UniTrain: Relazione Finale 23/06/2022

Studenti: Federico Pizzari (1936451) Giorgio Proietti (1914925)

REQUISITI MINIMI

Requisito A Lo schema deve prendere un domino che consenta di identificare più di 12 entità principali	L'ER iniziale conta più di venti entità concettuali principali
Requisito B Lo schema deve avere una complessità tale per cui lo schema ER contiene sia relazioni ISA che generalizzazioni.	Lo schema ER contiene sia relazioni ISA che generalizzazioni
Requisito C Deve esserci una struttura sufficiente nelle relazioni. Questo comporta che se si vede il diagramma ER come un grafo questo deve contenere dei cicli.	All'interno dello schema ER sono presenti diversi cicli
Requisito D Lo schema deve contenere vincoli di cardinalità sulla partecipazione delle entità con relazioni diverse dal valore predefinito	La maggior parte delle relazioni nello schema possiede vincoli di cardinalità con valore diverso dal predefinito.
Requisito E Lo schema deve contenere alcuni attributi facoltativi e alcuni multi-valore.	Lo schema contiene attributi facoltativi e alcuni multi-valore
Requisito F Lo schema deve contenere almeno quattro vincoli esterni	Lo schema contiene diverse decine di vincoli esterni
Requisito G Le specifiche devono includere un'indicazione dei volumi per le varie entità e relazioni	Il documento qui presente riporta sia le indicazioni dei volumi prima che le indicazioni dei volumi dopo la ristrutturazione dello schema ER
Requisito H Le specifiche devono includere un carico di lavoro delle query e delle operazioni più comuni (tra 10 e 15) che sono di interesse nel dominio modellato, con un'indicazione della loro frequenza.	Lo schema contiene 15 tra le operazioni più comuni incluso il loro carico di lavoro e 15 query
Requisito H.1	Tutte le query riportate richiedono l'utilizzo degli operatori aggregati
Requisito H.2	Tutte le query riportate (ad esclusione di quelle riscritte con le view) contengono query annidate. Inoltre, più di una query possiede tre livelli
Requisito H.3	Le metà delle query è stata riscritta utilizzando le viste
Requisito H.4	Tutte le query coinvolgono almeno due tabelle

Indice

Specifica dei Requisiti

Abbiamo la necessità di rappresentare una azienda ferroviaria alta velocità che effettua traffico passeggeri a livello nazionale

Dobbiamo quindi prima di tutto tenere conto di tutti i treni a nostra disposizione. Un treno sarà composto da diverse carrozze e un suo modello, oltre ad avere il suo identificativo. Un treno deve effettuare la manutenzione per verificare che sia sempre funzionale o per riparare guasti.

Un treno ovviamente è basato su un modello. Del modello abbiamo ovviamente il suo nome e la sua velocità massima.

Una carrozza di un treno ha una classe di viaggio, ovvero quell'indicativo che rappresenta il comfort che un passeggero si può aspettare in modo molto generale durante un viaggio.

Un treno ha poi una sua composizione, ovvero vogliamo sapere di quali carrozze è composto.
Una carrozza è composta dalla sua classe di viaggio e dal suo "numero identificativo". Inoltre, per ogni carrozza vogliamo sapere il numero di posti che sono presenti al loro interno.

Un posto è composto dal suo numero identificativo e nella carrozza e da un valore che ci indica se il posto è privilegiato per i diversamente abili.
Abbiamo inoltre la necessità di tenere conto del livello di occupazione del treno per ogni viaggio che esso effettua.

I passeggeri sono coloro che effettivamente viaggiano sui nostri treni. Di loro dobbiamo salvare nome cognome e una e-mail per poterli contattare in caso di comunicazioni sul viaggio. Volendo, è anche possibile inserire il proprio numero di telefono per inviare le comunicazioni anche via SMS. Un passeggero potrebbe avere più numeri di telefono nel caso voglia ricevere le comunicazioni per viaggi diversi su cellulari diversi.

I passeggeri ottengono il diritto di viaggiare su un treno acquistando un biglietto. Un biglietto dà la possibilità di viaggiare su un treno ad un passeggero per arrivare da una stazione X ad una stazione Y. Ogni biglietto ha ovviamente un prezzo che scala in base a diversi fattori, tra cui livello di servizio scelto e KM effettuati.

I passeggeri inoltre possono essere in grado di creare un loro account, con la quale acquistare viaggi e collezionare punti che possono dare diritto a viaggi gratis. Inoltre, per ogni account vogliamo sapere la lista dei viaggi che ha acquistato.

Un passeggero qualsiasi deve essere in grado di effettuare delle richieste di assistenza, che poi possono essere più specifiche in base a se si riferiscono all'account, ad un biglietto acquistato, oppure semplicemente relative ad informazioni. Di una richiesta abbiamo bisogno della data e dell'ora di scrittura, del testo e dell'e-mail dell'utente che ha inviato la richiesta. Inoltre, vogliamo sapere la lingua con la quale la richiesta è stata scritta, oltre a tenere conto di quali addetti al customer care hanno risposto e in che modo

Ogni treno effettua più spostamenti nell'arco di una giornata. Di uno spostamento vogliamo sapere il treno che lo effettua, il personale di bordo incaricato della gestione e il giorno di partenza. Per ogni spostamento ci vogliono un macchinista e un capotreno.

Se il treno in quell'itinerario effettua servizio viaggiatori, allora è da considerare anche un viaggio. Un viaggio ha un suo identificativo che si ripete giornalmente se l'itinerario è lo stesso e parte alla stessa ora.

Inoltre, ricade in una sua categoria, caratteristica che condivide con il treno che lo effettuerà. Per un viaggio, sul treno abbiamo bisogno anche degli assistenti, ovvero gli "hostess e stewards".

Un treno durante uno spostamento segue un percorso, che ha un suo ID, un suo orario di partenza e di arrivo. Un percorso può essere fatto anche in giorni diversi, ma sempre alla stessa ora.

Ogni percorso segue una serie di sezioni, zone in cui il viaggio è diviso e nella quale noi verifichiamo gli orari per capire quanto il treno sia effettivamente in ritardo. Le sezioni sono costruite sulla base di tratte preesistenti, che non sono altro che i percorsi fisici costituiti da binari di cui la rete ferroviaria è composta. Di una sezione vogliamo quindi sapere l'ora di ingresso teorica ed effettiva, l'ora di uscita teorica ed effettiva, la sosta programmata nel punto finale della tratta per quel treno e la sosta effettiva. Inoltre, vogliamo avere un modo per ordinare queste sezioni in un viaggio.

Una tratta a sua volta è invece composta da un limite di velocità (fare ben attenzione che la stessa tratta può avere limiti di velocità diversi) e la sua lunghezza (per calcolare il prezzo del biglietto)

Il punto di partenza e il punto di arrivo di una tratta viene definito "punto di interesse". Esso può o essere un luogo nelle vicinanze di un deviatoio, un punto in cui c'è un cambio di velocità, un semaforo, oppure una stazione/deposito.
Un punto di interesse ha quindi un nome e il suo posizionamento in coordinate sulla mappa.
Nel caso un punto di interesse sia o una stazione o un deposito abbiamo bisogno di più informazioni.
Per la stazione, abbiamo bisogno del numero di binari presenti e l'indirizzo, oltre all'informazione se la stazione è accessibile ai diversamente abili.
Per il deposito abbiamo bisogno di sapere l'indirizzo e una lista di tutti i banchi di lavoro presenti nel caso un treno abbia bisogno di manutenzione.

Per ogni banco di lavoro vogliamo avere il suo ID e i modelli per la quale il banco è indirizzato (ergo quali modelli possono fare la manutenzione in quel banco di lavoro)

Quando un treno effettua la manutenzione, vogliamo sapere data di inizio della manutenzione, la descrizione della manutenzione effettuata e il treno controllato.
Vogliamo inoltre sapere anche quale banco è stato occupato, per quanto tempo, e quali operai hanno effettuato la manutenzione.

Le manutenzioni, e tutte le altre attività di gestione e svolgimento delle operazioni, vengono effettuate dagli Impiegati. Degli impiegati vogliamo saper il loro Codice Fiscale, il loro Nome, Cognome, l'IBAN e se sono Operai, Segretari o Ferrovieri, oltre ai loro vari turni di lavoro.
Ogni impiegato ha inoltre un contratto, della quale si vogliono sapere gli estremi legali.
Abbiamo 3 tipologie di impiegati:
Gli operai, gli incaricati ad effettuare le manutenzioni dei treni, della quale vogliamo sapere i depositi in cui lavorano;
I segretari, gli incaricati a svolgere le parti burocratiche del lavoro e la gestione del Customer care, come rispondere alle richieste di assistenza clienti. Dei seguenti vogliamo sapere gli uffici in cui lavorano e le lingue che parlano, oltre alla tipologia di segretario (Dirigente, Addetto al Customer Care ecc);
I Ferrovieri, coloro che staranno sui treni durante gli spostamenti, della quale vogliamo sapere la tipologia di ferroviere (capotreno, macchinista o Hostess) oltre a sapere in quali treni hanno effettuato/effettueranno servizio
Degli uffici vogliamo sapere quali sono gli impiegati passati e futuri, il dirigente attuale e la posizione e la tipologia dell'ufficio.

