`, `Merce`, `Massa_Complessiva`, `Vagone_Merci`, `Data_Corsa`, `Tratta`) values (var_Azienda_Mittente, var_Azienda_Destinataria, var_Merce, var_Massa_Complessiva, var_Vagone_Merci, var_Data_Corsa, var_Tratta);
        commit;
END


CREATE PROCEDURE `aggiungi_treno_merci` (IN var_Matricola varchar(4), IN var_Numero_Vagoni int, IN var_Data_di_Acquisto date, IN var_Marca_Locomotrice varchar(45), IN var_Modello_Locomotrice varchar(45), IN var_Marca_Vagone varchar(45), IN var_Modello_Vagone varchar(45), IN var_Portata int)
BEGIN
    declare count int default 0;


        set transaction isolation level repeatable read;
        start transaction;
    insert into `Locomotrice` (`Marca`, `Modello`) values (var_Marca_Locomotrice, var_Modello_Locomotrice);
```

```
        insert into `Treno_Merci` (`Matricola`, `Numero_Vagoni`, `Data_di_Acquisto`, `Locomotrice`) values (var_Matricola, var_Numero_Vagoni, var_Data_di_Acquisto, last_insert_id());


        -- aggiungo vagoni
    vagoni_loop: loop
                if count >= var_Numero_Vagoni then
                        leave vagoni_loop;
                end if;


      set count = count +1;


      insert into `Vagone_Merci` (`Marca`, `Modello`, `Portata`, `Treno`) values (var_Marca_Vagone, var_Modello_Vagone, var_Portata, var_Matricola);
        end loop;


    commit;
END


CREATE PROCEDURE `aggiungi_treno_passeggeri` (IN var_Matricola varchar(4), IN var_Numero_Vagoni int, IN var_Data_di_Acquisto date, IN var_Numero_Carrozze_I_Classe int, IN var_Numero_Carrozze_II_Classe int, IN var_Marca_Locomotrice varchar(45), IN var_Modello_Locomotrice varchar(45), IN var_Marca_Vagone_I_Classe varchar(45), IN var_Modello_Vagone_I_Classe varchar(45), IN var_Numero_Massimo_Passeggeri_Vagone_I_Classe int, IN var_Marca_Vagone_II_Classe varchar(45), IN var_Modello_Vagone_II_Classe varchar(45), IN var_Numero_Massimo_Passeggeri_Vagone_II_Classe int)
```

```
BEGIN
   declare count int default 0;
   declare count_posti int default 0;

        set transaction isolation level repeatable read;
        start transaction;
   insert into `Locomotrice` (`Marca`, `Modello`) values (var_Marca_Locomotrice, var_Modello_Locomotrice);

        insert into `Treno_Passeggeri` (`Matricola`, `Numero_Vagoni`, `Data_di_Acquisto`, `Locomotrice`, `Numero_Carrozze_I_Classe`, `Numero_Carrozze_II_Classe`) values (var_Matricola, var_Numero_Vagoni, var_Data_di_Acquisto, last_insert_id(), var_Numero_Carrozze_I_Classe, var_Numero_Carrozze_II_Classe);

        -- aggiungo vagoni prima classe
   prima_classe_loop: loop
                if count >= var_Numero_Carrozze_I_Classe then
                        set count = 0;
                        leave prima_classe_loop;
                end if;

      set count = count +1;

      insert into `Vagone_Passeggeri` (`Marca`, `Modello`, `Classe`, `Numero_Massimo_di_Passeggeri`, `Treno`) values (var_Marca_Vagone_I_Classe,
```

```
var_Modello_Vagone_I_Classe, '1', var_Numero_Massimo_Passeggeri_Vagone_I_Classe, var_Matricola);

            -- aggiungo posti prima classe
    posti_prima_classe_loop: loop
                if count_posti >= var_Numero_Massimo_Passeggeri_Vagone_I_Classe then
                    set count_posti = 0;
                    leave posti_prima_classe_loop;
                end if;

                set count_posti = count_posti +1;

                insert into Posto (Numero, Vagone_Passeggeri) values (count_posti, last_insert_id());

    end loop;
      end loop;

      -- aggiungo vagoni seconda classe
  seconda_classe_loop: loop
            if count >= var_Numero_Carrozze_II_Classe then
                leave seconda_classe_loop;
            end if;

    set count = count +1;
```

```
insert into `Vagone_Passeggeri` (`Marca`, `Modello`, `Classe`, `Numero_Massimo_di_Passeggeri`, `Treno`) values (var_Marca_Vagone_II_Classe, var_Modello_Vagone_II_Classe, '2', var_Numero_Massimo_Passeggeri_Vagone_II_Classe, var_Matricola);


        -- aggiungo posti seconda classe
    posti_seconda_classe_loop: loop
            if count_posti >= var_Numero_Massimo_Passeggeri_Vagone_II_Classe then
                set count_posti = 0;
                leave posti_seconda_classe_loop;
            end if;

            set count_posti = count_posti +1;

            insert into Posto (Numero, Vagone_Passeggeri) values (count_posti, last_insert_id());

    end loop;
      end loop;

  commit;
END

CREATE PROCEDURE `aggiungi_turno_a_report_sui_turni` (in var_Data_Corsa date, in var_Tratta int, in var_Data_Report date)
```

```
BEGIN
        set transaction isolation level repeatable read;
        start transaction;
        insert into `Corsa_Report` (`Lavoratore`, `Data_Report`, `Data_Corsa`, `Tratta` ) values (var_Lavoratore, var_Data_Report, var_Data_Corsa, var_Tratta);
  commit;
END
CREATE PROCEDURE `convalida_biglietto` (in var_Codice_di_Prenotazione int)
BEGIN

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

        set transaction isolation level repeatable read;
        start transaction;

   if exists (select Biglietto.Codice_di_Prenotazione
                                from `Biglietto`
            where Biglietto.Codice_di_Prenotazione = var_Codice_di_Prenotazione)
        then update `Biglietto`
                    set  `Status` = 'convalidato'
      where Biglietto.Codice_di_prenotazione = var_Codice_di_prenotazione;
```

```
    else
            signal sqlstate '45002' set message_text = "Il codice immesso non e' stato trovato";
    end if;
    commit;
END

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

CREATE PROCEDURE `genera_report_sui_turni_di_lavoro` (IN var_Lavoratore varchar(16), IN var_Testo_Rapporto varchar(500))
BEGIN
    declare var_entry_number int default 0;

    set transaction isolation level repeatable read;
    start transaction;
  insert into `Report_sui_turni` (`Lavoratore`, `Data`, `Testo_Rapporto`)  values (var_Lavoratore, current_date(), var_Testo_Rapporto);
commit;
END
```

```
CREATE PROCEDURE `inserisci_orari_corsa_effettivi` (in var_Data date, in var_Tratta int, in var_Orario_Effettivo_di_Partenza datetime, in var_Orario_Effettivo_di_Arrivo datetime)
BEGIN
	set transaction isolation level repeatable read;
	start transaction;

  update `Corsa`
  set `Orario_Effettivo_di_Partenza` = var_Orario_Effettivo_di_Partenza and `Orario_Effettivo_di_Arrivo` = var_Orario_Effettivo_di_Arrivo
  where Corsa.`Data` = var_Data and Corsa.`Tratta` = var_Tratta;

  commit;
END

CREATE PROCEDURE `login` (in var_username varchar(45), in var_pass varchar(45), out var_role INT)
BEGIN
	declare var_user_role ENUM('Gestore_del_servizio', 'Lavoratore', 'Addetto_alla_manutenzione', 'Controllore', 'Acquirente');

	select `Ruolo` from `Utenti`
		where `Username` = var_username
    and `Password` = md5(var_pass)
    into var_user_role;
```

```
    -- See the corresponding enum in the client
            if var_user_role = 'Gestore_del_servizio' then
                set var_role = 1;
            elseif var_user_role = 'Lavoratore' then
                set var_role = 2;
            elseif var_user_role = 'Addetto_alla_manutenzione' then
                set var_role = 3;
            elseif var_user_role = 'Controllore' then
                set var_role = 4;
            else
                set var_role = 5;
            end if;
END

CREATE PROCEDURE `sostituisci_lavoratore` (IN var_Data_Turno date, IN var_Lavoratore_Malattia varchar(16) , IN var_Lavoratore_Sostituto varchar(16), IN var_Tratta int)
BEGIN
      declare exit handler for sqlexception
  begin
    rollback;  -- rollback any changes made in the transaction
    resignal;  -- raise again the sql exception to the caller
  end;