PRIMA PARTE – PROGETTAZIONE CONCETTUALE

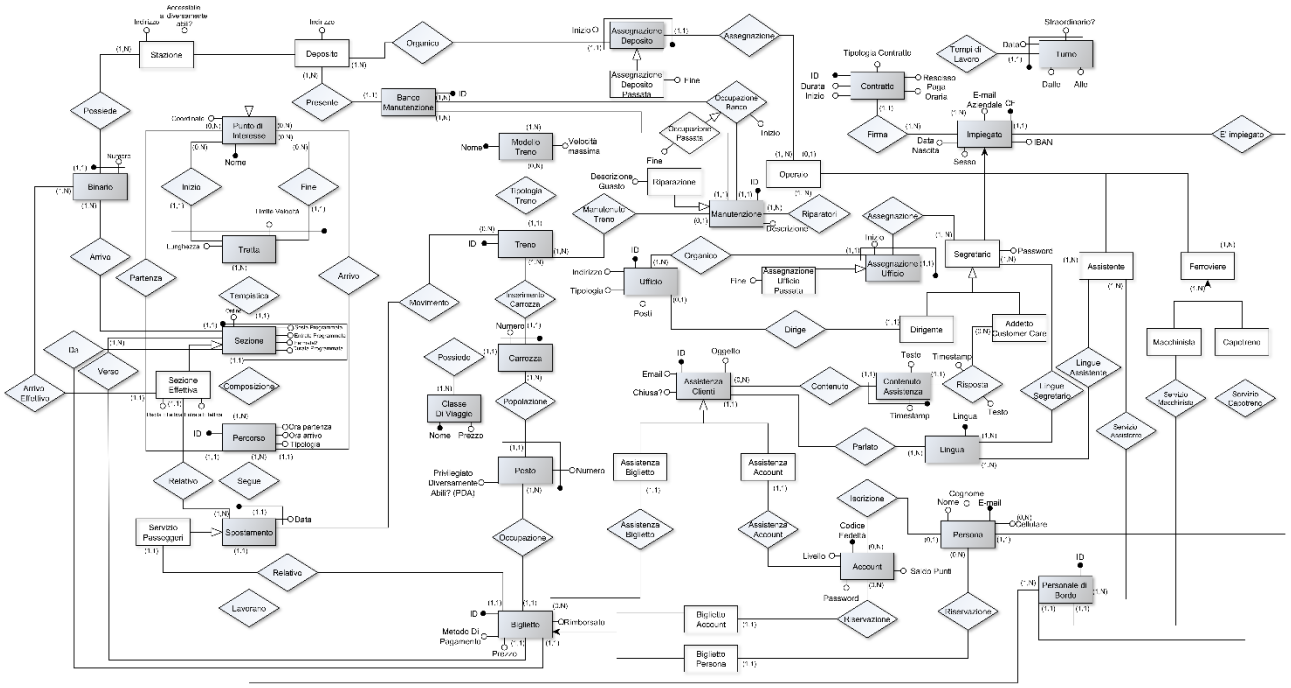
Glossario dei Termini

Entità	Descrizione	Sinonimi	Collegamenti
Account	Un passeggero può avere un account, un servizio che gli permette di avere la lista di tutti i viaggi effettuati e da effettuare e l'accesso ad un saldo punti fedeltà	Utente	Biglietto, Assistenza, Persona
Assegnazione Deposito	Rappresenta l'assegnazione di un operaio ad un deposito nel tempo		Operaio, Deposito
Assegnazione Ufficio	Rappresenta l'assegnazione di un segretario ad un ufficio nel tempo		Segretario, Ufficio
Assistente	Rappresenta una "hostess" o uno "steward"	Hostess, Steward	
Assistenza Clienti	Registra la richiesta di assistenza per un qualsivoglia problema riscontrato utilizzando i servizi dell'azienda	Aiuto	Account, Biglietto, Contenuto Assistenza, Lingue
Banco Manutenzione	Rappresenta delle zone presenti all'interno dei depositi che ci permettono di effettuare la manutenzione dei nostri treni	Banco	Modello Treno, Manutenzione
Biglietto	Rappresenta un biglietto, il documento con la quale un passeggero ha il diritto di viaggiare su uno o più treni per arrivare a destinazione		Account, Assistenza
Binario	Rappresenta un binario di una stazione dove i treni fanno sosta		Stazione, Sezione, Sezione Effettiva
Carrozza	Rappresenta la carrozza, parte del treno dove sono situati i posti nella quale i passeggeri si siederanno. Può avere diverse classi di viaggio e può essere rimossa da un treno e inserita su un altro	Vagoni	Inserimento, Posto, Modello Carrozza, Classe Di Viaggio
Classe Di Viaggio	Rappresenta uno standard di servizi e di comodità che i clienti si possono aspettare durante il viaggio.	Ambiente di Viaggio	Carrozza
Contenuto Assistenza	Il contenuto effettivo di una delle tante possibili interazioni con il servizio clienti per una richiesta di assistenza		Assistenza, Segretario
Contratto	Rappresenta il contratto stipulato tra l'azienda e l'impiegato che ne registra le informazioni contrattuali che entrambe le parti hanno accettato		Impiegato
Deposito	Rappresenta un punto di interesse in cui un treno non fa servizio passeggeri, dove può stazionare indefinitamente e ricevere manutenzione		Punto Di Interesse, Operaio, Banco Manutenzione
Direzione Ufficio	Rappresenta chi sono i segretari che dirigono un ufficio		Ufficio, Segretario
Ferroviere	Rappresenta un impiegato che svolge il ruolo di assicurare il corretto movimento del treno		Spostamento

Impiegato	Rappresenta una persona che svolge un qualche ruolo nell'azienda		Operaio, Segretario, Ferroviere, Contratto, Turno
Lingua	Rappresenta una lingua del mondo		Segretario, Ferroviere, Assistenza
Manutenzione	Rappresenta una specifica manutenzione eseguita su un treno e/o carrozza		Treno, Operaio, Banco, Carrozza
Modello Treno	Rappresenta un modello di un treno		Modello Carrozza, Treno, Ferroviere
Operaio	Rappresenta la persona che svolge la manutenzione del materiale rotabile		Manutenzione, Impiegato, Deposito
Percorso	Rappresenta le sezioni di percorso che un treno deve effettuare	Itinerario	Sezione, Percorso
Persona	Rappresenta le generalità di una persona fisica		Biglietto, Account, Impiegato
Personale di bordo	Rappresenta un gruppo di impiegati situati sui treni che solitamente svolgono gli stessi percorsi		Ferroviere, Spostamento
Posto	Rappresenta un posto a sedere presente in una carrozza		Carrozza, Biglietto
Punto di Interesse	Rappresenta il luogo in cui un treno passa ed in cui ne viene registrata la posizione. Un punto di questo genere può essere: un deviatoio, un semaforo, una stazione, un deposito.	Luogo	Tratta, Stazione, Deposito
Riparatori	Rappresenta la partecipazione di un determinato operaio ad una determinata manutenzione.	Manutentori	Operaio, Manutenzione
Risposta	Rappresenta la risposta che l'assistenza fornisce ad una determinata richiesta di assistenza da parte di un cliente		Segretario, Assistenza
Segretario	Rappresenta un impiegato che lavora in un ufficio e svolge le operazioni burocratiche di dovere della azienda		Assegnazione Ufficio, Assistenza, Lingua, Impiegato, Direzione Ufficio
Sezione	Rappresenta le informazioni della sezione che un treno percorre in un dato percorso	Zona	Percorso, Tratta
Spostamento	Rappresenta uno spostamento di un determinato treno all'interno della rete ferroviaria	Movimento	Percorso, Treno, Biglietto, Ferroviere
Stazione	Rappresenta una stazione passeggeri all'interno della rete ferroviaria dove i passeggeri possono scendere o salire	Fermata	Punto Di Interesse, Biglietto
Tratta	Rappresenta le caratteristiche del tragitto presente tra due punti di interesse	Tratto	Punto di Interesse, Sezione
Treno	Rappresenta un Treno, inteso come motrice di un certo modello, che traporta un certo numero di carrozze.		Modello treno, Manutenzione, Inserimento, Spostamento
Turno	Rappresenta un turno di lavoro di un dato impiegato		Impiegato

Ufficio	Rappresenta un luogo dove i segretari vanno per svolgere il loro lavoro		Assegnazione Ufficio, Direzione Ufficio

Primo Schema ER



Per vedere l'immagine ingrandita dello schema ER, fare CTRL+clicl (immagine hostata su imgur)

Dizionario dei Dati

Entità

Entità	Descrizione	Vincoli Esterni	Identificatore
Account	Un passeggero può avere un account, un servizio che gli permette di avere la lista di tutti i viaggi effettuati e da effettuare e l'accesso ad un saldo punti fedeltà		Codice Fedeltà, Persona (Esterno)
Assegnazione Deposito	Rappresenta l'assegnazione di un operaio ad un deposito nel tempo	Un operaio non può essere assegnato a due depositi diversi contemporaneamente	Inizio, Deposito (esterno), Operaio (esterno)
Assegnazione Ufficio	Rappresenta l'assegnazione di un segretario ad un ufficio nel tempo	Un segretario non può essere assegnato a due uffici diversi contemporaneamente Quando rimuoviamo un segretario da un ufficio, fare attenzione al fatto che non sia il dirigente attuale	Inizio, Ufficio (esterno), Segretario (esterno)
Assistente ISA Impiegato	Rappresenta una "hostess" o uno "steward"	Bisogna controllare se l'impiegato è presente nella lista degli	Impiegato (Esterno)

		impiegati attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)	
Assistenza Account ISA Assistenza Clienti	Registra la richiesta di assistenza da parte di un possessore di account per un qualsivoglia problema riscontrato utilizzando i servizi dell'azienda.		Account (Esterno)
Assistenza Biglietto ISA Assistenza Clienti	Registra la richiesta di assistenza da parte di un possessore di biglietto per un qualsivoglia problema riscontrato utilizzando i servizi dell'azienda.		Biglietto (Esterno)
Assistenza Clienti	Registra la richiesta di assistenza per un qualsivoglia problema riscontrato utilizzando i servizi dell'azienda		Identificativo, Lingua (Esterno)
Banco Manutenzioni	Rappresenta delle zone presenti all'interno dei depositi che ci permettono di effettuare la manutenzione dei nostri treni		Identificativo, Deposito (Esterno)
Biglietto	Rappresenta un biglietto, il documento con la quale un passeggero ha il diritto di viaggiare su uno o più treni per arrivare a destinazione		Identificativo
Biglietto Account ISA Biglietto	Rappresenta un biglietto legato ad un account	Ogni volta che si aggiunge un elemento bisogna aumentare il totale del saldo punti dell'account relativo e controllare se si è saliti di livello. Nel caso, aggiornare il livello. Gli account per la quale vengono accumulati punti siano gli stessi definiti all'interno del nome	Account (Esterno)
Biglietto Persona ISA Biglietto	Rappresenta un biglietto legato ad una persona		Persona (Esterno)
Binario	Rappresenta un binario di una stazione dove i treni fanno sosta		Stazione (esterno)
Carrozza	Rappresenta la carrozza, parte del treno dove sono situati i i posti nella quale i passeggeri si siederanno. Può avere diverse classi di viaggio e può		Numero, Treno (Esterno), Classe Viaggio (Esterno)

	essere rimossa da un treno e inserita su un altro		
Classe Viaggio	Rappresenta uno standard di servizi e di comodità che i clienti si possono aspettare durante il viaggio.		Nome
Contenuto Assistenza	Il contenuto effettivo di una delle tante possibili interazioni con il servizio clienti per una richiesta di assistenza	Bisogna controllare se a rispondere sono solo segretari autorizzati (ovvero che lavorano null'ufficio del supporto clienti) e che parlino la lingua della persona che ha scritto il messaggio originale	Assistenza (Esterno), Timestamp Richiesta, Segretario (Esterno)
Contratto	Rappresenta il contratto stipulato tra l'azienda e l'impiegato che ne registra le informazioni contrattuali che entrambe le parti hanno accettato		Identificativo
Deposito ISA Punto di Interesse	Rappresenta un punto di interesse in cui un treno non fa servizio passeggeri, dove può stazionare indefinitamente e ricevere manutenzione		
Direzione Ufficio	Rappresenta chi sono i segretari che dirigono un ufficio	Durante l'inserimento di completare l'incarico al dirigente precedente e di controllare che lavori nell'ufficio nella quale si sta dando l'incarico	Ufficio, Segretario
Impiegato	Rappresenta una persona che svolge un qualche ruolo nell'azienda		Codice Fiscale, Persona (Esterno)
Ferroviere ISA Impiegato	Rappresenta un impiegato che svolge il ruolo di assicurare il corretto movimento del treno	Bisogna controllare se il ferroviere è presente nella lista dei ferrovieri attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)	
Lingua	Rappresenta una lingua del mondo		Lingua
Manutenzione	Rappresenta una specifica manutenzione eseguita su un treno e/o carrozza	Il treno su cui si sta facendo la manutenzione, sia presente nel deposito Controllare che il banco che si sta	Identificativo, Treno (Esterno), Banco (Esterno)

		cercando di utilizzare non sia già occupato	
Modello Treno	Rappresenta un modello di un treno		Nome
Operaio ISA Impiegato	Rappresenta un Operaio	Controlliamo se l'operaio è presente nella lista degli operai attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)	Codice Fiscale
Percorso	Rappresenta le sezioni di percorso che un treno deve effettuare	Non modificare un percorso se è già associato ad uno spostamento e aggiornare il percorso solo con dati validi In fase iniziale di creazione, non inserire l'ora di arrivo e l'ora di partenza	Identificativo, Punto di Interesse (Esterno)
Persona	Rappresenta le generalità di una persona fisica		E-mail
Personale di Bordo	Rappresenta un gruppo di impiegati situati sui treni che solitamente svolgono gli stessi percorsi	Il capotreno deve essere un capotreno e il macchinista deve essere un macchinista.	Identificativo, Capotreno (Esterno), Macchinista (Esterno)
Posto	Rappresenta un posto a sedere presente in una carrozza		Numero, Carrozza (Esterno), Treno (Esterno)
Punto di Interesse	Un luogo di particolare importanza per il treno e/o i passeggeri, come stazioni, depositi, deviatoi ecc		Nome
Segretario ISA Impiegato	Rappresenta un impiegato che lavora in un ufficio e svolge le operazioni burocratiche di dovere della azienda	Controlliamo se il segretario è presente nella lista dei segretari attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)	
Sezione	Rappresenta le informazioni della sezione che un treno percorre in un dato percorso	Ogni volta che si aggiunge una sezione, aggiornare l'ora di arrivo e il punto di interesse di arrivo. Inoltre, non si possono aggiungere sezioni ad	Percorso (Esterno), Tratta (Esterno), Ordine

		un percorso che ha già uno spostamento Inoltre controllare se la sezione è compatibile con la tipologia di percorso specificato Non si possono modificare le sezioni. Nel caso una sezione faccia fermata, controllare che sia in una stazione. Se si passa in una stazione, controllare che il binario sia inserito correttamente Non si può cancellare una sezione che è già associata ad uno spostamento o che abbia valore di ordine minore del massimo valore per quel percorso. Nel caso si possa cancellare la sezione, modificare i dati del percorso.	
Sezione Effettiva ISA Sezione	Rappresenta le informazioni reali della sezione che un treno ha percorso in un dato itinerario	Quando si aggiorna il campo uscita effettiva di una sezione bisogna controllare che sia l'ultima dell'itinerario. Nel caso sia l'ultima, controllare che i ferrovieri interessati non abbiano superato l'orario del turno. Nel caso, aggiungere lo straordinario L'orario di entrata deve essere maggiore dell'orario di uscita della sezione	Identificativo

		precedente (se esiste), l'ora di uscita della sezione corrente deve essere maggiore di quello d'entrata	
Spostamento	Rappresenta uno spostamento di un determinato treno all'interno della rete ferroviaria	I ferrovieri devono lavorare durante il loro turno e non devono essere assegnati ad un altro spostamento Gli spostamenti non si devono accavallare su una determinata tratta Un treno può partire solo dal punto di arrivo dell'ultimo viaggio Non cancellare gli spostamenti effettuati nel passato o che non siano l'ultimo spostamento programmato per quel treno L'ultima stazione referenziata dal percorso deve essere anche una fermata se la tipologia del percorso è impostata a "Viaggio"	Percorso (Esterno), Treno (Esterno), Giorno, Personale di Bordo (Esterno)
Stazione ISA Punto di Interesse	Rappresenta una stazione passeggeri all'interno della rete ferroviaria dove i passeggeri possono scendere o salire		
Tratta	Rappresenta le caratteristiche del tragitto presente tra due punti di interesse		Limite Velocità, Punto di Interesse (Esterno)
Treno	Rappresenta un Treno, inteso come motrice di un certo modello, che traporta un certo numero di carrozze.		Identificativo, Modello Treno (Esterno)
Turno	Rappresenta un turno di lavoro di un dato impiegato	I turni di un impiegato non si devono accavallare	Impiegato (Esterno), Giorno, Straordinario

		Il turno di un impiegato deve ricadere in un periodo in cui lui ha il contratto Non cancellare un turno nel passato o uno nel futuro in cui un ferroviere è impegnato Un turno non deve essere più lungo di 4 ore se la tipologia di contratto dell'impiegato è part-time	
Ufficio	Rappresenta un luogo dove i segretari vanno per svolgere il loro lavoro		Identificativo

Relazioni

Lingua Assistente	Rappresenta le lingue parlate da un assistente		Segretario (Esterno), Lingua (Esterno)
Lingua Segretario	Rappresenta le lingue parlate da un segretario		Segretario (Esterno), Lingua (Esterno)
Riparatori	Rappresenta la partecipazione di un determinato operaio ad una determinata manutenzione.		Manutenzione (Esterno), Operaio (Esterno)
Servizio Assistente	Rappresenta il servizio che fa un assistente nel personale di bordo		Ferroviere (Esterno), Personale di Treno (Esterno)

Tavola dei Volumi

Nome	Tipologia	Volume (indicativo)	Spiegazione
Account	E	250.000 (All'incirca 25 mila in più all'anno)	All'incirca 5 milioni di clienti sul suolo nazionale con un rate di iscrizione pari 1 su 20
Assegnazione Deposito	E	150	Su 75 operai, la metà avrà cambiato deposito, le metà sarà stata licenziata
Assegnazione Ufficio	E	400	Guardare Assegnazione Deposito
Assistenza Clienti	E	400.000 (all'anno)	Considerando circa 8 milioni di record tra biglietto, persone e account, e considerando che solo il 5% dei record richiedono assistenza
Banco Manutenzione	E	15	Su 30 treni circolanti è ideale avere almeno la metà in banchi

Biglietto	E	Cinque milioni (all'anno)	(30 treni circolanti * 3 viaggi per treno al giorno in media * 365 giorni * 40 posti per carrozza in media * 8 carrozze) /2 (un treno è riempito a metà in media)
Binario	E	150	5*30 stazioni
Carrozza	E	420	40 treni * 8 carrozze per treno
Classe di Viaggio	E	4	Quattro possibili classi di viaggio
Contenuto Assistenza	E	1 milione (all'anno)	400 mila richieste di assistenza * 2.5 interazioni medie
Contratto	E	450 (Ogni 5 anni)	450 impiegati attivi in contemporanea
Direzione Ufficio	E	5	5 Uffici, ogni ufficio ha il suo capo
Impiegato	E	900	All'incirca: 50 capotreni, 50 macchinisti, 125 assistenti, 25 operai, 200 segretari. Raddoppiare per coprire eventuali licenziamenti
Lingua	E	24	Lingue ufficiali unione europea
Lingue assistente	R	700	4 lingue conosciute in media * 175 ferrovieri
Lingue segretario	R	600	3 lingue conosciute in media * 200
Manutenzione	E	550 (all'anno)	12 manutenzioni all'anno * 40 treni + 1 guasto a settimana* 52 settimane in un anno
Modello Treno	E	3	La nostra azienda utilizza 3 modelli diversi di treni
Percorso	E	150	(30 treni circolanti* 3 percorsi al giorno per treno + 2 per rientrare/uscire dai depositi) + 10 percorsi effettuati solo in alcuni giorni della settimana
Persona	E	3 milioni (All'incirca 100.000 in più all'anno)	Platea dei nostri percorsi. La nostra espansione dei percorsi nel tempo potrebbe portare all'incirca 100mila passeggeri in più all'anno
Personale di Bordo	E	150	30 treni circolanti* 5 percorsi al giorno per treno + 10 percorsi effettuati solo in alcuni giorni della settimana
Posto	E	16.000	50 posti * 8 carrozze * 40 treni
Punto di Interesse	E	5000	Mediamente su un percorso: 1 semaforo ogni 1KM * 800 KM di tracciato + 0.1 deviatoio ogni 1 KM * 800 KM + 30 stazioni servite + 3 Depositi
Riparatori	R	4000 (All'anno)	400 manutenzioni all'anno * 10 operai per manutenzione
Servizio Assistente	R	500	125 assistenti * 3 viaggi diversi al giorno + eventuali
Sezione	E	20 milioni (all'anno)	In media una tratta viene attraversata una decina di volte al giorno * 365 giorni * 5000 tratte
Spostamento	E	55 mila (all'anno)	30 treni * 5 viaggi per treno in media * 365 giorni
Tratta		5000	All'incirca come i punti di interesse
Treno	E	40	Flotta iniziale. 30 treni circolanti + 10 sostitutivi. Potrebbe aumentare nel caso l'azienda si espanda
Turno	E	350.000	1 turno al giorno in media (considerando anche i possibili straordinari) * 900 impiegati * 365 giorni
Ufficio	E	5	5 uffici, uno per ogni tipologia

Tavola delle Operazioni

Operazione	Descrizione	Frequenza	Tipo
Inserimento Persona	Inserimento di una persona che non ha mai preso un treno con la nostra azienda	280/giorno	I
Creazione Account	Creazione di un nuovo account per una persona	70/giorno	I
Acquisto Biglietto	Inserimento di un biglietto all'interno della base di dati	14.000/giorno	I
Visualizzazione Viaggi	Visualizzazione dei viaggi disponibili e i loro relativi prezzi	50.000/giorno	I
Inserimento Percorso	Inserimento di un nuovo Percorso nella base di dati	100/anno	I
Inserimento Spostamento	Inserimento di un nuovo spostamento nella base di dati	45.000/anno	I
Nuovo Contratto	Rinnovo o Inserimento di un contratto di un impiegato	450/anno	I
Inserimento nuovo Impiegato	Inserimento di un nuovo impiegato	150/anno	I
Calcolo ritardo	Calcolo del ritardo medio e totale degli spostamenti di una giornata	1/giorno	B
Calcolo guadagno giornata	Calcolo del guadagno effettivo della giornata	1/giorno	B
Manutenzione	Inserimento di un treno in manutenzione	50/mese	I
Inserimento Personale di bordo	Inserimento di un gruppo di ferrovieri e assistenti nella tabella personale di bordo	100/anno	I
Visualizzazione Arrivi in Stazione	Visualizzazione dei prossimi treni in arrivo nella stazione	20 milioni/anno	I
Visualizza Partenze in Stazione	Visualizzazione dei prossimi treni in partenza nella stazione	20 milioni/anno	I
Calcolo retribuzione di un impiegato	Calcolo della retribuzione di un impiegato	450/mese	B

Tavola degli Accessi (Delle 5 operazioni principali)

Visualizzazione Viaggi

Concetto	Costrutto	Accessi	Tipo
Spostamento	Entità	1	Lettura
Punto di Interesse	Entità	2	Lettura
Tratta	Entità	1	Lettura
Classe Viaggio	Entità	1	Lettura
Percorso	Entità	3	Lettura
Sezione	Entità	4	Lettura

Calcolo Retribuzione di un Impiegato

Concetto	Costrutto	Accessi	Tipo
Turno	Entità	1	Lettura
Contratto	Entità	1	Lettura

Acquisto Biglietto

Concetto	Costrutto	Accessi	Tipo
Spostamento	Entità	1	Lettura
Carrozza	Entità	1	Lettura
Posto	Entità	2	Lettura
Biglietto	Entità	3	Lettura
Account	Entità	1	Lettura
Tratta	Entità	1	Lettura
Sezione	Entità	3	Lettura
Biglietto	Entità	2	Scrittura

Inserimento Personale di Bordo

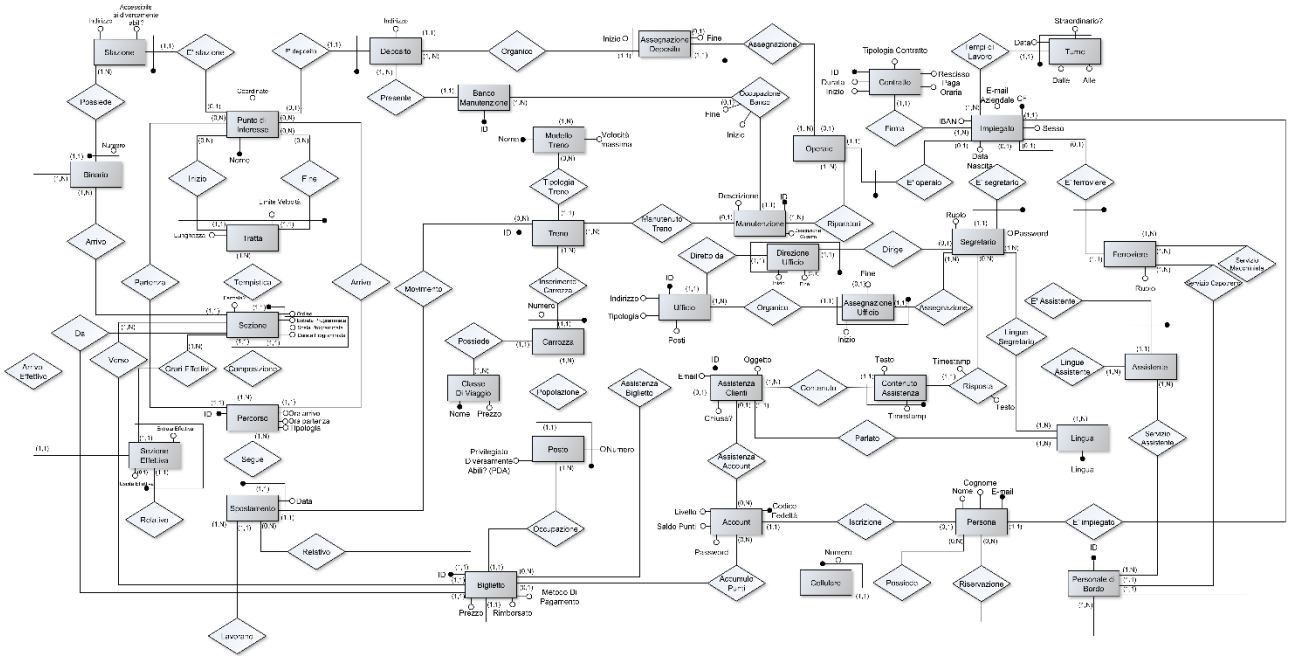
Concetto	Costrutto	Accessi	Tipo
Personale di Bordo	Entità	1	Lettura
Personale di Bordo	Entità	1	Scrittura
Servizio Assistente	Relazione	N con N uguale agli assistenti che fanno parte del personale di bordo	Scrittura

Visualizzazione Arrivi Stazione

Concetto	Costrutto	Accessi	Tipo
Treno	Entità	1	Lettura
Percorso	Entità	1	Lettura
Spostamento	Entità	1	Lettura
Sezione	Entità	1	Lettura

SECONDA PARTE – RISTRUTTURAZIONE DELLO SCHEMA CONCETTUALE

Schema ER Ristrutturato



Per vedere l'immagine ingrandita dello schema ER, fare CTRL+clicl (immagine hostata su imgur)