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

```
if (var_Data_Turno <= current_date()) then
        signal sqlstate '45001' set message_text = "Non è possibile modificare un turno passato";
    end if;

if not exists (select *
                    from `Turno`
        where `Copertura` = 'coperto' and `Turno`.`Lavoratore` = var_Lavoratore_Malattia  and `Turno`.`Data_Corsa` = var_Data_Turno and `Turno`.`Tratta` = var_Tratta)
        then
                    signal sqlstate '45001' set message_text = "Non è stato possibile trovare un turno valido che corrispondesse ai criteri di ricerca";
    end if;

if not exists (select  Malato.Ruolo = Sostituto.Ruolo
                    from `Lavoratore` as `Malato`, `Lavoratore` as `Sostituto`
                    where Malato.Codice_Fiscale =  var_Lavoratore_Malattia and Sostituto.Codice_Fiscale = var_Lavoratore_Sostituto) then
        signal sqlstate '45001' set message_text = "I lavoratori selezionati devono avere lo stesso ruolo";
    end if;

    update `Turno`
    set  `Copertura` = 'malattia'
```

```
    WHERE `Turno`.`Lavoratore` = var_Lavoratore_Malattia  and `Turno`.`Data_Corsa` = var_Data_Turno and `Turno`.`Tratta` = var_Tratta;


  INSERT INTO `Turno` (`Data_Corsa`, `Lavoratore`, `Tratta`) values (var_Data_Turno, var_Lavoratore_Sostituto, var_Tratta);
      commit;
END

CREATE PROCEDURE `verifica_biglietto` (IN var_Codice_di_prenotazione int)
BEGIN

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

      select `Codice_Fiscale`, `Nome`, `Cognome`, `Data_di_Nascita`
  from `Biglietto`
      where `Codice_di_prenotazione` = var_Codice_di_prenotazione;

  commit;
END

CREATE PROCEDURE `visualizza_storico_treno_merci` (in var_Treno varchar(4))
BEGIN

      set transaction isolation level read committed;
```

```
    start transaction read only;

  -- storico manutenzione locomotrice

    select `Report_Manutenzione_Locomotrice`.`Timestamp`,
`Report_Manutenzione_Locomotrice`.`Testo`
  from `Report_Manutenzione_Locomotrice` join `Locomotrice` on
`Report_Manutenzione_Locomotrice`.`Locomotrice` = `Locomotrice`.`Id` join `Treno_Merci` on
`Locomotrice`.`Id` = `Treno_Merci`.`Locomotrice`
    where `Treno_Merci`.`Matricola` = var_Treno
  order by `Report_Manutenzione_Locomotrice`.`Timestamp`;

  -- storico manutenzione vagoni

    select `Vagone_Merci`.`Id` as 'vagone',
`Report_Manutenzione_Vagone_Merci`.`Timestamp`,
`Report_Manutenzione_Vagone_Merci`.`Testo`
  from `Report_Manutenzione_Vagone_Merci` join `Vagone_Merci` on
`Report_Manutenzione_Vagone_Merci`.`Vagone_Merci` = `Vagone_Merci`.`Id` join `Treno_Merci`
on `Vagone_Merci`.`Treno` = `Treno_Merci`.`Matricola`
    where `Treno_Merci`.`Matricola` = var_Treno
  order by `Vagone_Merci`.`Id`, `Report_Manutenzione_Vagone_Merci`.`Timestamp`;

    commit;
END
```

```
CREATE PROCEDURE `visualizza_storico_treno_passeggeri` (in var_Treno varchar(4))
BEGIN

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

  -- storico manutenzione locomotrice

    select `Report_Manutenzione_Locomotrice`.`Timestamp`, `Report_Manutenzione_Locomotrice`.`Testo`
  from `Report_Manutenzione_Locomotrice` join `Locomotrice` on `Report_Manutenzione_Locomotrice`.`Locomotrice` = `Locomotrice`.`Id` join `Treno_Passeggeri` on `Locomotrice`.`Id` = `Treno_Passeggeri`.`Locomotrice`
    where `Treno_Passeggeri`.`Matricola` = var_Treno
  order by `Report_Manutenzione_Locomotrice`.`Timestamp`;

  -- storico manutenzione vagoni

    select `Vagone_Passeggeri`.`Id` as 'vagone', `Report_Manutenzione_Vagone_Passeggeri`.`Timestamp`, `Report_Manutenzione_Vagone_Passeggeri`.`Testo`
  from `Report_Manutenzione_Vagone_Passeggeri` join `Vagone_Passeggeri` on `Report_Manutenzione_Vagone_Passeggeri`.`Vagone_Passeggeri` = `Vagone_Passeggeri`.`Id` join `Treno_Passeggeri` on `Vagone_Passeggeri`.`Treno` = `Treno_Passeggeri`.`Matricola`
    where `Treno_Passeggeri`.`Matricola` = var_Treno
  order by `Vagone_Passeggeri`.`Id`, `Report_Manutenzione_Vagone_Passeggeri`.`Timestamp`;
```

```
        commit;
END
```

Appendice: Implementazione

Codice SQL per instanziare il database

```
-- MySQL Workbench Forward Engineering

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

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

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

-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Azienda`
```

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

DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Azienda` ;


CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Azienda` (

  `Partita_IVA` VARCHAR(11) NOT NULL,

  `Recapito` VARCHAR(45) NOT NULL,

  `Ragione_Sociale` VARCHAR(45) NOT NULL,

  PRIMARY KEY (`Partita_IVA`))

ENGINE = InnoDB;


CREATE UNIQUE INDEX `Recapito_UNIQUE` ON `Trasporto_Ferroviario_Alta_Velocita`.`Azienda` (`Recapito` ASC) VISIBLE;


CREATE UNIQUE INDEX `Ragione_Sociale_UNIQUE` ON `Trasporto_Ferroviario_Alta_Velocita`.`Azienda` (`Ragione_Sociale` ASC) VISIBLE;



-- -----------------------------------------------------

-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Locomotrice`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Locomotrice` ;


CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Locomotrice` (

  `Id` INT UNSIGNED NOT NULL AUTO_INCREMENT,

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

```
  `Modello` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Id`))
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Passeggeri`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Passeggeri` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Passeggeri` (
  `Matricola` VARCHAR(4) NOT NULL,
  `Numero_Vagoni` INT UNSIGNED NOT NULL,
  `Data_di_Acquisto` DATE NOT NULL,
  `Locomotrice` INT UNSIGNED NOT NULL,
  `Numero_Carrozze_I_Classe` INT UNSIGNED NOT NULL,
  `Numero_Carrozze_II_Classe` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`Matricola`),
  CONSTRAINT `fk_Treno_Passeggeri_1`
    FOREIGN KEY (`Locomotrice`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Locomotrice` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
CREATE INDEX `fk_Treno_Passeggeri_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Passeggeri` (`Locomotrice` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri` ;


CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri` (
  `Id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Marca` VARCHAR(45) NOT NULL,
  `Modello` VARCHAR(45) NOT NULL,
  `Classe` ENUM('1', '2') NOT NULL,
  `Numero_Massimo_di_Passeggeri` INT UNSIGNED NOT NULL,
  `Treno` VARCHAR(4) NOT NULL,
  PRIMARY KEY (`Id`),
  CONSTRAINT `fk_Vagone_Passeggeri_1`
    FOREIGN KEY (`Treno`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Passeggeri` (`Matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
CREATE INDEX `fk_Vagone_Passeggeri_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri` (`Treno` ASC) VISIBLE;


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

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (
  `Numero` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Vagone_Passeggeri` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`Numero`, `Vagone_Passeggeri`),
  CONSTRAINT `fk_Posto_1`
    FOREIGN KEY (`Vagone_Passeggeri`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Posto_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Vagone_Passeggeri` ASC) VISIBLE;


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

```
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Stazione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Stazione` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Stazione` (
  `Codice` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Nome` VARCHAR(45) NOT NULL,
  `Citta` VARCHAR(45) NOT NULL,
  `Provincia` VARCHAR(2) NOT NULL,
  PRIMARY KEY (`Codice`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Tratta`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Tratta` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Tratta` (
  `Codice` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Capolinea_di_Partenza` INT UNSIGNED NOT NULL,
  `Capolinea_di_Arrivo` INT UNSIGNED NOT NULL,
  `Orario_Partenza` TIME NOT NULL,
  `Orario_Arrivo` TIME NOT NULL,
```

```
  PRIMARY KEY (`Codice`),
  CONSTRAINT `fk_Tratta_1`
    FOREIGN KEY (`Capolinea_di_Partenza`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Stazione` (`Codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Tratta_2`
    FOREIGN KEY (`Capolinea_di_Arrivo`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Stazione` (`Codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Tratta_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Tratta` (`Capolinea_di_Partenza` ASC) VISIBLE;

CREATE INDEX `fk_Tratta_2_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Tratta` (`Capolinea_di_Arrivo` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Corsa`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` ;
```

```
CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` (
  `Data` DATE NOT NULL,
  `Tratta` INT UNSIGNED NOT NULL,
  `Orario_Effettivo_di_Partenza` DATETIME NULL,
  `Orario_Effettivo_di_Arrivo` DATETIME NULL,
  PRIMARY KEY (`Data`, `Tratta`),
  CONSTRAINT `fk_Corsa_1`
    FOREIGN KEY (`Tratta`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Tratta` (`Codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Corsa_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` (`Tratta` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Passeggeri`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Passeggeri` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Passeggeri` (
  `Data_Corsa` DATE NOT NULL,
```

```
  `Tratta` INT UNSIGNED NOT NULL,
  `Treno_Passeggeri` VARCHAR(4) NOT NULL,
  PRIMARY KEY (`Data_Corsa`, `Tratta`),
  CONSTRAINT `fk_Corsa_Treno_Passeggeri_1`
    FOREIGN KEY (`Data_Corsa` , `Tratta`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` (`Data` , `Tratta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Corsa_Treno_Passeggeri_2`
    FOREIGN KEY (`Treno_Passeggeri`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Passeggeri` (`Matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Corsa_Treno_Passeggeri_2_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Passeggeri` (`Treno_Passeggeri` ASC) INVISIBLE;