Spiegazione delle modifiche

- **Create le entità Stazione e Deposito (Metodo 3)**
  Visto la necessità di avere un gran numero di accessi sia a stazione che a deposito, e viste le numerose relazioni che dipendono da queste due, abbiamo deciso di separarle dalla entità padre e di renderle delle vere e proprie entità
- **Creata l'entità Stazione Effettiva (Metodo 3 + modifica dell'ER)**
  Abbiamo creato l'entità stazione effettiva, anche se in realtà abbiamo deciso di non applicarlo "alla lettera" visto e considerato che nel mentre della ristrutturazione, abbiamo notato una problematica, ovvero che la sezione effettiva non deve solo dipendere dalla sezione, ma anche dagli spostamenti che un treno fa su quel percorso. Abbiamo quindi deciso di applicare il metodo 3, ma di estendere la chiave non solo al padre, ma anche alla relazione che l'entità aveva (e ha) con Spostamento.
- **Accorpato Servizio Passeggeri all'interno di Spostamento (Metodo 1)**
  Vista la semplicità del concetto, abbiamo deciso di accorparlo direttamente al padre e di aggiungere il campo tipologia per capire quando rendere lo spostamento aperto al traffico passeggeri
- **Accorpati ai padri tutti i concetti che esprimevano del tempo (Metodo 1)**
  Abbiamo deciso di unire questi passaggi visto il collegamento logico che c'è tra loro. Visto che questi concetti contenevano solamente l'informazione di quando una assegnazione o un procedimento era terminato (o se non era ancora terminato nel caso il concetto non esisteva) abbiamo deciso di usare una regola generare e accorpare tutti questi figli al padre, inserendo gli attributi accorpati con una cardinalità (0,1)
- **Accorpata la Riparazione alla Manutenzione (Metodo 1)**

Visto che il concetto della riparazione è fortemente legato a quello della manutenzione (un treno che si guasta deve essere riparato, e quindi deve entrare in manutenzione), abbiamo deciso di unire i due concetti in un'unica entità, alla quale abbiamo fornito un campo in cui specifichiamo se la manutenzione è iniziata per un guasto o per un semplice controllo di routine.

- **Accorpate le diverse tipologie di Assistenza Clienti (Metodo 1)**
  Una Assistenza Clienti si risolve sempre allo stesso modo, l'unica differenza è sul contenuto che l'utente fornisce per aprirla. Visto che il nostro punto di interesse non è tanto l'apertura del caso, ma più che altro la risoluzione in modo veloce e il più indolore possibile, abbiamo deciso di unire anche in questo caso i concetti. Sarà poi in base ai contenuti forniti dall'utente che il sistema in automatico salverà le informazioni in modo tale da salvare tramite un attributo in fase di creazione del record di quale tipologia di assistenza stiamo parlando
- **Accorpati le diverse tipologie di Biglietto (Metodo 1)**
  Anche in questo caso l'unione è la scelta più immediata, visto che fornire tipologie di biglietti diversi in base a come lo si è acquistato non è una scelta intelligente, anche perché abbiamo comunque bisogno sul biglietto delle generalità della persona che possiede l'account, quindi l'accorpamento è la scelta ideale
- **Creazione delle entità figlie a Impiegato (Metodo 3)**
  Visto e considerato che applicare il Metodo 2 era impossibile visto che sarebbe venuto un gran pasticcio con le entità contratto e le entità turno, abbiamo considerato il Metodo 3 l'unico approccio possibile visto e considerato l'importanza e le relazioni che queste entità hanno nel nostro DB
- **Creata entità Cellulare**
  Visto che Persona aveva un attributo multi-valore in cellulare, in fase di ristrutturazione abbiamo creato l'entità Cellulare e fatto in modo che puntasse all'entità persona

E questo completa la ristrutturazione del nostro DB

Dizionario dei Dati

Entità

Entità	Descrizione	Vincoli Esterni	Identificatore
Account	Un passeggero può avere un account, un servizio che gli permette di avere la lista di tutti i viaggi effettuati e da effettuare e l'accesso ad un saldo punti fedeltà		Codice Fedeltà, Persona (Esterno)
Assegnazione Deposito	Rappresenta l'assegnazione di un operaio ad un deposito nel tempo	Un operaio non può essere assegnato a due depositi diversi contemporaneamente	Inizio, Deposito (esterno), Operaio (esterno)
Assegnazione Ufficio	Rappresenta l'assegnazione di un segretario ad un ufficio nel tempo	Un segretario non può essere assegnato a due depositi diversi contemporaneamente Quando rimuoviamo un segretario da un ufficio, fare attenzione al fatto che non sia il dirigente attuale	Inizio, Ufficio (esterno), Segretario (esterno)

Assistente	Rappresenta una “hostess” o uno “steward”	Bisogna controllare se l’impiegato è presente nella lista degli impiegati attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)	Impiegato (Esterno)
Assistenza Clienti	Registra la richiesta di assistenza per un qualsivoglia problema riscontrato utilizzando i servizi dell’azienda		Identificativo, Lingua (Esterno), Account (Esterno), Biglietto (Esterno)
Banco Manutenzioni	Rappresenta delle zone presenti all’interno dei depositi che ci permettono di effettuare la manutenzione dei nostri treni		Identificativo, Deposito (Esterno)
Biglietto	Rappresenta un biglietto, il documento con la quale un passeggero ha il diritto di viaggiare su uno o più treni per arrivare a destinazione	Ogni volta che si aggiunge un elemento bisogna aumentare il totale del saldo punti dell’account relativo e controllare se si è saliti di livello. Nel caso, aggiornare il livello. Gli account per la quale vengono accumulati punti siano gli stessi definiti all’interno del nome	Identificativo, Account (Esterno), Persona (Esterno)
Binario	Rappresenta un binario di una stazione dove i treni fanno sosta		Stazione (esterno)
Carrozza	Rappresenta la carrozza, parte del treno dove sono situati i posti nella quale i passeggeri si siederanno. Può avere diverse classi di viaggio e può essere rimossa da un treno e inserita su un altro		Numero, Treno (Esterno), Classe Viaggio (Esterno)
Cellulare	Contiene l’associazione tra numero di cellulare ed e-mail del possessore		Numero Telefonico, Account (Esterno)
Classe Viaggio	Rappresenta uno standard di servizi e di comodità che i clienti si possono aspettare durante il viaggio.		Nome
Contenuto Assistenza	Il contenuto effettivo di una delle tante possibili interazioni con il servizio clienti per una richiesta di assistenza	Bisogna controllare se a rispondere sono solo segretari autorizzati (ovvero che lavorano	Assistenza (Esterno), Timestamp Richiesta, Segretario (Esterno)

		null'ufficio del supporto clienti) e che parlino la lingua della persona che ha scritto il messaggio originale	
Contratto	Rappresenta il contratto stipulato tra l'azienda e l'impiegato che ne registra le informazioni contrattuali che entrambe le parti hanno accettato		Identificativo
Deposito	Rappresenta un punto di interesse in cui un treno non fa servizio passeggeri, dove può stazionare indefinitamente e ricevere manutenzione		Nome
Direzione Ufficio	Rappresenta chi sono i segretari che dirigono un ufficio	Durante l'inserimento di completare l'incarico al dirigente precedente e di controllare che lavori nell'ufficio nella quale si sta dando l'incarico	Ufficio, Segretario
Impiegato	Rappresenta una persona che svolge un qualche ruolo nell'azienda		Codice Fiscale, Persona (Esterno)
Ferroviere	Rappresenta un impiegato che svolge il ruolo di assicurare il corretto movimento del treno	Bisogna controllare se il ferroviere è presente nella lista dei ferrovieri attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)	Codice Fiscale
Lingua	Rappresenta una lingua del mondo		Lingua
Manutenzione	Rappresenta una specifica manutenzione eseguita su un treno e/o carrozza	Il treno su cui si sta facendo la manutenzione, sia presente Controllare che il banco che si sta cercando di utilizzare non sia già occupato	Identificativo, Treno (Esterno), Banco (Esterno)
Modello Treno	Rappresenta un modello di un treno		Nome
Operaio	Rappresenta un Operaio	Controlliamo se l'operaio è presente nella lista degli operai attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)	Codice Fiscale

Percorso	Rappresenta le sezioni di percorso che un treno deve effettuare	Non modificare un percorso se è già associato ad uno spostamento e aggiornare il percorso solo con dati validi	Identificativo, Punto di Interesse (Esterno)
Persona	Rappresenta le generalità di una persona fisica		E-mail
Personale di Bordo	Rappresenta un gruppo di impiegati situati sui treni che solitamente svolgono gli stessi percorsi	Il capotreno deve essere un capotreno e il macchinista deve essere un macchinista.	Identificativo, Capotreno (Esterno), Macchinista (Esterno)
Posto	Rappresenta un posto a sedere presente in una carrozza		Numero, Carrozza (Esterno), Treno (Esterno)
Punto di Interesse	Un luogo di particolare importanza per il treno e/o i passeggeri, come stazioni, depositi, deviatoi ecc		Nome
Segretario	Rappresenta un impiegato che lavora in un ufficio e svolge le operazioni burocratiche di dovere della azienda	Controlliamo se il segretario è presente nella lista dei segretari attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)	Codice Fiscale
Sezione	Rappresenta le informazioni della sezione che un treno percorre in un dato percorso	Ogni volta che si aggiunge una sezione, aggiornare l'ora di arrivo e il punto di interesse di arrivo. Inoltre, non si possono aggiungere sezioni ad un percorso che ha già uno spostamento Inoltre controllare se la sezione è compatibile con la tipologia di percorso specificato Non si possono modificare le sezioni. Nel caso una sezione faccia fermata, controllare che sia in una stazione. Se si passa in una stazione, controllare	Percorso (Esterno), Tratta (Esterno), Ordine

		che il binario sia inserito correttamente Non si può cancellare una sezione che è già associata ad uno spostamento o che abbia valore di ordine minore del massimo valore per quel percorso. Nel caso si possa cancellare la sezione, modificare i dati del percorso.	
Sezione Effettiva	Rappresenta le informazioni reali della sezione che un treno ha percorso in un dato itinerario	Quando si aggiorna il campo uscita effettiva di una sezione bisogna controllare che sia l'ultima dell'itinerario. Nel caso sia l'ultima, controllare che i ferrovieri interessati non abbiano superato l'orario del turno. Nel caso, aggiungere lo straordinario L'orario di entrata deve essere maggiore dell'orario di uscita della sezione precedente (se esiste), l'ora di uscita della sezione corrente deve essere maggiore di quello d'entrata	Identificativo
Spostamento	Rappresenta uno spostamento di un determinato treno all'interno della rete ferroviaria	I ferrovieri devono lavorare durante il loro turno e non devono essere assegnati ad un altro spostamento Gli spostamenti non si devono accavallare su una determinata tratta	Percorso (Esterno), Treno (Esterno), Giorno, Personale di Bordo (Esterno)

		Un treno può partire solo dal punto di arrivo dell'ultimo viaggio Non cancellare gli spostamenti effettuati nel passato o che non siano l'ultimo spostamento programmato per quel treno L'ultima stazione referenziata dal percorso deve essere anche una fermata se la tipologia del percorso è impostata a "Viaggio"	
Stazione	Rappresenta una stazione passeggeri all'interno della rete ferroviaria dove i passeggeri possono scendere o salire		Nome
Tratta	Rappresenta le caratteristiche del tragitto presente tra due punti di interesse		Limite Velocità, Punto di Interesse (Esterno)
Treno	Rappresenta un Treno, inteso come motrice di un certo modello, che traporta un certo numero di carrozze.		Identificativo, Modello Treno (Esterno)
Turno	Rappresenta un turno di lavoro di un dato impiegato	I turni di un impiegato non si devono accavallare Il turno di un impiegato deve ricadere in un periodo in cui lui ha il contratto Non cancellare un turno nel passato o uno nel futuro in cui un ferroviere è impegnato Un turno non deve essere più lungo di 4 ore se la tipologia di contratto dell'impiegato è part-time	Impiegato (Esterno), Giorno, Straordinario

Ufficio	Rappresenta un luogo dove i segretari vanno per svolgere il loro lavoro		Identificativo

Relazioni

Lingua Assistente	Rappresenta le lingue parlate da un assistente		Segretario (Esterno), Lingua (Esterno)
Lingua Segretario	Rappresenta le lingue parlate da un segretario		Segretario (Esterno), Lingua (Esterno)
Riparatori	Rappresenta la partecipazione di un determinato operaio ad una determinata manutenzione.		Manutenzione (Esterno), Operaio (Esterno)
Servizio Assistente	Rappresenta il servizio che fa un assistente nel personale di bordo		Ferroviere (Esterno), Personale di Treno (Esterno)

Tavola dei Volumi

Nome	Tipologia	Volume (indicativo)	Spiegazione
Account	E	250.000 (All'incirca 25 mila in più all'anno)	All'incirca 5 milioni di clienti sul suolo nazionale con un rate di iscrizione pari 1 su 20
Assegnazione Deposito	E	150	Su 75 operai, la metà avrà cambiato deposito, le metà sarà stata licenziata
Assegnazione Ufficio	E	400	Guardare Assegnazione Deposito
Assistente	E	250	125 Assistenti + Licenziati
Assistenza Clienti	E	400.000 (all'anno)	Considerando circa 8 milioni di record tra biglietto, persone e account, e considerando che solo il 5% dei record richiedono assistenza
Banco Manutenzione	E	15	Su 30 treni circolanti è ideale avere almeno la metà in banchi
Biglietto	E	5 milioni (all'anno)	(30 treni circolanti * 3 viaggi per treno al giorno in media * 365 giorni * 40 posti per carrozza in media * 8 carrozze) /2 (un treno è riempito a metà in media)
Binario	E	150	5*30 stazioni
Carrozza	E	420	40 treni * 8 carrozze per treno
Cellulare	E	3 milioni (All'incirca 100.000 in più all'anno)	1 per ogni persona
Classe di Viaggio	E	4	Quattro possibili classi di viaggio
Contenuto Assistenza	E	1 milione (all'anno)	400 mila richieste di assistenza * 2.5 interazioni medie
Contratto	E	450 (Ogni 5 anni)	450 impiegati attivi in contemporanea
Deposito	E	4	10 treni per deposito

Direzione Ufficio	E	5	5 Uffici, ogni ufficio ha il suo capo
Ferroviere	E	100	50 capotreni, 50 macchinisti
Impiegato	E	900	All'incirca: 50 capotreni, 50 macchinisti, 125 assistenti, 25 operai, 200 segretari. Raddoppiare per coprire eventuali licenziamenti
Lingua	E	24	Lingue ufficiali unione europea
Lingue assistente	R	700	4 lingue conosciute in media * 175 ferrovieri
Lingue segretario	R	600	3 lingue conosciute in media * 200
Manutenzione	E	550 (all'anno)	12 manutenzioni all'anno * 40 treni + 1 guasto a settimana* 52 settimane in un anno
Modello Treno	E	3	La nostra azienda utilizza 3 modelli diversi di treni
Operaio	E	25	Abbiamo 25 operai
Percorso	E	150	(30 treni circolanti* 3 percorsi al giorno per treno + 2 per rientrare/uscire dai depositi) + 10 percorsi effettuati solo in alcuni giorni della settimana
Persona	E	3 milioni (All'incirca 100.000 in più all'anno)	Platea dei nostri percorsi. La nostra espansione dei percorsi nel tempo potrebbe portare all'incirca 100mila passeggeri in più all'anno
Personale di Bordo	E	150	30 treni circolanti* 5 percorsi al giorno per treno + 10 percorsi effettuati solo in alcuni giorni della settimana
Posto	E	16.000	50 posti * 8 carrozze * 40 treni
Punto di Interesse	E	5000	Mediamente su un percorso: 1 semaforo ogni 1KM * 800 KM di tracciato + 0.1 deviatoio ogni 1 KM * 800 KM + 30 stazioni servite + 3 Depositi
Riparatori	R	4000 (All'anno)	400 manutenzioni all'anno * 10 operai per manutenzione
Segretario	E		
Servizio Assistente	R	500	125 assistenti * 3 viaggi diversi al giorno + eventuali
Sezione	E	5000	500 tratte per percorso in media * 100 percorsi
Sezione Effettiva	E	20 milioni (all'anno)	In media una sezione viene attraversata una decina di volte al giorno * 365 giorni * 5000 sezioni
Spostamento	E	55 mila (all'anno)	30 treni * 5 viaggi per treno in media * 365 giorni
Stazione	E	20	Nella nostra realtà abbiamo all'incirca 20 stazioni AV
Tratta	E	5000	All'incirca come i punti di interesse
Treno	E	40	Flotta iniziale. 30 treni circolanti + 10 sostitutivi. Potrebbe aumentare nel caso l'azienda si espanda
Turno	E	350.000	1 turno al giorno in media (considerando anche i possibili straordinari) * 900 impiegati * 365 giorni
Ufficio	E	5	5 uffici, uno per ogni tipologia

Tavola degli Accessi **(Delle 5 operazioni principali)**

Visualizzazione Viaggi

Concetto	Costrutto	Accessi	Tipo
Spostamento	Entità	1	Lettura
Punto di Interesse	Entità	2	Lettura
Tratta	Entità	1	Lettura
Classe Viaggio	Entità	1	Lettura
Percorso	Entità	3	Lettura
Sezione	Entità	4	Lettura

Calcolo Retribuzione di un Impiegato

Concetto	Costrutto	Accessi	Tipo
Turno	Entità	1	Lettura
Contratto	Entità	1	Lettura

Acquisto Biglietto

Concetto	Costrutto	Accessi	Tipo
Spostamento	Entità	1	Lettura
Carrozza	Entità	1	Lettura
Posto	Entità	2	Lettura
Biglietto	Entità	2	Lettura
Biglietto	Entità	1	Scrittura
Account	Entità	1	Lettura
Tratta	Entità	1	Lettura
Sezione	Entità	3	Lettura
Biglietto	Entità	1	Scrittura

Inserimento Personale di Bordo

Concetto	Costrutto	Accessi	Tipo
Personale di Bordo	Entità	1	Lettura
Personale di Bordo	Entità	1	Scrittura
Servizio Assistente	Relazione	N con N uguale agli assistenti che fanno parte del personale di bordo	Scrittura

Visualizzazione Arrivi Stazione

Concetto	Costrutto	Accessi	Tipo
Treno	Entità	1	Lettura
Percorso	Entità	1	Lettura
Spostamento	Entità	1	Lettura
Sezione	Entità	1	Lettura

TERZA PARTE – TRADUZIONE DIRETTA AL MODELLO RELAZIONALE

Leggenda: Attributo; Chiave, **Chiave Esterna**, *Attributo Nullo

- Account (Saldo Punti, Codice Fedeltà, Livello, Account Password**, E-mail**)

- Assegnazione Deposito (**Deposito, Operaio**, Inizio, *Fine)

- Assegnazione Ufficio (**Ufficio Segretario**, Inizio *Fine,)

- Assistente (**Impiegato**)

- Assistenza Clienti (ID, E-Mail, Oggetto, **Lingua, Account*, Biglietto*,** Oggetto, Chiusa)

- Banco_Manutenzione(Identificativo, **Deposito**)

- Biglietto (ID, Prezzo, Rimborsato, Data_Acquisto**,** MetodoPagamento, Id_Pagamento, ***Account, Spostamento_Percorso, Posto, Carrozza, Treno, E-mail, Data_Viaggio, Partenza_Ordine, Arrivo_Ordine**)

- Binario (**Stazione**, Numero)

- Carrozza (Numero, IDTreno, **ClasseViaggio**)

- Cellulare (**E-mail,** Numero)

- Classe_Viaggio(Nome, prezzo)

- Contenuto Assistenza (**Assistenza**, Timestamp Richiesta, testo, *testo risposta, *timestamp risposta, ***segretario**)

- Contratto (ID, Tipo, Inizio, Rescisso, Fine, Paga_Oraria, **Impiegato**)

- Deposito (**Nome**, Via, Civico, Città, Cap)

- Direzione Ufficio (**Ufficio, Segretario**, Inizio, *Fine)

- Ferroviere (**Impiegato**, Ruolo)

- Impiegato (CF, **E-mail,** E-mail Aziendale, Data di Nascita, Sesso, Iban, Tipologia)

- Lingua (Lingua)

- Lingue Assistente (**Ferroviere, Lingua**)

- Lingue Segretario (**Segretario, Lingua**)

- Manutenzione (ID, Descrizione, *Fine, Inizio, **IDtreno, Banco, ***Descrizione Guasto)

- Modello Treno (Nome, Velocità Max)

- Operaio (**Impiegato**)

- Percorso (Identificativo, Giorno, Tipologia, Ora Arrivo, Ora Partenza, **Partenza, Arrivo**)

- Persona (Nome, Cognome, E-mail, *Cellulare)

- Personale di Bordo (**ID, Capotreno, Macchinista**)

- Posto (Numero, **Carrozza, IDtreno,** PDA)

- Punto Interesse (Nome, Coordinate)

- Riparatori (**Manutenzione**, **Operaio**)

- Segretario (**Impiegato**, Account Password)

- Servizio Assistente (**Assistente, IDgruppo**)

- Sezione (**Percorso,** Ordine, **Tratta_Limit, Tratta_Inizio, Tratta_Fine**, ***Binario**, Entrata Programmata, Uscita Programmata, Sosta Programmata, Sosta?)

- Sezione Effettiva (Identificativo, **Percorso, Sezione_Ordine, Giorno,** Entrata, Uscita, **Binario, Stazione**)

- Spostamento (**Percorso,** Giorno, **Treno, IDgruppo**)

- Stazione (**Nome**, Via, Città, Civico, Cap, NumeroBinari, ADA)

- Tratta (Limite_Velocità, **Inizio, Fine**, Lunghezza, Numero_Binari)

- Treno (Identificativo, **Modello_Treno**)

- Turno (**Impiegato**, Giorno, Inizio, Fine, Straordinario)

- Ufficio (Identificativo, Via, Città, Civico, Cap, Tipologia, Posti, **Dirigente**)

QUARTA PARTE – RISTRUTTURAZIONE DELLO SCHEMA RELAZIONALE TENENDO CONTO DEL CARICO DELL'APPLICAZIONE

Ristrutturazione dello schema ER tenendo conto del carico dell'applicazione

Dopo una attenta analisi dello schema, non siamo riusciti a trovare modi per ridurre i carichi dello schema.

La parte relativa alla gestione dei dipendenti è molto efficiente e non crediamo di poter trovare criticità in quella sezione dello schema, essendo le entità molto ben distribuite.

Anche per quanto riguarda la sezione "centrale" dello schema, ovvero quella della composizione di un treno, siamo convinti di questo, in quanto le entità sono distribuite e non abbiamo relazioni che accorpano una grande quantità di attributi.
Concentrandoci sulle rimanenti relazioni più corpulente, ovvero coloro quelle che comportano operazioni con costi molto elevati :
- L'entità percorso ci aiuta a non avere delle ripetizioni per Spostamenti che vengono effettuati tutti i giorni alla stessa ora e che si fermano alle stesse fermate alla stessa ora, risparmiando così non solo tempo, ma anche spazio sulla nostra base di dati. (e tempo per l'inserimento delle tratte nella base di dati, il che non è da sottovalutare!)
- Sezione è forse l'entità più problematica, visto che è l'incaricata di salvare tutte le informazioni sulle tratte e le tempistiche del viaggio, ma nonostante questo riusciamo a fare tutto il necessario con un solo accesso durante le operazioni.
Concludendo quindi, siamo abbastanza soddisfatti delle prestazioni del nostro Database nonostante il carico importante della realtà che si vuole rappresentare

QUINTA PARTE – SPECIFICA DEL DATABASE IN SQL

Schema Logico

Leggenda: Attributo; Chiave, **Chiave Esterna**, *null

Account (Saldo Punti, Codice Fedeltà, Livello, Account Password, **E-mail**)

```
CREATE TABLE Account (
    Saldo_punti INTEGER NOT NULL DEFAULT 0,
    Codice_fedeltà INTEGER NOT NULL PRIMARY KEY,
    Livello          LIVELLI_ACCOUNT DEFAULT 'Standard' NOT NULL,
    Account_password VARCHAR(30) NOT NULL,
    Email            EMAIL  NOT NULL,
    CONSTRAINT fk_account_persona
        REFERENCES persona
);
```

Assegnazione Deposito (**Deposito, Operaio**, Inizio, *Fine)

```
CREATE TABLE Assegnazione_Deposito (
  Deposito VARCHAR(30) NOT NULL,
  Operaio CODICE_FISCALE NOT NULL,
  Inizio TIMESTAMP NOT NULL,
  Fine TIMESTAMP,
  CONSTRAINT fk_luogodilavoro_nome
    FOREIGN KEY(deposito)
      REFERENCES deposito(nome)
      ON UPDATE CASCADE,
  CONSTRAINT fk_luogodilavoro_cf
    FOREIGN KEY(operaio)
      REFERENCES operaio(impiegato)
      ON UPDATE CASCADE,
  PRIMARY KEY (deposito, operaio, inizio)
  CONSTRAINT assegnazione_deposito_check
      CHECK (fine > inizio)

);
```