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

```
CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Biglietto` (
  `Codice_di_prenotazione` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Numero_Posto` INT UNSIGNED NOT NULL,
  `Vagone_Passeggeri` INT UNSIGNED NOT NULL,
  `Data_Corsa` DATE NOT NULL,
  `Tratta` INT UNSIGNED NOT NULL,
  `Status` ENUM('non-convalidato', 'convalidato') NULL DEFAULT 'non-convalidato',
  `Codice_Fiscale` VARCHAR(16) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `Data_di_nascita` DATE NOT NULL,
  `Numero_CC` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`Codice_di_prenotazione`),
  CONSTRAINT `fk_Biglietto_1`
    FOREIGN KEY (`Numero_Posto` , `Vagone_Passeggeri`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero` , `Vagone_Passeggeri`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Biglietto_2`
    FOREIGN KEY (`Data_Corsa` , `Tratta`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Passeggeri` (`Data_Corsa` ,
`Tratta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
CREATE INDEX `fk_Biglietto_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Biglietto` (`Numero_Posto` ASC, `Vagone_Passeggeri` ASC) VISIBLE;


CREATE INDEX `fk_Biglietto_2_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Biglietto` (`Data_Corsa` ASC, `Tratta` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Lavoratore`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Lavoratore` ;


CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Lavoratore` (
  `Codice_Fiscale` VARCHAR(16) NOT NULL,
  `Nome` VARCHAR(45) NOT NULL,
  `Cognome` VARCHAR(45) NOT NULL,
  `Data_di_Nascita` DATE NOT NULL,
  `Citta_di_Nascita` VARCHAR(45) NOT NULL,
  `Ruolo` ENUM('capotreno', 'conducente') NOT NULL,
  `Provincia_di_Nascita` VARCHAR(2) NOT NULL,
  PRIMARY KEY (`Codice_Fiscale`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Report_sui_turni`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Report_sui_turni` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Report_sui_turni` (
  `Lavoratore` VARCHAR(16) NOT NULL,
  `Data` DATE NOT NULL,
  `Testo_Rapporto` VARCHAR(500) NOT NULL,
  PRIMARY KEY (`Lavoratore`, `Data`),
  CONSTRAINT `fk_Report_sui_turni_1`
    FOREIGN KEY (`Lavoratore`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Lavoratore` (`Codice_Fiscale`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Report`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Report` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Report` (
```

```
  `Numero` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Lavoratore` VARCHAR(16) NOT NULL,
  `Data_Report` DATE NOT NULL,
  `Data_Corsa` DATE NOT NULL,
  `Tratta` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`Numero`, `Lavoratore`, `Data_Report`),
  CONSTRAINT `fk_Corsa_Report_1`
    FOREIGN KEY (`Lavoratore` , `Data_Report`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Report_sui_turni` (`Lavoratore` , `Data`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Corsa_Report_2`
    FOREIGN KEY (`Data_Corsa` , `Tratta`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` (`Data` , `Tratta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Corsa_Report_2_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Report` (`Data_Corsa` ASC, `Tratta` ASC) VISIBLE;

-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Merci`
```

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

DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Merci` ;


CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Merci` (
  `Matricola` VARCHAR(4) NOT NULL,
  `Numero_Vagoni` INT UNSIGNED NOT NULL,
  `Data_di_Acquisto` DATE NOT NULL,
  `Locomotrice` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`Matricola`),
  CONSTRAINT `fk_Treno_Merci_1`
    FOREIGN KEY (`Locomotrice`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Locomotrice` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `fk_Treno_Merci_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Merci` (`Locomotrice` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Merci`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Merci` ;
```

```
CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Merci` (
  `Data_Corsa` DATE NOT NULL,
  `Tratta` INT UNSIGNED NOT NULL,
  `Treno_Merci` VARCHAR(4) NOT NULL,
  PRIMARY KEY (`Data_Corsa`, `Tratta`),
  CONSTRAINT `fk_Corsa_Treno_Merci_1`
    FOREIGN KEY (`Data_Corsa` , `Tratta`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` (`Data` , `Tratta`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Corsa_Treno_Merci_2`
    FOREIGN KEY (`Treno_Merci`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Merci` (`Matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Corsa_Treno_Merci_2_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Merci` (`Treno_Merci` ASC) INVISIBLE;

CREATE UNIQUE INDEX `fk_Corsa_Treno_Merci_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Merci` (`Data_Corsa` ASC, `Tratta` ASC) VISIBLE;
```

```
-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Ferma`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Ferma` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Ferma` (
  `Orario_Partenza` TIME NOT NULL,
  `Orario_Arrivo` TIME NOT NULL,
  `Tratta` INT UNSIGNED NOT NULL,
  `Stazione` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`Tratta`, `Stazione`),
  CONSTRAINT `fk_Ferma_1`
    FOREIGN KEY (`Tratta`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Tratta` (`Codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Ferma_2`
    FOREIGN KEY (`Stazione`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Stazione` (`Codice`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Ferma_2_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Ferma` (`Stazione`
ASC) VISIBLE;
```

```
-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Locomotrice`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Locomotrice` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Locomotrice` (
  `Locomotrice` INT UNSIGNED NOT NULL,
  `Testo` VARCHAR(200) NOT NULL,
  `Timestamp` TIMESTAMP NOT NULL,
  PRIMARY KEY (`Locomotrice`, `Timestamp`),
  CONSTRAINT `fk_Report_Manutenzione_Locomotrice_1`
    FOREIGN KEY (`Locomotrice`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Locomotrice` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Merci`
-- -----------------------------------------------------
```

```
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Merci` ;


CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Merci` (
  `Id` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Marca` VARCHAR(45) NOT NULL,
  `Modello` VARCHAR(45) NOT NULL,
  `Portata` INT UNSIGNED NOT NULL,
  `Treno` VARCHAR(4) NOT NULL,
  PRIMARY KEY (`Id`),
  CONSTRAINT `fk_Vagone_Merci_1`
    FOREIGN KEY (`Treno`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Merci` (`Matricola`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


CREATE INDEX `fk_Vagone_Merci_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Merci` (`Treno` ASC) VISIBLE;



-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Merci`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Merci` ;
```

```
CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Merci` (
  `Vagone_Merci` INT UNSIGNED NOT NULL,
  `Testo` VARCHAR(200) NOT NULL,
  `Timestamp` TIMESTAMP NOT NULL,
  PRIMARY KEY (`Vagone_Merci`, `Timestamp`),
  CONSTRAINT `fk_Report_Manutenzione_Vagone_Merci_1`
    FOREIGN KEY (`Vagone_Merci`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Merci` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Passeggeri`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Passeggeri` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Passeggeri` (
  `Vagone_Passeggeri` INT UNSIGNED NOT NULL,
  `Testo` VARCHAR(200) NOT NULL,
```

```
  `Timestamp` TIMESTAMP NOT NULL,
  PRIMARY KEY (`Vagone_Passeggeri`, `Timestamp`),
  CONSTRAINT `fk_Report_Manutenzione_Vagone_Passeggeri_1`
    FOREIGN KEY (`Vagone_Passeggeri`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Spedizione`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Spedizione` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Spedizione` (
  `Azienda_Mittente` VARCHAR(11) NOT NULL,
  `Azienda_Destinataria` VARCHAR(11) NOT NULL,
  `Merce` VARCHAR(45) NOT NULL,
  `Massa_Complessiva` INT UNSIGNED NOT NULL,
  `Vagone_Merci` INT UNSIGNED NOT NULL,
  `Data_Corsa` DATE NOT NULL,
  `Tratta` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`Tratta`, `Data_Corsa`, `Vagone_Merci`),
```

```
  CONSTRAINT `fk_Spedizione_1`
    FOREIGN KEY (`Azienda_Mittente`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Azienda` (`Partita_IVA`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Spedizione_2`
    FOREIGN KEY (`Azienda_Destinataria`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Azienda` (`Partita_IVA`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Spedizione_3`
    FOREIGN KEY (`Vagone_Merci`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Merci` (`Id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Spedizione_4`
    FOREIGN KEY (`Tratta` , `Data_Corsa`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` (`Tratta` , `Data`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Spedizione_1_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Spedizione`
(`Azienda_Mittente` ASC) VISIBLE;
```

```
CREATE INDEX `fk_Spedizione_2_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Spedizione` (`Azienda_Destinataria` ASC) INVISIBLE;


CREATE INDEX `fk_Spedizione_3_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Spedizione` (`Vagone_Merci` ASC) INVISIBLE;



-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Turno`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Turno` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Turno` (
  `Lavoratore` VARCHAR(4) NOT NULL,
  `Data_Corsa` DATE NOT NULL,
  `Tratta` INT UNSIGNED NOT NULL,
  `Copertura` ENUM('coperto', 'malattia') NULL DEFAULT 'coperto',
  PRIMARY KEY (`Lavoratore`, `Data_Corsa`, `Tratta`),
  CONSTRAINT `fk_Turno_1`
    FOREIGN KEY (`Lavoratore`)
    REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Lavoratore` (`Codice_Fiscale`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Turno_2`
    FOREIGN KEY (`Data_Corsa` , `Tratta`)
```

```
  REFERENCES `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` (`Data` , `Tratta`)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION)
ENGINE = InnoDB;

CREATE INDEX `fk_Turno_2_idx` ON `Trasporto_Ferroviario_Alta_Velocita`.`Turno` (`Data_Corsa` ASC, `Tratta` ASC) VISIBLE;


-- -----------------------------------------------------
-- Table `Trasporto_Ferroviario_Alta_Velocita`.`Utenti`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Utenti` ;

CREATE TABLE IF NOT EXISTS `Trasporto_Ferroviario_Alta_Velocita`.`Utenti` (
  `Username` VARCHAR(45) NOT NULL,
  `Password` CHAR(32) NOT NULL,
  `Ruolo` ENUM('Gestore_del_servizio', 'Lavoratore', 'Addetto_alla_manutenzione', 'Controllore', 'Acquirente') NOT NULL,
  PRIMARY KEY (`Username`))
ENGINE = InnoDB;

SET SQL_MODE = '';
DROP USER IF EXISTS Gestore_del_servizio;
```

```
SET
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO
_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Gestore_del_servizio' IDENTIFIED BY 'Gestore_del_servizio';


GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_azienda` TO
'Gestore_del_servizio';

GRANT EXECUTE ON procedure
`Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_corsa_treno_merci` TO 'Gestore_del_servizio';