Assegnazione Ufficio (**Ufficio Segretario**, Inizio *Fine,)

```
CREATE TABLE Assegnazione_Ufficio (
    Segretario codice_fiscale NOT NULL
        CONSTRAINT fk_locazione_cf
            REFERENCES segretario
            ON UPDATE CASCADE,
    Fine       DATE,
    Inizio     DATE NOT NULL,
    Ufficio    INTEGER NOT NULL
    CONSTRAINT fk_assegnazione_ufficio_ufficio
        REFERENCES ufficio,
    PRIMARY KEY (segretario, ufficio, inizio)
    CONSTRAINT assegnazione_ufficio_check
        CHECK (fine > inizio)
);
```

Assistente (**Impiegato**)

```
CREATE TABLE Assistente (
    Impiegato codice_fiscale PRIMARY KEY
        CONSTRAINT fk_assistente_impiegato
            REFERENCES impiegato
);
```

UniTrain: Relazione Finale

Assistenza Clienti (ID, E-Mail, Oggetto, **Lingua, Account*, Biglietto*,** Oggetto, Chiusa)

```
CREATE TABLE Assistenza_Clienti (
    Identificativo SERIAL PRIMARY KEY NOT NULL
    Email          email NOT NULL,
    Lingua         VARCHAR(20) NOT NULL
        CONSTRAINT fk_assistenza_lingua
            REFERENCES lingua
            ON UPDATE CASCADE,
    Account        INTEGER
        CONSTRAINT fk_account
            REFERENCES account,
    Biglietto      INTEGER
        CONSTRAINT fk_assistenza_biglietto
            REFERENCES biglietto,
    Oggetto        VARCHAR(100) NOT NULL,
    Chiusa         BOOLEAN DEFAULT FALSE NOT NULL
);
```

Banco Manutenzione (Identificativo, **Deposito**)

```
CREATE TABLE Banco_Manutenzione (
    Identificativo SERIAL PRIMARY KEY,
    Deposito VARCHAR(30) NOT NULL,
    CONSTRAINT fk_banco_nome
         REFERENCES deposito
         ON UPDATE CASCADE
);
```

Biglietto (ID, Prezzo, Rimborsato, Data_Acquisto**,** MetodoPagamento, Id_Pagamento, ***Account, Spostamento_Percorso, Posto, Carrozza, Treno, E-mail, Data_Viaggio, Partenza_Ordine, Arrivo_Ordine**)

```
CREATE TABLE Biglietto (
    Identificativo SERIAL DEFAULT NOT NULL PRIMARY KEY,
    Prezzo NUMERIC(6, 2) DEFAULT 0.00 NOT NULL,
    Rimborsato BOOLEAN DEFAULT FALSE NOT NULL,
    Metodopagamento metodipagamento NOT NULL,
    Id_pagamento VARCHAR(100) NOT NULL,
    Account INTEGER
        CONSTRAINT fk_biglietto_account
            REFERENCES account,
    Spostamento_percorso INTEGER NOT NULL,
    Posto SMALLINT NOT NULL,
    Carrozza SMALLINT NOT NULL,
    Email email NOT NULL
        CONSTRAINT fk_biglietto_persona
            REFERENCES persona,
    Treno SMALLINT,
    Data_acquisto TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
    Data_viaggio DATE NOT NULL,
    Partenza_ordine INTEGER NOT NULL,
    Arrivo_ordine INTEGER NOT NULL,
    CONSTRAINT fk_biglietto_posto
        FOREIGN KEY (carrozza, treno, posto) REFERENCES posto (carrozza,
idtreno, numero),
    CONSTRAINT fk_biglietto_spostamento
        FOREIGN KEY (spostamento_percorso, data_viaggio) REFERENCES spostamento,
    CONSTRAINT fk_biglietto_sezione_arrivo
        FOREIGN KEY (arrivo_ordine, spostamento_percorso) REFERENCES sezione
(ordine, percorso),
    CONSTRAINT fk_biglietto_sezione_partenza
        FOREIGN KEY (spostamento_percorso, partenza_ordine) REFERENCES sezione
);
```

Binario (**Stazione**, Numero)

```
CREATE TABLE Binario (
    Stazione VARCHAR(100) NOT NULL
        CONSTRAINT binario_stazione_nome_fk
            REFERENCES stazione,
    Numero   VARCHAR(10)  NOT NULL,
    CONSTRAINT binario_pk
        PRIMARY KEY (stazione, numero)
);
```

Carrozza (Numero, IDTreno, **ClasseViaggio**)

```
CREATE TABLE Carrozza (
    Numero  SMALLINT NOT NULL,
    Idtreno SMALLINT NOT NULL
        CONSTRAINT fk_carrozza_idtreno
            REFERENCES treno
            ON UPDATE CASCADE,
    Classe_viaggio VARCHAR(30) NOT NULL
        CONSTRAINT fk_carrozza_classeviaggio
            REFERENCES classe_viaggio
            ON UPDATE CASCADE,
    PRIMARY KEY (numero, idtreno)
);
```

Cellulare (numero_telefonico, **E-mail**)

```
CREATE TABLE Cellulare (
    Email             email       NOT NULL
        CONSTRAINT fk_cellulare_email
            REFERENCES persona
            ON UPDATE CASCADE,
    Numero_Telefonico VARCHAR(10) NOT NULL,
    PRIMARY KEY (email, numero_telefonico)
);
```

Classe Viaggio (Nome, prezzo)

```
CREATE TABLE Classe_Viaggio (
    Nome            VARCHAR(30) NOT NULL,
    Prezzo          NUMERIC(6, 2) NOT NULL,
    PRIMARY KEY (nome)
);
```

Contenuto Assistenza (**Assistenza**, Timestamp Richiesta, Testo, *Testo risposta, *Timestamp risposta, ***Segretario**)

```
CREATE TABLE Contenuto_Assistenza (
    Assistenza  INTEGER NOT NULL
        CONSTRAINT fk_contenuto_assistenza_assistenza
            REFERENCES assistenza_clienti,
    Testo               TEXT      NOT NULL,
    Timestamp_richiesta TIMESTAMP NOT NULL,
    Testo_risposta      TEXT,
    Timestamp_risposta  TIMESTAMP,
    Segretario          codice_fiscale
        CONSTRAINT fk_contenuto_assistenza_segretario
            REFERENCES segretario,
    CONSTRAINT contenuto_assistenza_key
        PRIMARY KEY (assistenza, timestamp_richiesta)
);
```

Contratto (ID, Tipo, Inizio, Rescisso, Fine, Paga_Oraria, **Impiegato**)

```
CREATE TABLE Contratto (
    Identificativo SERIAL
        PRIMARY KEY,
    Tipo           tipo_contratto               NOT NULL,
    Inizio         DATE    DEFAULT (NOW())::DATE NOT NULL,
    Rescisso       BOOLEAN DEFAULT FALSE         NOT NULL,
    Paga_oraria    NUMERIC(5, 2)                 NOT NULL,
    Fine           DATE                          NOT NULL,
    Impiegato      codice_fiscale                NOT NULL
        CONSTRAINT contratto_impiegato_cf_fk
            REFERENCES impiegato,
    CONSTRAINT contratto_check
        CHECK (fine > inizio)
);
```

Deposito (**Nome**, Via, Civico, Città, Cap)

```
CREATE TABLE Deposito (
  Nome VARCHAR(30) PRIMARY KEY,
  Via VARCHAR(30) NOT NULL,
  Città VARCHAR(30) NOT NULL,
  Civico INTEGER NOT NULL,
  Cap INTEGER NOT NULL,
  CONSTRAINT fk_deposito_nome
    FOREIGN KEY(Nome)
      REFERENCES PuntoDiSosta(Nome)
      ON UPDATE CASCADE
);
```

Direzione Ufficio (**Ufficio, Segretario**, Inizio, *Fine)

```
CREATE TABLE Direzione_Ufficio (
    Ufficio    INTEGER        NOT NULL
        CONSTRAINT fk_direzione_ufficio_ufficio
            REFERENCES ufficio,
    Segretario codice_fiscale NOT NULL
        CONSTRAINT fk_direzione_ufficio_segretario
            REFERENCES segretario,
    Inizio     TIMESTAMP      NOT NULL,
    Fine       TIMESTAMP,
    CONSTRAINT direzione_ufficio_pk
        PRIMARY KEY (ufficio, segretario, inizio)
    CONSTRAINT direzione_ufficio_check
        CHECK (fine > inizio)
);
```

Ferroviere (**Impiegato**, Ruolo)

```
CREATE TABLE Ferroviere (
  Impiegato CODICE_FISCALE PRIMARY KEY,
  Ruolo RUOLO_FERROVIERE NOT NULL,
  CONSTRAINT fk_ferroviere_cf
    FOREIGN KEY(Impiegato)
      REFERENCES Impiegato(CF)
      ON UPDATE CASCADE
);
```

Impiegato (CF, N**ome, Cognome, Email,** Email Aziendale, Data di Nascita, Sesso, Iban, Tipologia)

```
CREATE TABLE Impiegato (
    CF CODICE_FISCALE  NOT NULL PRIMARY KEY,
    Nome            VARCHAR(100) NOT NULl,
    Cognome         VARCHAR(100) NOT NULL,
    Sesso           VARCHAR(100) NOT NULL,
    Email           EMAIL NOT NULL,
    Email_aziendale EMAIL  NOT NULL,
    Data_nascita    DATE NOT NULL,
    Iban            IBAN NOT NULL,
    Tipologia       tipo_impiegato NOT NULL
    CONSTRAINT fk_impiegato_persona
        FOREIGN KEY (Nome, Cognome, Email) REFERENCES Persona
);
```

Lingue Assistente (**Ferroviere, Lingua**)

```
CREATE TABLE Lingue_Ferroviere (
  Assistente CODICE_FISCALE,
  Lingua VARCHAR(20),
  CONSTRAINT fk_comunicare_cf
    FOREIGN KEY(Assistente)
      REFERENCES Assistente(Impiegato)
      ON UPDATE CASCADE,
  CONSTRAINT fk_comunicare_lingua
    FOREIGN KEY(Lingua)
      REFERENCES Lingua(Lingua)
      ON UPDATE CASCADE,
  PRIMARY KEY (Ferroviere, Lingua)
);
```

Lingua (**Lingua**)

```
CREATE TABLE Lingua (
  Lingua VARCHAR(20) NOT NULL PRIMARY KEY
);
```

Lingue Segretario (**Segretario, Lingua**)

```
CREATE TABLE Lingue_Segretario (
  Segretario CODICE_FISCALE,
  Lingua VARCHAR(20),
  CONSTRAINT fk_scrivere_cf
    FOREIGN KEY(Segretario)
      REFERENCES Segretario(impiegato)
      ON UPDATE CASCADE,
  CONSTRAINT fk_scrivere_lingua
    FOREIGN KEY(Lingua)
      REFERENCES Lingua(Lingua)
      ON UPDATE CASCADE,
  PRIMARY KEY (Segretario, Lingua)
);
```

Manutenzione (ID, Descrizione, *Fine, Inizio, **IDtreno, Banco, ***Descrizione Guasto)

```
CREATE TABLE Manutenzione (
    Identificativo SERIAL NOT NULL PRIMARY KEY,
    Descrizione    TEXT NOT NULL,
    Idtreno        INTEGER NOT NULL,
    Fine           TIMESTAMP,
    Inizio         TIMESTAMP NOT NULL,
    Banco          INTEGER NOT NULL,
    Descrizione_guasto    TEXT,
    CONSTRAINT fk_manutenzione_idtreno
```

```
        FOREIGN KEY(idtreno)
            REFERENCES treno
            ON UPDATE CASCADE,
    CONSTRAINT fk_manutenzione_banco
        FOREIGN KEY(banco)
            REFERENCES banco_riparazioni
            ON UPDATE CASCADE
    CONSTRAINT manutenzione_check
        CHECK (fine > inizio));
```

Modello Treno (Nome, Velocità Max)

```
CREATE TABLE Modello_Treno (
  Nome VARCHAR(30) NOT NULL PRIMARY KEY,
  Velocita_Max SMALLINT NOT NULL
);
```

Operaio (**Impiegato**)

```
CREATE TABLE Operaio (
  Impiegato CODICE_FISCALE PRIMARY KEY,
  CONSTRAINT fk_operaio_cf
    FOREIGN KEY(Impiegato)
      REFERENCES Impiegato(CF)
      ON UPDATE CASCADE
);
```