GRANT EXECUTE ON procedure
`Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_corsa_treno_passeggeri` TO
'Gestore_del_servizio';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_lavoratore`
TO 'Gestore_del_servizio';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_spedizione`
TO 'Gestore_del_servizio';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_treno_merci`
TO 'Gestore_del_servizio';

GRANT EXECUTE ON procedure
`Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_treno_passeggeri` TO 'Gestore_del_servizio';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`crea_utente` TO
'Gestore_del_servizio';

GRANT EXECUTE ON procedure
`Trasporto_Ferroviario_Alta_Velocita`.`inserisci_orari_corsa_effettivi` TO 'Gestore_del_servizio';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`sostituisci_lavoratore`
TO 'Gestore_del_servizio';

GRANT EXECUTE ON procedure
`Trasporto_Ferroviario_Alta_Velocita`.`visualizza_storico_treno_merci` TO 'Gestore_del_servizio';
```

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`visualizza_storico_treno_passeggeri` TO 'Gestore_del_servizio';

SET SQL_MODE = '';

DROP USER IF EXISTS Lavoratore;

SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Lavoratore' IDENTIFIED BY 'Lavoratore';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`genera_report_sui_turni_di_lavoro` TO 'Lavoratore';

SET SQL_MODE = '';

DROP USER IF EXISTS Acquirente;

SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Acquirente' IDENTIFIED BY 'Acquirente';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`acquista_biglietto` TO 'Acquirente';

SET SQL_MODE = '';

DROP USER IF EXISTS Controllore;

SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Controllore' IDENTIFIED BY 'Controllore';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`verifica_biglietto` TO 'Controllore';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`convalida_biglietto` TO 'Controllore';

SET SQL_MODE = '';

DROP USER IF EXISTS Addetto_alla_manutenzione;

SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Addetto_alla_manutenzione' IDENTIFIED BY 'Addetto_alla_manutenzione';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_report_manutenzione_locomotrice` TO 'Addetto_alla_manutenzione';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_report_manutenzione_vagone_merci` TO 'Addetto_alla_manutenzione';

GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`aggiungi_report_manutenzione_vagone_passeggeri` TO 'Addetto_alla_manutenzione';

SET SQL_MODE = '';

DROP USER IF EXISTS Login;

SET SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE USER 'Login' IDENTIFIED BY 'Login';

```
GRANT EXECUTE ON procedure `Trasporto_Ferroviario_Alta_Velocita`.`login` TO 'Login';


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


-- ----------------------------------------------------
-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Azienda`
-- ----------------------------------------------------
START TRANSACTION;
USE `Trasporto_Ferroviario_Alta_Velocita`;
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Azienda` (`Partita_IVA`, `Recapito`, `Ragione_Sociale`) VALUES ('a', 'a', 'a');
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Azienda` (`Partita_IVA`, `Recapito`, `Ragione_Sociale`) VALUES ('b', 'b', 'b');


COMMIT;


-- ----------------------------------------------------
-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Locomotrice`
-- ----------------------------------------------------
START TRANSACTION;
USE `Trasporto_Ferroviario_Alta_Velocita`;
```

```
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Locomotrice` (`Id`, `Marca`, `Modello`) VALUES (1, 'a', 'a');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Locomotrice` (`Id`, `Marca`, `Modello`) VALUES (2, 'a', 'a');


COMMIT;



-- -----------------------------------------------------

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Passeggeri`

-- -----------------------------------------------------

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Passeggeri` (`Matricola`, `Numero_Vagoni`, `Data_di_Acquisto`, `Locomotrice`, `Numero_Carrozze_I_Classe`, `Numero_Carrozze_II_Classe`) VALUES ('2', 3, '2017-06-02', 2, 1, 2);


COMMIT;



-- -----------------------------------------------------

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri`

-- -----------------------------------------------------

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;
```

```
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri` (`Id`, `Marca`, `Modello`, `Classe`, `Numero_Massimo_di_Passeggeri`, `Treno`) VALUES (1, 'a', 'a', '1', 5, '2');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri` (`Id`, `Marca`, `Modello`, `Classe`, `Numero_Massimo_di_Passeggeri`, `Treno`) VALUES (2, 'b', 'b', '2', 4, '2');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Passeggeri` (`Id`, `Marca`, `Modello`, `Classe`, `Numero_Massimo_di_Passeggeri`, `Treno`) VALUES (3, 'b', 'b', '2', 4, '2');


COMMIT;



-- -----------------------------------------------------

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Posto`

-- -----------------------------------------------------

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (1, 1);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (2, 1);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (3, 1);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (4, 1);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (5, 1);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (1, 2);
```

```
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (2, 2);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (3, 2);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (4, 2);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (1, 3);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (2, 3);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (3, 3);

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Posto` (`Numero`, `Vagone_Passeggeri`) VALUES (4, 3);


COMMIT;



-- -----------------------------------------------------

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Stazione`

-- -----------------------------------------------------

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Stazione` (`Codice`, `Nome`, `Citta`, `Provincia`) VALUES (1, 'part', 'aq', 'aq');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Stazione` (`Codice`, `Nome`, `Citta`, `Provincia`) VALUES (2, 'arr', 'aq', 'aq');
```

```
COMMIT;


-- -----------------------------------------------------
-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Tratta`
-- -----------------------------------------------------
START TRANSACTION;
USE `Trasporto_Ferroviario_Alta_Velocita`;
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Tratta` (`Codice`, `Capolinea_di_Partenza`, `Capolinea_di_Arrivo`, `Orario_Partenza`, `Orario_Arrivo`) VALUES (1, 1, 2, '16:00:00', '17:00:00');

COMMIT;


-- -----------------------------------------------------
-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Corsa`
-- -----------------------------------------------------
START TRANSACTION;
USE `Trasporto_Ferroviario_Alta_Velocita`;
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` (`Data`, `Tratta`, `Orario_Effettivo_di_Partenza`, `Orario_Effettivo_di_Arrivo`) VALUES ('2021-01-01', 1, NULL, NULL);
```

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Corsa` (`Data`, `Tratta`, `Orario_Effettivo_di_Partenza`, `Orario_Effettivo_di_Arrivo`) VALUES ('2021-01-02', 1, NULL, NULL);

COMMIT;

-- ---

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Passeggeri`

-- ---

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Passeggeri` (`Data_Corsa`, `Tratta`, `Treno_Passeggeri`) VALUES ('2021-01-02', 1, '2');

COMMIT;

-- ---

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Lavoratore`

-- ---

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Lavoratore` (`Codice_Fiscale`, `Nome`, `Cognome`, `Data_di_Nascita`, `Citta_di_Nascita`, `Ruolo`, `Provincia_di_Nascita`) VALUES ('a', 'a', 'a', '2017-06-02', 'a', 'conducente', 'aq');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Lavoratore` (`Codice_Fiscale`, `Nome`, `Cognome`, `Data_di_Nascita`, `Citta_di_Nascita`, `Ruolo`, `Provincia_di_Nascita`) VALUES ('b', 'b', 'b', '2017-06-02', 'a', 'capotreno', 'aq');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Lavoratore` (`Codice_Fiscale`, `Nome`, `Cognome`, `Data_di_Nascita`, `Citta_di_Nascita`, `Ruolo`, `Provincia_di_Nascita`) VALUES ('c', 'c', 'c', '2017-06-02', 'c', 'conducente', 'aq');

COMMIT;

-- --

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Merci`

-- --

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Treno_Merci` (`Matricola`, `Numero_Vagoni`, `Data_di_Acquisto`, `Locomotrice`) VALUES ('1', 2, '2017-02-05', 1);

COMMIT;

-- --

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Merci`

-- --

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

```
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Corsa_Treno_Merci` (`Data_Corsa`, `Tratta`, `Treno_Merci`) VALUES ('2021-01-01', 1, '1');


COMMIT;



-- -----------------------------------------------------
-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Locomotrice`
-- -----------------------------------------------------
START TRANSACTION;
USE `Trasporto_Ferroviario_Alta_Velocita`;
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Locomotrice` (`Locomotrice`, `Testo`, `Timestamp`) VALUES (1, 'ok', '2021-04-18 16:10:59');
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Locomotrice` (`Locomotrice`, `Testo`, `Timestamp`) VALUES (2, 'ok', '2021-04-18 16:11:59');


COMMIT;



-- -----------------------------------------------------
-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Merci`
-- -----------------------------------------------------
START TRANSACTION;
USE `Trasporto_Ferroviario_Alta_Velocita`;
```

```
INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Merci` (`Id`, `Marca`, `Modello`, `Portata`, `Treno`) VALUES (1, 'a', 'a', 20, '1');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Vagone_Merci` (`Id`, `Marca`, `Modello`, `Portata`, `Treno`) VALUES (2, 'a', 'a', 20, '1');


COMMIT;



-- -----------------------------------------------------

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Merci`

-- -----------------------------------------------------

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Merci` (`Vagone_Merci`, `Testo`, `Timestamp`) VALUES (1, 'ok', '2021-04-18 16:12:59');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Merci` (`Vagone_Merci`, `Testo`, `Timestamp`) VALUES (2, 'not ok', '2021-04-18 16:13:59');


COMMIT;



-- -----------------------------------------------------

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Passeggeri`

-- -----------------------------------------------------

START TRANSACTION;
```

```
USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Passeggeri` (`Vagone_Passeggeri`, `Testo`, `Timestamp`) VALUES (1, 'ok', '2021-04-18 16:14:59');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Report_Manutenzione_Vagone_Passeggeri` (`Vagone_Passeggeri`, `Testo`, `Timestamp`) VALUES (2, 'ok', '2021-04-18 16:15:59');


COMMIT;



-- -----------------------------------------------------

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Turno`

-- -----------------------------------------------------

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Turno` (`Lavoratore`, `Data_Corsa`, `Tratta`, `Copertura`) VALUES ('a', '2021-01-01', 1, 'coperto');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Turno` (`Lavoratore`, `Data_Corsa`, `Tratta`, `Copertura`) VALUES ('a', '2021-01-02', 1, 'coperto');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Turno` (`Lavoratore`, `Data_Corsa`, `Tratta`, `Copertura`) VALUES ('b', '2021-01-02', 1, 'coperto');


COMMIT;



-- -----------------------------------------------------

-- Data for table `Trasporto_Ferroviario_Alta_Velocita`.`Utenti`
```

-- ---

START TRANSACTION;

USE `Trasporto_Ferroviario_Alta_Velocita`;

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('aldo', '0c88028bf3aa6a6a143ed846f2be1ea4', 'Lavoratore');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('giovanni', '0c88028bf3aa6a6a143ed846f2be1ea4', 'Gestore_del_servizio');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('giacomo', '0c88028bf3aa6a6a143ed846f2be1ea4', 'Addetto_alla_manutenzione');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('luigi', '0c88028bf3aa6a6a143ed846f2be1ea4', 'Controllore');

INSERT INTO `Trasporto_Ferroviario_Alta_Velocita`.`Utenti` (`Username`, `Password`, `Ruolo`) VALUES ('mario', '0c88028bf3aa6a6a143ed846f2be1ea4', 'Acquirente');

COMMIT;

Codice del Front-End

Acquirente.c

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

#include "defines.h"

static void buy_ticket(MYSQL *conn) {
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[8];
    MYSQL_TIME data;
    MYSQL_TIME nascita;

    // Input for the registration routine
    char tratta[46];
    int tratta_int;
    char classe[46];
    int classe_int;
    char codice_fiscale[46];
    char nome[46];
    char cognome[46];
    char cc[46];
```

```
    char data_anno[46];
    char data_mese[46];
    char data_giorno[46];

    char nascita_anno[46];
    char nascita_mese[46];
    char nascita_giorno[46];

    // Get the required information
    printf("\nRide Date: ");
    printf("\nYear: ");
    getInput(46, data_anno, false);
    printf("\nMonth: ");
    getInput(46, data_mese, false);
    printf("\nDay: ");
    getInput(46, data_giorno, false);
    printf("\nRailway Line: ");
    getInput(46, tratta, false);
    printf("\nClass: ");
    getInput(46, classe, false);
    printf("\nCodice Fiscale: ");
    getInput(46, codice_fiscale, false);
    printf("\nnome: ");
    getInput(46, nome, false);
    printf("\nCognome: ");
    getInput(46, cognome, false);
    printf("\nDate of Birth: ");
    printf("\nYear: ");
    getInput(46, nascita_anno, false);
    printf("\nMonth: ");
    getInput(46, nascita_mese, false);
    printf("\nDay: ");
    getInput(46, nascita_giorno, false);
    printf("\nCredit Card number: ");
    getInput(46, cc, false);

    // Apply proper type conversions
    tratta_int = atoi(tratta);
    classe_int = atoi(classe);

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

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

    param[0].buffer_type = MYSQL_TYPE_DATE;
    param[0].buffer = (char *) &data;
    param[0].buffer_length = sizeof(data);

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

    param[2].buffer_type = MYSQL_TYPE_LONG;
    param[2].buffer = &classe_int;
```

```
    param[2].buffer_length = sizeof(classe_int);

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

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

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

    param[6].buffer_type = MYSQL_TYPE_DATE;
    param[6].buffer = (char *) &nascita;
    param[6].buffer_length = sizeof(nascita);

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


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

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

    // Dump the result set
    dump_result_set(conn, prepared_stmt, "\nTicket data:");
    mysql_stmt_close(prepared_stmt);
}

void run_as_buyer(MYSQL *conn) {

        char options[3] = {'1', '2', '3'};
        char op;

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

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

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

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

```
            printf("2) Quit\n");

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

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

                case '2':
                    return;

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

            getchar();
        }
    }
```

Addetto_alla_manutenzione.c

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

#include "defines.h"

static void add_locomotive_maintenance(MYSQL *conn){

    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[2];

    char locomotrice[46];
    int locomotrice_int;
    char rapporto[200];

    // Get the required information
    printf("\nLocomotive Id: ");
    getInput(46, locomotrice, false);
    printf("Report: ");
    getInput(200, rapporto, false);

    // Apply proper type conversions
    locomotrice_int = atoi(locomotrice);

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

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

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

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

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

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

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

    mysql_stmt_close(prepared_stmt);
}


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

    char vagone[46];
    int vagone_int;
    char rapporto[200];

    // Get the required information
    printf("\nBoxcar Id: ");
    getInput(46, vagone, false);
    printf("Report: ");
    getInput(200, rapporto, false);

    // Apply proper type conversions
    vagone_int = atoi(vagone);

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

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

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

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

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind
```

```
parameters\n", true);
    }

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

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

    char carrozza[46];
    int carrozza_int;
    char rapporto[200];

    // Get the required information
    printf("\nPassenger Car Id: ");
    getInput(46, carrozza, false);
    printf("Report: ");
    getInput(200, rapporto, false);

    // Apply proper type conversions
    carrozza_int = atoi(carrozza);

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

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

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

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

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

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

```
    mysql_stmt_close(prepared_stmt);
}

void run_as_maintenance(MYSQL* conn) {

    char options[4] = {'1','2', '3', '4'};
   char op;

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

    if(!parse_config("users/Addetto_alla_manutenzione.json", &conf)) {
      fprintf(stderr, "Unable to load maintenance 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 locomotive maintenance report\n");
        printf("2) Add boxcar maintenance report\n");
        printf("3) Add passenger car maintenance report\n");
      printf("4) Quit\n");

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

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

            case '2':
            add_boxcar_maintenance(conn);
            break;

            case '3':
            add_passenger_car_maintenance(conn);
            break;

         case '4':
            return;

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

      getchar();
   }
}
```

Riportare (correttamente formattato) il codice C del thin client realizzato per interagire con la base di dati.

Sì, avete letto bene: **riportare il codice C**. Frasi del tipo "il codice è nei file allegati" non rispondono alla richiesta di riportare il codice C.

Controllore.c

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

#include "defines.h"

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

    char ticket[46];
    int ticket_int;

    // Get the required information
    printf("\nTicket Id: ");
    getInput(46, ticket, false);

    // Apply proper type conversions
    ticket_int = atoi(ticket);

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

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

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

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

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

    mysql_stmt_close(prepared_stmt);
}

static void check_ticket(MYSQL *conn){

    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[1];

    char ticket[46];
    int ticket_int;

    // Get the required information
    printf("\nTicket Id: ");
    getInput(46, ticket, false);
```

```
    // Apply proper type conversions
    ticket_int = atoi(ticket);

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

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

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

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

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

    // Dump the result set
    dump_result_set(conn, prepared_stmt, "\nList of exams assigned to you");
    mysql_stmt_close(prepared_stmt);
}

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

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

   if(!parse_config("users/Controllore.json", &conf)) {
      fprintf(stderr, "Unable to load inspector 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) Validate ticket\n");
      printf("2) Check ticket information\n");
        printf("3) Quit\n");

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

      switch(op) {
```

```
            case '1':
               validate_ticket(conn);
               break;

            case '2':
                   check_ticket(conn);
               break;

               case '3':
                   return;

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

         getchar();
      }
}
```

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_error(MYSQL *conn, char *message);
extern void finish_with_stmt_error(MYSQL *conn, MYSQL_STMT *stmt, char *message,
bool close_stmt);
extern bool setup_prepared_stmt(MYSQL_STMT **stmt, char *statement, MYSQL *conn);
extern void dump_result_set(MYSQL *conn, MYSQL_STMT *stmt, char *title);
extern void run_as_buyer(MYSQL *conn);
extern void run_as_maintenance(MYSQL *conn);
extern void run_as_administrator(MYSQL *conn);
extern void run_as_inspector(MYSQL* conn);
extern void run_as_employee(MYSQL* conn);
```

Gestore_del_servizio.c

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

#include "defines.h"

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

    char partita_iva[46];
    char recapito[46];
    char ragione_sociale[46];

    // Get the required information
    printf("\nPartita IVA: ");
    getInput(46, partita_iva, false);
    printf("\nAddress: ");
    getInput(46, recapito, false);
    printf("\nName: ");
    getInput(46, ragione_sociale, false);

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

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

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

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

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

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

    // Run procedure
    if (mysql_stmt_execute(prepared_stmt) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not add company\n",
true);
    } else {
        printf("RCompany correctly added...\n");
    }

}
```

```
static void add_passenger_car_ride(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[5];
    MYSQL_TIME data;

    // Input for the registration routine
    char tratta[46];
    int tratta_int;
    char treno[46];
    char conducente[46];
    char capotreno[46];
    char data_anno[46];
    char data_mese[46];
    char data_giorno[46];

    // Get the required information
    printf("\nDate: ");
    printf("\nYear: ");
    getInput(46, data_anno, false);
    printf("\nMonth: ");
    getInput(46, data_mese, false);
    printf("\nDay: ");
    getInput(46, data_giorno, false);
    printf("\nRailway Line: ");
    getInput(46, tratta, false);
    printf("\nTreno: ");
    getInput(46, treno, false);
    printf("\nDriver: ");
    getInput(46, conducente, false);
    printf("\nConductor: ");
    getInput(46, capotreno, false);

    // Apply proper type conversions
    tratta_int = atoi(tratta);
    data.year = atoi(data_anno);
    data.month = atoi(data_mese);
    data.day = atoi(data_giorno);

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

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

    param[0].buffer_type = MYSQL_TYPE_DATE;
    param[0].buffer = (char*)&data;
    param[0].buffer_length = sizeof(data);

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

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

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

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

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

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

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

    mysql_stmt_close(prepared_stmt);
}

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

    // Input for the registration routine
    char tratta[46];
    int tratta_int;
    char treno[46];
    char conducente[46];
    char data_anno[46];
    char data_mese[46];
    char data_giorno[46];

    // Get the required information
    printf("\nDate: ");
    printf("\nYear: ");
    getInput(46, data_anno, false);
    printf("\nMonth: ");
    getInput(46, data_mese, false);
    printf("\nDay: ");
    getInput(46, data_giorno, false);
    printf("\nRailway Line: ");
    getInput(46, tratta, false);
    printf("\nTreno: ");
    getInput(46, treno, false);
    printf("\nDriver: ");
    getInput(46, conducente, false);

    // Apply proper type conversions
    tratta_int = atoi(tratta);
    data.year = atoi(data_anno);
    data.month = atoi(data_mese);
    data.day = atoi(data_giorno);

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

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

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

    param[0].buffer_type = MYSQL_TYPE_DATE;
    param[0].buffer = (char*)&data;
    param[0].buffer_length = sizeof(data);

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

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

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

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

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

    mysql_stmt_close(prepared_stmt);
}

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

    char options[2] = {'1','2'};
    char r;
    // Input for the registration routine
    char codice_fiscale[46];
    char nome[46];
    char cognome[46];
    char citta[46];
    char ruolo[46];
    char provincia[46];
    char data_anno[46];
    char data_mese[46];
    char data_giorno[46];

    // Get the required information
    printf("\nCodice Fiscale: ");
    getInput(46, codice_fiscale, false);
    printf("Nome: ");
```

```
    getInput(46, nome, false);
    printf("Cognome: ");
    getInput(46, cognome, false);
    printf("Date of Birth: ");
    printf("\nYear: ");
    getInput(46, data_anno, false);
    printf("\nMonth: ");
    getInput(46, data_mese, false);
    printf("\nDay: ");
    getInput(46, data_giorno, false);

    printf("\nBirth city: ");
    getInput(46, citta, false);

    printf("Assign a possible role:\n");
    printf("\t1) Conducente\n");
    printf("\t2) Capotreno\n");
    r = multiChoice("Select role", options, 3);

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

    printf("\nBirth provincia: ");
    getInput(46, provincia, false);

    data.year = atoi(data_anno);
    data.month = atoi(data_mese);
    data.day = atoi(data_giorno);

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

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

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

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

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