Percorso (Identificativo, Ora Arrivo, Ora Partenza, **Partenza, Arrivo**, Tipologia)

```
CREATE TABLE Percorso (
    Identificativo SERIAL NOT NULL PRIMARY KEY,
    Ora_arrivo TIME NOT NULL,
    Ora_partenza TIME NOT NULL,
    Partenza VARCHAR(100) NOT NULL,
    Arrivo VARCHAR(100) NOT NULL,
    Tipologia      tipo_spostamento DEFAULT 'Viaggio'::tipo_spostamento NOT NULL
    CONSTRAINT fk_percorso_partenza
        FOREIGN KEY (partenza)
            REFERENCES punto_interesse,
    CONSTRAINT fk_percorso_arrivo
        FOREIGN KEY (arrivo)
            REFERENCES punto_interesse
    CONSTRAINT percorso_check
        CHECK (ora_arrivo > ora_partenza)
)
```

Persona (Nome, Cognome, E-mail, *Cellulare)

```
CREATE TABLE Persona (
    Nome      VARCHAR(100) NOT NULL,
    Cognome   VARCHAR(100) NOT NULL,
    Email     email PRIMARY KEY NOT NULL,
    Cellulare VARCHAR(10),
);
```

Personale di Bordo (**ID, Capotreno, Macchinista**)

```
CREATE TABLE Personale_Bordo (
    Identificativo SERIAL NOT NULL
        CONSTRAINT gruppo_ferrovieri_pkey
            PRIMARY KEY,
    Capotreno      codice_fiscale NOT NULL
        CONSTRAINT fk_gruppo_ferrovieri_capotreno
            REFERENCES ferroviere,
    Macchinista    codice_fiscale NOT NULL
        CONSTRAINT fk_gruppo_ferrovieri_macchinista
```

```
            REFERENCES ferroviere
);
```

Posto (Numero, **Carrozza, IDtreno,** PDA)

```
CREATE TABLE Posto (
    Numero   SMALLINT NOT NULL,
    Carrozza SMALLINT NOT NULL,
    Pda      BOOLEAN,
    Idtreno  SMALLINT NOT NULL,
    PRIMARY KEY (numero, carrozza, idtreno),
    CONSTRAINT fk_posto_treno
        FOREIGN KEY (carrozza, idtreno) REFERENCES carrozza
            ON UPDATE CASCADE
);
```

Punto Interesse (Nome, Coordinate)

```
CREATE TABLE Punto_Interesse (
  Nome VARCHAR(100) NOT NULL PRIMARY KEY,
  Coordinate VARCHAR(40) NOT NULL
);
```

Riparatori (**Manutenzione**, **Operaio**)

```
CREATE TABLE Riparatori (
  Manutenzione INT,
  Operaio CODICE_FISCALE,
  CONSTRAINT fk_riparatori_id
    FOREIGN KEY(Manutenzione)
      REFERENCES Manutenzione(identificativo)
      ON UPDATE CASCADE,
  CONSTRAINT fk_riparatori_cf
    FOREIGN KEY(Operaio)
      REFERENCES Operaio(impiegato)
      ON UPDATE CASCADE,
  PRIMARY KEY (Manutenzione, Operaio)
);
```

Segretario (**Impiegato**, Account Password)

```
CREATE TABLE Segretario (
  Impiegato CODICE_FISCALE PRIMARY KEY,
  Account_password VARCHAR(30) NOT NULL,
  CONSTRAINT fk_segretario_cf
    FOREIGN KEY(Impiegato)
      REFERENCES Impiegato(CF)
      ON UPDATE CASCADE
);
```

Servizio Assistente (**Assistente, IDgruppo**)

```
CREATE TABLE Servizio_Assistente (
    Assistente codice_fiscale NOT NULL
        CONSTRAINT fk_servizio_assistente_assistente
            REFERENCES assistente
            ON UPDATE CASCADE,
    Idgruppo   INTEGER        NOT NULL
        CONSTRAINT fk_servizio_assistente_idgruppo
            REFERENCES personale_bordo
            ON UPDATE CASCADE,
    PRIMARY KEY (assistente, idgruppo)
);
```

Sezione (**Percorso,** Ordine, **Tratta_Limit, Tratta_Inizio, Tratta_Fine**, ***Binario**, Entrata Programmata, Uscita Programmata, Sosta Programmata, Sosta?)

```
CREATE TABLE Sezione (
    Percorso            INTEGER                 NOT NULL
        CONSTRAINT fk_sezione_percorso
            REFERENCES percorso,
    Tratta_limit        SMALLINT                NOT NULL,
    Tratta_inizio       VARCHAR(100)            NOT NULL,
    Tratta_fine         VARCHAR(100)            NOT NULL,
    Ordine              SMALLINT DEFAULT 0      NOT NULL,
    Entrata_programmata TIME                    NOT NULL,
    Uscita_programmata  TIME                    NOT NULL,
    "fermata?"          BOOLEAN  DEFAULT FALSE NOT NULL,
    Sosta               INTEGER  DEFAULT 0      NOT NULL,
    Binario             VARCHAR(10),
    PRIMARY KEY (percorso, ordine),
    CONSTRAINT fk_sezione_tratta
        FOREIGN KEY (tratta_inizio, tratta_limit, tratta_fine) REFERENCES tratta
(inizio, limite_velocita, fine)
            ON UPDATE CASCADE,
    CONSTRAINT sezione_binario_stazione_numero_fk
        FOREIGN KEY (tratta_fine, binario) REFERENCES binario,
    CONSTRAINT sezione_check
        CHECK (uscita_programmata > entrata_programmata)
);
```

Sezione Effettiva (Identificativo, **Percorso, Sezione_Ordine, Giorno,** Entrata, Uscita, **Binario, Stazione**)

```
CREATE TABLE Sezione_Effettiva (
    Uscita         TIME,
    Entrata        TIME     NOT NULL,
    Sezione_ordine SMALLINT NOT NULL,
    Percorso       INTEGER  NOT NULL,
    Giorno         DATE     NOT NULL,
    Identificativo BIGSERIAL
        CONSTRAINT sezione_effettiva_pk
            PRIMARY KEY,
    Binario        VARCHAR(10),
    Stazione       VARCHAR(30),
    CONSTRAINT sezione_effettiva_spostamento_fk
        FOREIGN KEY (percorso, giorno) REFERENCES spostamento,
    CONSTRAINT sezione_effettiva_sezione_ordine_percorso_fk
        FOREIGN KEY (sezione_ordine, percorso) REFERENCES sezione (ordine,
percorso),
    CONSTRAINT sezione_effettiva_binario_numero_stazione_fk
        FOREIGN KEY (binario, stazione) REFERENCES binario (numero, stazione),
    CONSTRAINT sezione_effettiva_check
        CHECK (uscita > entrata)
);
```

Spostamento (**Percorso,** Giorno, **Treno, IDgruppo**, Tipologia)

```
CREATE TABLE Spostamento (
    Percorso INTEGER  NOT NULL
        CONSTRAINT fk_spostamento_percorso
            REFERENCES percorso,
    Treno    SMALLINT NOT NULL
        CONSTRAINT fk_itinerario_treno
            REFERENCES treno
            ON UPDATE CASCADE,
    Giorno   DATE     NOT NULL
    CONSTRAINT spostamento_giorno_check
        CHECK (giorno > CURRENT_DATE),
```

```
    Idgruppo INTEGER  NOT NULL
        CONSTRAINT fk_spostamento_gruppo_ferrovieri
            REFERENCES personale_bordo
            ON UPDATE CASCADE,
    CONSTRAINT itinerario_pkey
        PRIMARY KEY (percorso, giorno)
);
```

Stazione (**Nome**, Via, Città, Civico, Cap, NumeroBinari, ADA)

```
CREATE TABLE Stazione (
  Nome VARCHAR(100) PRIMARY KEY ,
  Via VARCHAR(50) NOT NULL,
  Città VARCHAR(30) NOT NULL,
  Civico INTEGER NOT NULL,
  Cap INTEGER NOT NULL,
  NumeroBinari SMALLINT NOT NULL,
  ADA BOOLEAN NOT NULL,
  CONSTRAINT fk_stazione_nome
    FOREIGN KEY(Nome)
      REFERENCES Punto_Interesse(Nome)
      ON UPDATE CASCADE
);
```

Tratta (Limite_Velocità, **Inizio, Fine**, Lunghezza, Numero_Binari)

```
CREATE TABLE Tratta (
    Limite_velocità SMALLINT     NOT NULL,
    Inizio          VARCHAR(100) NOT NULL
        CONSTRAINT fk_tratta_inizio
            REFERENCES punto_interesse
            ON UPDATE CASCADE,
    Fine            VARCHAR(100) NOT NULL
        CONSTRAINT fk_tratta_fine
            REFERENCES punto_interesse
            ON UPDATE CASCADE,
    Lunghezza       INTEGER      NOT NULL,
    PRIMARY KEY (limite_velocità, inizio, fine)
);
```

Treno (Identificativo, **Modello_Treno**)

```
CREATE TABLE Treno (
    Identificativo SMALLINT    NOT NULL
        PRIMARY KEY,
    Modello_treno  VARCHAR(30) NOT NULL
        CONSTRAINT fk_treno_nome
            REFERENCES modello_treno
            ON UPDATE CASCADE
);
```

Turno (**Impiegato**, Giorno, Inizio, Fine, Straordinario)

```
CREATE TABLE Turno (
    Impiegato codice_fiscale NOT NULL
        CONSTRAINT fk_turno_cf
            REFERENCES impiegato
            ON UPDATE CASCADE,
    Giorno        DATE NOT NULL,
    Inizio        TIME NOT NULL,
    Fine          TIME NOT NULL,
    Straordinario BOOLEAN DEFAULT FALSE NOT NULL,
    PRIMARY KEY (impiegato, giorno, straordinario)
    CONSTRAINT turno_check
        CHECK (inizio < fine)
);
```

Ufficio (Identificativo, Via, Città, Civico, Cap, Tipologia, Posti, **Dirigente**)

```
CREATE TABLE Ufficio (
    Via VARCHAR(50) NOT NULL,
    Tipologia tipo_ufficio NOT NULL,
    Posti INTEGER NOT NULL,
    Città VARCHAR(30) NOT NULL,
    Civico INTEGER NOT NULL,
    Cap INTEGER NOT NULL,
    Identificativo SERIAL NOT NULL PRIMARY KEY
);
```

Views

Assistenti_Attuali

```
CREATE VIEW assistenti_attuali AS
    SELECT *
    FROM assistente, impiegato, contratto
    WHERE assistente.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Assistente'
    AND contratto.impiegato = impiegato.cf
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
    AND contratto.rescisso = false;
```

Capotreni_Attuali

```
CREATE VIEW capotreni_attuali AS
    SELECT *
    FROM ferroviere, impiegato, contratto
    WHERE ferroviere.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Ferroviere'
    AND ferroviere.ruolo = 'Capotreno'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Macchinisti_Attuali

```
CREATE VIEW macchinisti_attuali AS
    SELECT *
    FROM ferroviere, impiegato, contratto
    WHERE ferroviere.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Ferroviere'
    AND ferroviere.ruolo = 'Macchinista'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Ferrovieri_Attuali

```
CREATE VIEW ferrovieri_attuali AS
    SELECT *
    FROM ferroviere, impiegato, contratto
    WHERE ferroviere.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Ferroviere'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Operai_Attuali

```
CREATE VIEW operai_attuali AS
    SELECT *
    FROM operaio, impiegato, contratto
    WHERE operaio.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Operaio'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Segretari_Atttuali

```
CREATE VIEW segretari_attuali AS
    SELECT *
    FROM segretario, impiegato, contratto
    WHERE segretario.impiegato = impiegato.cf
    AND impiegato.tipologia = 'Segretario'
    AND contratto.impiegato = impiegato.cf
    AND contratto.rescisso = false;
    AND contratto.fine >= current_date
    AND contratto.inizio <= current_date;
```

Types

```
CREATE TYPE tipo_ufficio AS ENUM ('Dirigenza', 'Gestione Percorsi', 'Supporto
Clienti', 'Gestione Turni', 'Risorse Umane', 'Tecnici' );
CREATE TYPE ruolo_ferroviere AS ENUM ('Capotreno', 'Macchinista');
CREATE TYPE tipo_spostamento AS ENUM ('Viaggio', 'Spostamento')
CREATE TYPE livelli_account AS ENUM  ('Standard', 'Silver', 'Gold', 'Platinum')
CREATE TYPE tipo_assistenza AS ENUM ('Generale', 'Account', 'Biglietto')
CREATE TYPE tipo_contratto AS ENUM ('Full_Time', 'Part_Time');
CREATE TYPE tipo_impiegato AS ENUM ('Ferroviere', 'Assistente', 'Segretario',
'Operaio')
```