```
    param[3].buffer_type = MYSQL_TYPE_DATE;
    param[3].buffer = (char*)&data;
    param[3].buffer_length = sizeof(data);

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

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

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

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

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

    mysql_stmt_close(prepared_stmt);
}

static void add_shipping(MYSQL *conn) {

    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[7];
    MYSQL_TIME data;

    // Input for the registration routine
    char mittente[46];
    char destinatario[46];
    char merce[46];
    char massa[46];
    int massa_int;
    char vagone[46];
    int vagone_int;
    char tratta[46];
    int tratta_int;
    char data_anno[46];
    char data_mese[46];
    char data_giorno[46];

    // Get the required information
    printf("\nSender: ");
    getInput(46, mittente, false);
    printf("Addressee: ");
    getInput(46, destinatario, false);
    printf("Goods: ");
    getInput(46, merce, false);
    printf("Weight: ");
    getInput(46, massa, false);
```

```
    printf("Boxcar: ");
    getInput(46, vagone, false);
    printf("Shipment date: ");
    printf("\nYear: ");
    getInput(46, data_anno, false);
    printf("\nMonth: ");
    getInput(46, data_mese, false);
    printf("\nDay: ");
    getInput(46, data_giorno, false);
    printf("Railway Line: ");
    getInput(46, tratta, false);

    // Apply proper type conversions
    massa_int = atoi(massa);
    vagone_int = atoi(vagone);
    tratta_int = atoi(tratta);

    data.year = atoi(data_anno);
    data.month = atoi(data_mese);
    data.day = atoi(data_giorno);

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

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

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

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

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

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

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

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

    param[6].buffer_type = MYSQL_TYPE_LONG;
    param[6].buffer = &tratta_int;
    param[6].buffer_length = sizeof(tratta_int);

    if (mysql_stmt_bind_param(prepared_stmt, param) != 0) {
        finish_with_stmt_error(conn, prepared_stmt, "Could not bind
```

```
parameters\n", true);
    }

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

    mysql_stmt_close(prepared_stmt);

}

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

    // Input for the registration routine
    char matricola[46];
    char numero_vagoni[46];
    int numero_vagoni_int;
    char marca_locomotrice[46];
    char modello_locomotrice[46];
    char marca_vagone[46];
    char modello_vagone[46];
    char portata[46];
    int portata_int;
    char data_anno[46];
    char data_mese[46];
    char data_giorno[46];

    // Get the required information
    printf("\nSerial Number: ");
    getInput(46, matricola, false);
    printf("\nNumero Vagoni: ");
    getInput(46, numero_vagoni, false);
    printf("\nDate of Purchase: ");
    printf("\nYear: ");
    getInput(46, data_anno, false);
    printf("\nMonth: ");
    getInput(46, data_mese, false);
    printf("\nDay: ");
    getInput(46, data_giorno, false);
    printf("\nLocomotive Brand: ");
    getInput(46, marca_locomotrice, false);
    printf("\nLocomotive Model: ");
    getInput(46, modello_locomotrice, false);
    printf("\nBoxcar Brand: ");
    getInput(46, marca_vagone, false);
    printf("\nBoxcar Model: ");
    getInput(46, modello_vagone, false);
    printf("\nCarrying Capacity: ");
    getInput(46, portata, false);

    // Apply proper type conversions
    numero_vagoni_int = atoi(numero_vagoni);
    portata_int = atoi(portata);
```

```
    data.year = atoi(data_anno);
    data.month = atoi(data_mese);
    data.day = atoi(data_giorno);

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

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

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

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

    param[2].buffer_type = MYSQL_TYPE_DATE;
    param[2].buffer = (char*)&data;
    param[2].buffer_length = sizeof(data);

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

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

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

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

    param[7].buffer_type = MYSQL_TYPE_LONG;
    param[7].buffer = &portata_int;
    param[7].buffer_length = sizeof(portata_int);

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

    // 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_passenger_train(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[13];
    MYSQL_TIME data;

    // Input for the registration routine
    char matricola[46];
    char numero_vagoni[46];
    int numero_vagoni_int;
    char carrozze_prima[46];
    int carrozze_prima_int;
    char carrozze_seconda[46];
    int carrozze_seconda_int;
    char marca_locomotrice[46];
    char modello_locomotrice[46];
    char marca_vagone_prima[46];
    char modello_vagone_prima[46];
    char passeggeri_prima[46];
    int passeggeri_prima_int;
    char marca_vagone_seconda[46];
    char modello_vagone_seconda[46];
    char passeggeri_seconda[46];
    int passeggeri_seconda_int;
    char data_anno[46];
    char data_mese[46];
    char data_giorno[46];


    // Get the required information
    printf("\nSerial Number: ");
    getInput(46, matricola, false);
    printf("\nNumero Vagoni: ");
    getInput(46, numero_vagoni, false);
    printf("\nDate of Purchase: ");
    printf("\nYear: ");
    getInput(46, data_anno, false);
    printf("\nMonth: ");
    getInput(46, data_mese, false);
    printf("\nDay: ");
    getInput(46, data_giorno, false);
    printf("\nNumber of I Class Passenger Cars: ");
    getInput(46, carrozze_prima, false);
    printf("\nNumber of II Class Passenger Cars: ");
    getInput(46, carrozze_seconda, false);
    printf("\nLocomotive Brand: ");
    getInput(46, marca_locomotrice, false);
    printf("\nLocomotive Model: ");
    getInput(46, modello_locomotrice, false);
    printf("\nI Class Passenger Car Brand: ");
    getInput(46, marca_vagone_prima, false);
    printf("\nI Class Passenger Car Model: ");
    getInput(46, modello_vagone_prima, false);
    printf("\nMax I Class Passenger in a Car: ");
    getInput(46, passeggeri_prima, false);
    printf("\nII Class Passenger Car Brand: ");
    getInput(46, marca_vagone_seconda, false);
    printf("\nII Class Passenger Car Model: ");
    getInput(46, modello_vagone_seconda, false);
    printf("\nMax II Class Passenger in a Car: ");
    getInput(46, passeggeri_seconda, false);
```

```
    // Apply proper type conversions
    numero_vagoni_int = atoi(numero_vagoni);
    carrozze_prima_int = atoi(carrozze_prima);
    carrozze_seconda_int = atoi(carrozze_seconda);
    passeggeri_prima_int = atoi(passeggeri_prima);
    passeggeri_seconda_int = atoi(passeggeri_seconda);

    data.year = atoi(data_anno);
    data.month = atoi(data_mese);
    data.day = atoi(data_giorno);

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

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

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

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

    param[2].buffer_type = MYSQL_TYPE_DATE;
    param[2].buffer = (char*)&data;
    param[2].buffer_length = sizeof(data);

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

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

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

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

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

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

    param[9].buffer_type = MYSQL_TYPE_LONG;
    param[9].buffer = &passeggeri_prima_int;
```

```
    param[9].buffer_length = sizeof(passeggeri_prima_int);

    param[10].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[10].buffer = marca_vagone_seconda;
    param[10].buffer_length = strlen(marca_vagone_seconda);

    param[11].buffer_type = MYSQL_TYPE_VAR_STRING;
    param[11].buffer = modello_vagone_seconda;
    param[11].buffer_length = strlen(modello_vagone_seconda);

    param[12].buffer_type = MYSQL_TYPE_LONG;
    param[12].buffer = &passeggeri_seconda_int;
    param[12].buffer_length = sizeof(passeggeri_seconda_int);

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

    // 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_user(MYSQL *conn)
{
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[3];
    char options[5] = {'1','2', '3', '4', '5'};
    char r;

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

    // Get the required information
    printf("\nUsername: ");
    getInput(46, username, false);
    printf("password: ");
    getInput(46, password, true);
    printf("Assign a possible role:\n");
    printf("\t1) Acquirente\n");
    printf("\t2) Addetto alla manutenzione\n");
    printf("\t3) Controllore\n");
    printf("\t4) Gestore del servizio\n");
    printf("\t5) Lavoratore\n");
    r = multiChoice("Select role", options, 3);

    // Convert role into enum value
    switch(r) {
        case '1':
            strcpy(ruolo, "Acquirente");
            break;
        case '2':
```

```
            strcpy(ruolo, "Addetto_alla_manutenzione");
            break;
        case '3':
            strcpy(ruolo, "Controllore");
            break;
        case '4':
            strcpy(ruolo, "Gestore_del_servizio");
            break;
        case '5':
            strcpy(ruolo, "Lavoratore");
            break;
        default:
            fprintf(stderr, "Invalid condition at %s:%d\n", __FILE__, __LINE__);
            abort();
    }

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

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

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

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

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

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

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

    mysql_stmt_close(prepared_stmt);
}

static void add_ride_datetimes(MYSQL *conn){

    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[4];
    MYSQL_TIME data;
    MYSQL_TIME partenza;
    MYSQL_TIME arrivo;
```

```
    // Input for the registration routine
    char tratta[46];
    int tratta_int;
    char data_anno[46];
    char data_mese[46];
    char data_giorno[46];

    char partenza_anno[46];
    char partenza_mese[46];
    char partenza_giorno[46];

    char partenza_ora[46];
    char partenza_minuto[46];
    char partenza_secondo[46];

    char arrivo_anno[46];
    char arrivo_mese[46];
    char arrivo_giorno[46];

    char arrivo_ora[46];
    char arrivo_minuto[46];
    char arrivo_secondo[46];

    // Get the required information
    printf("\nRide Date: ");
    printf("\nYear: ");
    getInput(46, data_anno, false);
    printf("\nMonth: ");
    getInput(46, data_mese, false);
    printf("\nDay: ");
    getInput(46, data_giorno, false);
    printf("\nRailway Line: ");
    getInput(46, tratta, false);
    printf("\nDeparture Datetime: ");
    printf("\nYear: ");
    getInput(46, partenza_anno, false);
    printf("\nMonth: ");
    getInput(46, partenza_mese, false);
    printf("\nDay: ");
    getInput(46, partenza_giorno, false);
    printf("\nHour: ");
    getInput(46, partenza_ora, false);
    printf("\nMinute: ");
    getInput(46, partenza_minuto, false);
    printf("\nSecond: ");
    getInput(46, partenza_secondo, false);
    printf("\nArrival Datetime: ");
    printf("\nYear: ");
    getInput(46, arrivo_anno, false);
    printf("\nMonth: ");
    getInput(46, arrivo_mese, false);
    printf("\nDay: ");
    getInput(46, arrivo_giorno, false);
    printf("\nHour: ");
    getInput(46, arrivo_ora, false);
    printf("\nMinute: ");
    getInput(46, arrivo_minuto, false);
    printf("\nSecond: ");
    getInput(46, arrivo_secondo, false);
```

```
    // Apply proper type conversions
    tratta_int = atoi(tratta);

    data.year = atoi(data_anno);
    data.month = atoi(data_mese);
    data.day = atoi(data_giorno);

    partenza.year = atoi(partenza_anno);
    partenza.month = atoi(partenza_mese);
    partenza.day = atoi(partenza_giorno);

    partenza.hour = atoi(partenza_ora);
    partenza.minute = atoi(partenza_minuto);
    partenza.second = atoi(partenza_secondo);

    arrivo.year = atoi(arrivo_anno);
    arrivo.month = atoi(arrivo_mese);
    arrivo.day = atoi(arrivo_giorno);

    arrivo.hour = atoi(arrivo_minuto);
    arrivo.minute = atoi(arrivo_minuto);
    arrivo.second = atoi(arrivo_secondo);

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

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

    param[0].buffer_type = MYSQL_TYPE_DATE;
    param[0].buffer = (char*)&data;
    param[0].buffer_length = sizeof(data);

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

    param[0].buffer_type = MYSQL_TYPE_DATETIME;
    param[0].buffer = (char*)&partenza;
    param[0].buffer_length = sizeof(partenza);

    param[0].buffer_type = MYSQL_TYPE_DATETIME;
    param[0].buffer = (char*)&arrivo;
    param[0].buffer_length = sizeof(arrivo);

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

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

```
    mysql_stmt_close(prepared_stmt);

}

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

    // Input for the registration routine
    char malato[46];
    char sostituto[46];
    char tratta[46];
    int tratta_int;

    char data_anno[46];
    char data_mese[46];
    char data_giorno[46];

    // Get the required information
    printf("\nShift Date: ");
    printf("\nYear: ");
    getInput(46, data_anno, false);
    printf("\nMonth: ");
    getInput(46, data_mese, false);
    printf("\nDay: ");
    getInput(46, data_giorno, false);
    printf("\nSick Employee: ");
    getInput(46, malato, false);
    printf("\nSubstitute Employee: ");
    getInput(46, sostituto, false);
    printf("\nRailway Line: ");
    getInput(46, tratta, false);

    // Apply proper type conversions
    tratta_int = atoi(tratta);

    data.year = atoi(data_anno);
    data.month = atoi(data_mese);
    data.day = atoi(data_giorno);

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

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

    param[0].buffer_type = MYSQL_TYPE_DATE;
    param[0].buffer = (char*)&data;
    param[0].buffer_length = sizeof(data);

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

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

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

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

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

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

    mysql_stmt_close(prepared_stmt);
}

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

    char treno[46];

    // Get the required information
    printf("\nGood Train Serial Number: ");
    getInput(46, treno, false);

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

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

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

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

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

    // Dump the result set
    dump_result_set(conn, prepared_stmt, "\nMaintenance history of the selected
train");
```

```
    mysql_stmt_close(prepared_stmt);
}

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

    char treno[46];

    // Get the required information
    printf("\nPassenger Train Serial Number: ");
    getInput(46, treno, false);

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

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

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

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

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

    // Dump the result set
    dump_result_set(conn, prepared_stmt, "\nMaintenance history of the selected
train");
    mysql_stmt_close(prepared_stmt);
}

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

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

   if(!parse_config("users/Gestore_del_servizio.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 company\n");
   printf("2) Add passenger car ride\n");
   printf("3) Add boxcar ride\n");
   printf("4) Add employee\n");
     printf("5) Add shipping\n");
     printf("6) Add goods train\n");
     printf("7) Add passenger train\n");
     printf("8) Add user\n");
     printf("9) Go to page 2\n");

   op = multiChoice("Select an option", options, 5);

   switch(op) {
      case '1':
         add_company(conn);
         break;
      case '2':
         add_passenger_car_ride(conn);
         break;
      case '3':
         add_boxcar_ride(conn);
         break;
      case '4':
         add_employee(conn);
         break;
      case '5':
         add_shipping(conn);
         break;
      case '6':
         add_goods_train(conn);
         break;
      case '7':
         add_passenger_train(conn);
         break;
      case '8':
         add_user(conn);
         break;
      case '9':
            printf("\033[2J\033[H");
            printf("*** What should I do for you? ***\n\n");
            printf("1) Add real ride datetimes\n");
            printf("2) Substitute employee\n");
            printf("3) View goods train maintenance history\n");
            printf("4) View passengers train maintenance history\n");
            printf("5) Quit\n");

            op = multiChoice("Select an option", options, 5);

            switch(op) {
                case '1':
                    add_ride_datetimes(conn);
                    break;
                case '2':
                    substitute_employee(conn);
                    break;
                case '3':
                    view_goods_train_maintenance_history(conn);
                    break;
```

```
                    case '4':
                        view_passengers_train_maintenance_history(conn);
                        return;
                    case '5':
                        return;

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

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

        getchar();
    }
}
```

Inout.c

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

#include "defines.h"

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

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

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

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

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

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

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

 // Acquisisce da tastiera al pi 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  stata data
        if(c == '\0') { // getInput() non pu restituire '\n'!
            return predef;
        } else if(c == yes) {
            return true;
        } else if(c == no) {
            return false;
        } else if(c == toupper(yes)) {
            if(predef || insensitive) return true;
        } else if(c == toupper(yes)) {
            if(!predef || insensitive) return false;
        }
    }
}
```

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

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

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

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

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

Lavoratore.c

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

#include "defines.h"

static void add_shifts_report(MYSQL *conn){
    MYSQL_STMT *prepared_stmt;
    MYSQL_BIND param[3];
    MYSQL_TIME data_corsa;
    MYSQL_TIME data_report;

    char op;

    char lavoratore[16];
    char rapporto[500];
    int tratta_int;
    char tratta[46];

    // Get the required information
    printf("\nCodice Fiscale: ");
    getInput(46, lavoratore, false);
    printf("Report: ");
    getInput(500, rapporto, false);

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

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

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

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

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

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

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

    mysql_stmt_close(prepared_stmt);

    while(true){
        char options[2] = {'1','2'};
        printf("If you don't want to add a shift to the report press '1', else
press another key\n");
        op = multiChoice("Select an option", options, 2);

        if (op == '1') {
            return;
        }

        // Get the required information
        printf("\nRide Date: ");
        printf("\nYear: ");
        getInput(46, (char*)&data_corsa.year, false);
        printf("\nMonth: ");
        getInput(46, (char*)&data_corsa.month, false);
        printf("\nDay: ");
        getInput(46, (char*)&data_corsa.day, false);
        printf("\nRailway Line: ");
        getInput(46, tratta, false);
        printf("Report Date: ");
        printf("\nYear: ");
        getInput(46, (char*)&data_report.year, false);
        printf("\nMonth: ");
        getInput(46, (char*)&data_report.month, false);
        printf("\nDay: ");
        getInput(46, (char*)&data_report.day, false);

        // Apply proper type conversions
        tratta_int = atoi(tratta);
```

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

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

        param[0].buffer_type = MYSQL_TYPE_DATE;
        param[0].buffer = (char*)&data_corsa;
        param[0].buffer_length = sizeof(data_corsa);

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

        param[2].buffer_type = MYSQL_TYPE_DATE;
        param[2].buffer = (char*)&data_report;
        param[2].buffer_length = sizeof(data_report);

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

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

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

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

   if(!parse_config("users/Lavoratore.json", &conf)) {
      fprintf(stderr, "Unable to load employee 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 shifts report\n");
      printf("2) Quit\n");

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

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

         case '2':
            return;

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

      getchar();
   }
}
```

Main.c

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

#include "defines.h"


typedef enum {
   ADMINISTRATOR = 1,
   BUYER,
   MAINTENANCE,
   INSPECTOR,
   EMPLOYEE,
   FAILED_LOGIN
} role_t;


struct configuration conf;

static MYSQL *conn;


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

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

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

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

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

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

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

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

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

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

    // Prepare output parameters
    memset(param, 0, sizeof(param));
    param[0].buffer_type = MYSQL_TYPE_LONG; // OUT
    param[0].buffer = &role;
    param[0].buffer_length = sizeof(role);

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

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

    mysql_stmt_close(login_procedure);
    return role;

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

int main(void) {
    role_t role;

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

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

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

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

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

   switch(role) {
      case BUYER:
         run_as_buyer(conn);
         break;

      case MAINTENANCE:
         run_as_maintenance(conn);
         break;

      case INSPECTOR:
         run_as_inspector(conn);
         break;

      case EMPLOYEE:
         run_as_employee(conn);
         break;

      case ADMINISTRATOR:
         run_as_administrator(conn);
         break;

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

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

   printf("Bye!\n");

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

Parse.c

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

```c
#include "defines.h"

#define BUFF_SIZE 4096

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

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

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

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

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

/**
 * Allocates a fresh unused token from the token pool.
 */
```

```
static jsmntok_t *jsmn_alloc_token(jsmn_parser *parser, jsmntok_t *tokens, size_t
num_tokens) {
   jsmntok_t *tok;
   if (parser->toknext >= num_tokens) {
      return NULL;
   }
   tok = &tokens[parser->toknext++];
   tok->start = tok->end = -1;
   tok->size = 0;
#ifdef JSMN_PARENT_LINKS
   tok->parent = -1;
#endif
   return tok;
}

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

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

   start = parser->pos;

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

found:
   if (tokens == NULL) {
      parser->pos--;
      return 0;
```

```
    }
    token = jsmn_alloc_token(parser, tokens, num_tokens);
    if (token == NULL) {
        parser->pos = start;
        return JSMN_ERROR_NOMEM;
    }
    jsmn_fill_token(token, JSMN_PRIMITIVE, start, parser->pos);
#ifdef JSMN_PARENT_LINKS
    token->parent = parser->toksuper;
#endif
    parser->pos--;
    return 0;
}

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

    int start = parser->pos;

    parser->pos++;

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

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

        /* Backslash: Quoted symbol expected */
        if (c == '\\' && parser->pos + 1 < len) {
            int i;
            parser->pos++;
            switch (js[parser->pos]) {
                /* Allowed escaped symbols */
                case '\"': case '/' : case '\\' : case 'b' :
                case 'f' : case 'r' : case 'n'  : case 't' :
                    break;
                /* Allows escaped symbol \uXXXX */
                case 'u':
                    parser->pos++;
                    for(i = 0; i < 4 && parser->pos < len && js[parser->pos] != '\0';
i++) {
                        /* If it isn't a hex character we have an error */
```

```
                if(!((js[parser->pos] >= 48 && js[parser->pos] <= 57) || /* 0-9
*/
                        (js[parser->pos] >= 65 && js[parser->pos] <= 70) || /*
A-F */
                        (js[parser->pos] >= 97 && js[parser->pos] <= 102))) {
/* a-f */
                    parser->pos = start;
                    return JSMN_ERROR_INVAL;
                }
                parser->pos++;
             }
             parser->pos--;
             break;
          /* Unexpected symbol */
          default:
             parser->pos = start;
             return JSMN_ERROR_INVAL;
       }
    }
   }
   parser->pos = start;
   return JSMN_ERROR_PART;
}

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

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

      c = js[parser->pos];
      switch (c) {
         case '{': case '[':
            count++;
            if (tokens == NULL) {
               break;
            }
            token = jsmn_alloc_token(parser, tokens, num_tokens);
            if (token == NULL)
               return JSMN_ERROR_NOMEM;
            if (parser->toksuper != -1) {
               tokens[parser->toksuper].size++;
#ifdef JSMN_PARENT_LINKS
               token->parent = parser->toksuper;
#endif
            }
            token->type = (c == '{' ? JSMN_OBJECT : JSMN_ARRAY);
            token->start = parser->pos;
            parser->toksuper = parser->toknext - 1;
            break;
         case '}': case ']':
            if (tokens == NULL)
               break;
```

```
            type = (c == '}' ? JSMN_OBJECT : JSMN_ARRAY);
#ifdef JSMN_PARENT_LINKS
            if (parser->toknext < 1) {
               return JSMN_ERROR_INVAL;
            }
            token = &tokens[parser->toknext - 1];
            for (;;) {
               if (token->start != -1 && token->end == -1) {
                  if (token->type != type) {
                     return JSMN_ERROR_INVAL;
                  }
                  token->end = parser->pos + 1;
                  parser->toksuper = token->parent;
                  break;
               }
               if (token->parent == -1) {
                  if(token->type != type || parser->toksuper == -1) {
                     return JSMN_ERROR_INVAL;
                  }
                  break;
               }
               token = &tokens[token->parent];
            }
#else
            for (i = parser->toknext - 1; i >= 0; i--) {
               token = &tokens[i];
               if (token->start != -1 && token->end == -1) {
                  if (token->type != type) {
                     return JSMN_ERROR_INVAL;
                  }
                  parser->toksuper = -1;
                  token->end = parser->pos + 1;
                  break;
               }
            }
            /* Error if unmatched closing bracket */
            if (i == -1) return JSMN_ERROR_INVAL;
            for (; i >= 0; i--) {
               token = &tokens[i];
               if (token->start != -1 && token->end == -1) {
                  parser->toksuper = i;
                  break;
               }
            }
#endif
            break;
         case '\"':
            r = jsmn_parse_string(parser, js, len, tokens, num_tokens);
            if (r < 0) return r;
            count++;
            if (parser->toksuper != -1 && tokens != NULL)
               tokens[parser->toksuper].size++;
            break;
         case '\t' : case '\r' : case '\n' : case ' ':
            break;
         case ':':
            parser->toksuper = parser->toknext - 1;
            break;
         case ',':
            if (tokens != NULL && parser->toksuper != -1 &&
                  tokens[parser->toksuper].type != JSMN_ARRAY &&
```

```
                    tokens[parser->toksuper].type != JSMN_OBJECT) {
#ifdef JSMN_PARENT_LINKS
                parser->toksuper = tokens[parser->toksuper].parent;
#else
                for (i = parser->toknext - 1; i >= 0; i--) {
                   if (tokens[i].type == JSMN_ARRAY || tokens[i].type ==
JSMN_OBJECT) {
                      if (tokens[i].start != -1 && tokens[i].end == -1) {
                         parser->toksuper = i;
                         break;
                      }
                   }
                }
#endif
             }
             break;
#ifdef JSMN_STRICT
          /* In strict mode primitives are: numbers and booleans */
          case '-': case '0': case '1' : case '2': case '3' : case '4':
          case '5': case '6': case '7' : case '8': case '9':
          case 't': case 'f': case 'n' :
             /* And they must not be keys of the object */
             if (tokens != NULL && parser->toksuper != -1) {
                jsmntok_t *t = &tokens[parser->toksuper];
                if (t->type == JSMN_OBJECT ||
                      (t->type == JSMN_STRING && t->size != 0)) {
                   return JSMN_ERROR_INVAL;
                }
             }
#else
          /* In non-strict mode every unquoted value is a primitive */
          default:
#endif
             r = jsmn_parse_primitive(parser, js, len, tokens, num_tokens);
             if (r < 0) return r;
             count++;
             if (parser->toksuper != -1 && tokens != NULL)
                tokens[parser->toksuper].size++;
             break;

#ifdef JSMN_STRICT
          /* Unexpected char in strict mode */
          default:
             return JSMN_ERROR_INVAL;
#endif
       }
    }

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

    return count;
}

/**
```

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

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

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

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

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

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

   config[fsize] = 0;
   return fsize;
}

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

   load_file(path);

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

   /* Assume the top-level element is an object */
   if (r < 1 || t[0].type != JSMN_OBJECT) {
```

```
      printf("Object expected\n");
      return 0;
   }

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

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

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

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

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

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

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

   mysql_field_seek (res_set, 0);

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

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

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

   print_dashes(res_set);
}

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

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

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

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

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

      dump_result_set_header(rs_metadata);

      fields = mysql_fetch_fields(rs_metadata);

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

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

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

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

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

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

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

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

         putchar('|');

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

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

            switch (rs_bind[i].buffer_type) {

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

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

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

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

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

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

```
                    break;

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

      mysql_free_result(rs_metadata); /* free metadata */

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