Domains

```
CREATE DOMAIN email AS VARCHAR(100) check ( value ~ '^\w+@[a-zA-Z_]+?\.[a-zA-
Z]{2,3}$' );
CREATE DOMAIN codice_fiscale AS VARCHAR(16) check ( length(value) = 16 );
CREATE DOMAIN iban AS VARCHAR(27) check ( length(value) = 27 );
```

Trigger

ASSEGNAZIONE_DEPOSITO

-Controllare che un operaio non sia assegnato a due depositi diversi contemporaneamente

```
CREATE FUNCTION insert_assegnazione_deposito() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
    BEGIN
    --Controlliamo se esiste già un record senza fine impostata in cui risulta
il nostro operaio
        SELECT count(*) INTO valore
        FROM assegnazione_deposito
        where assegnazione_deposito.operaio = new.operaio AND fine IS NULL;
    --Se non esiste, prendiamo il timestamp corrente e restituiamo il record
    IF (valore = 0) THEN
        new.inizio = now()::TIMESTAMP;
        RETURN (NEW);
    --Altrimenti segnalarlo
    ELSE
        RAISE EXCEPTION 'Doppia_Assegnazione' USING HINT = 'Questo impiegato è
già stato assegnato ad un deposito, ' ||
                                                        'per favore prima
rimuoverlo dal deposito e dopo inserirlo in quello nuovo';
    END IF;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER insert_assegnazione_deposito
    BEFORE INSERT ON assegnazione_deposito
    FOR EACH ROW
    EXECUTE PROCEDURE insert_assegnazione_deposito()
```

ASSEGNAZIONE_UFFICIO

-Controllare che un segretario non sia assegnato a due uffici diversi contemporaneamente

```
CREATE FUNCTION insert_assegnazione_ufficio() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
    BEGIN
        --Prima controlliamo se l'impiegato è un impiegato attuale (e che quindi
abbia un contratto valido e/o altre mansioni)
        SELECT count(*) INTO valore
        FROM segretari_attuali
        WHERE new.segretario = segretari_attuali.cf;

        IF valore = 0 THEN
            RAISE EXCEPTION 'L''impiegato definito non è un segretario';
        END IF;

        --Se lo è, controllare che non sia assegnato ad altri uffici nel mentre
        SELECT count(*) INTO valore
        FROM assegnazione_ufficio
        where assegnazione_ufficio.segretario = new.segretario AND fine IS NULL;
    IF (valore = 0) then
        new.inizio = now()::timestamp;
```

```
        return (NEW);
    ELSE
         RAISE EXCEPTION 'Doppia_Assegnazione' USING HINT = 'Questo segretario è
già stato assegnato ad un ufficio, ' ||
                                                        'per favore prima
rimuoverlo dal suo vecchio ufficio e dopo inserirlo in quello nuovo';
    END IF;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER assegnazione_ufficio_insert
    BEFORE INSERT ON assegnazione_ufficio
    FOR EACH ROW
    EXECUTE PROCEDURE insert_assegnazione_ufficio()
```

-Se l'impiegato è un dirigente nell'ufficio nella quale se ne sta andando, impedire l'aggiunta del timestamp rappresentante la fine di assegnazione in quell'ufficio

```
CREATE FUNCTION assegnazione_ufficio_update() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
    BEGIN
    --Se ad essere diversa è solamente la data di fine (data che in precedenza
non era stata impostata)
    IF old.fine IS NOT NULL THEN
        IF new.fine != old.fine AND old.segretario = new.segretario
               AND old.ufficio = new.ufficio AND old.inizio = new.inizio THEN
            --Allora controllare che non sia un dirigente in questo momento
            SELECT count(*) INTO valore
            FROM direzione_ufficio
            WHERE direzione_ufficio.ufficio = old.ufficio
            AND direzione_ufficio.fine IS NULL;
            IF valore > 0 THEN
                RAISE EXCEPTION 'Questo segretario è il dirigente, non puoi
rimuoverlo da qui';
            end if;
        end if;
    end if;
    return new;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER assegnazione_ufficio_update
    BEFORE UPDATE ON assegnazione_ufficio
    FOR EACH ROW
    EXECUTE PROCEDURE assegnazione_ufficio_update()
```

ASSISTENTE

- Controlliamo se l'impiegato è presente nella lista degli impiegati attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)

```
CREATE FUNCTION check_assistente() RETURNS trigger AS
$BODY$
    DECLARE
        valore integer;
    BEGIN
    --Controlliamo se l'impiegato è presente nella lista degli impiegati attuali
    --(e che quindi abbia un contratto valido e non abbia altre mansioni)
```

```
        SELECT count(*) INTO valore
        FROM assistenti_attuali
        where cf = new.impiegato;
    --Se è presente restituiamo il record
    IF valore > 0 THEN
        RETURN new;
    ELSE
        RAISE EXCEPTION 'L impiegato non è un assistente';
    END IF;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_assistente
    AFTER INSERT ON assistente
    FOR EACH ROW
    EXECUTE PROCEDURE check_assistente()
```

BIGLIETTO

-Ogni volta che si aggiunge un elemento bisogna aumentare il totale del saldo punti dell'account relativo e controllare se si è saliti di livello. Nel caso, aggiornare il livello.

```
create function accumulo_punti_funct() returns trigger
    language plpgsql
as
$$
BEGIN
        UPDATE account
        SET saldo_punti = saldo_punti + floor(new.prezzo)::integer
        WHERE codice_fedeltà = new.account;
        EXECUTE controllo_livello(new.account);
    END;
$$;


create trigger accumulo_punti_trigger
    after insert
    on biglietto
    for each row
execute procedure accumulo_punti_funct();

create function rimuovi_punti_funct() returns trigger
    language plpgsql
as
$$
BEGIN
        UPDATE account
        SET saldo_punti = saldo_punti - floor(new.prezzo)::integer
        WHERE codice_fedeltà = new.account;
        EXECUTE controllo_livello(new.account);
        RETURN NULL;
    END;
$$;

create trigger rimuovi_punti_trigger
    after delete
    on biglietto
    for each row
execute procedure rimuovi_punti_funct();
```

```
CREATE FUNCTION controllo_livello(fedeltà INTEGER) RETURNS VOID
    LANGUAGE plpgsql
AS
$BODY$
    DECLARE
        punti INTEGER;
        level livelli_account;
    BEGIN
        SELECT saldo_punti INTO punti
        FROM account
        WHERE codice_fedeltà = fedeltà;
        IF (punti > 0 AND punti < 1000) THEN level = 'Standard'; END IF;
        IF (punti >= 1000 AND punti < 2500) THEN level = 'Silver'; END IF;
        IF (punti >= 2500 AND punti <5000) THEN level = 'Gold'; END IF;
        IF (punti >= 5000) THEN level = 'Platinum'; END IF;
        UPDATE account
        SET livello = level
        WHERE codice_fedeltà = fedeltà;
    END;
$BODY$
```

-Controllare che gli account per la quale accumulo punti e il passeggero definito sul biglietto siano gli stessi

```
CREATE FUNCTION controllo_persona() RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$BODY$
    DECLARE
        EMAIL_CHECK EMAIL;
    BEGIN
        IF (new.account IS NULL) THEN RETURN NEW;
        else
            SELECT email INTO EMAIL_CHECK
            FROM account
            WHERE codice_fedeltà = new.account;
            IF (new.email = EMAIL_CHECK)
                THEN RETURN new;
            ELSE
                RAISE EXCEPTION 'Inconsistenza tra account e passeggero';
            END IF;
        END IF;
    END;
$$;
$BODY$;

CREATE TRIGGER controllo_persona_trigger
BEFORE INSERT ON biglietto
FOR EACH ROW
EXECUTE PROCEDURE controllo_persona();
```

-Controllare che le fermate di arrivo e di partenza della prenotazione siano possibili all'interno degli itinerari scelti.

```
CREATE FUNCTION controllo_fermate() RETURNS trigger AS
$$
DECLARE
        da integer;
        verso integer;
        valore integer;
    BEGIN
```

```
        SELECT sezione.* INTO da
        FROM sezione
        WHERE sezione.percorso = new.spostamento_percorso
        AND ((sezione.ordine = new.partenza_ordine - 1 AND sezione."fermata?")
        OR (sezione.ordine = 0 AND sezione.tratta_inizio IN (SELECT
s2.tratta_inizio
                                                             FROM sezione s2
                                                             WHERE s2.percorso =
new.spostamento_percorso
                                                             AND s2.ordine =
new.partenza_ordine)));


        SELECT count(*) INTO verso
        FROM sezione
        WHERE sezione.percorso = new.spostamento_percorso
        AND sezione.ordine = new.arrivo_ordine
        AND sezione."fermata?" = true;


        IF (da > 0) AND (verso > 0) THEN
    --Infine controlliamo se il posto scelto dall'utente è libero o già occupato
            SELECT count(DISTINCT biglietto.posto) INTO valore
            FROM biglietto, sezione partenza, sezione arrivo
            WHERE biglietto.spostamento_percorso = new.spostamento_percorso
            AND biglietto.data_viaggio = new.data_viaggio
            AND biglietto.carrozza = new.carrozza
            AND partenza.percorso = biglietto.spostamento_percorso
            AND arrivo.percorso = biglietto.spostamento_percorso
            AND partenza.ordine = biglietto.partenza_ordine
            AND arrivo.ordine = biglietto.arrivo_ordine
            AND biglietto.posto = new.posto
            AND ((arrivo.ordine >= new.partenza_ordine AND partenza.ordine <=
new.arrivo_ordine)
            OR (partenza.ordine <= new.arrivo_ordine AND arrivo.ordine >=
new.partenza_ordine));
            --Se il risultato della query è maggiore di 1, allora il posto è
occupato su tutto o su una parte del tracciato
            IF valore > 0 THEN
                RAISE EXCEPTION 'Il posto scelto non è valido, in questo
intervallo di stazioni questo posto è occupato';
            END IF;
            RETURN new;
        ELSE
            RAISE EXCEPTION 'Le stazioni non sono raggiungibili con questo
percorso';
        end if;
$$;
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_fermate_biglietto
    BEFORE INSERT ON biglietto
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_fermate()
```

CONTENUTO ASSISTENZA

-Controlliamo se a rispondere sono solo segretari autorizzati (ovvero che lavorano null'ufficio del supporto clienti) e che parlino la lingua della persona che ha scritto il messaggio originale

```
DECLARE
        segretarioc INTEGER;
    BEGIN

        SELECT count(*) INTO segretarioc
        FROM lingue_segretario, assegnazione_ufficio, ufficio
        WHERE lingue_segretario.segretario = new.segretario
        AND assegnazione_ufficio.segretario = new.segretario
        AND assegnazione_ufficio.fine IS NULL
        AND assegnazione_ufficio.ufficio = ufficio.identificativo
        AND lingue_segretario = (SELECT lingua
                                 FROM assistenza_clienti
                                 WHERE assistenza_clienti.identificativo =
new.assistenza);

        IF (segretarioc = 1) then
                return (NEW);
            else
                RAISE EXCEPTION 'Il segretario che sta rispondendo o non parla
quella lingua o non fa parte del supporto clienti';
        END IF;
    END
```

CONTRATTO

-Controllare che un contratto per una persona non sia vada ad accavallare con un altro contratto ancora attivo

```
create function controllo_contratto() returns trigger
    language plpgsql
as
$$
DECLARE
        data_fine DATE;
    BEGIN
        SELECT MAX(fine) INTO data_fine
        FROM contratto
        WHERE contratto.impiegato = new.impiegato
        IF (new.inizio < data_fine) then
            RAISE EXCEPTION 'Accavallameno Contratti';
        END IF;
        RETURN NEW;
END
$$;

create trigger controllo_contratto_trigger
    before insert
    on contratto
    for each row
execute procedure controllo_contratto();
```

DIREZIONE_UFFICIO

-Durante l'inserimento di completare l'incarico al dirigente precedente e di controllare che lavori nell'ufficio nella quale si sta dando l'incarico

```
CREATE FUNCTION insert_direzione() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
```

```
        vecchio_capo codice_fiscale;
        old_inizio TIMESTAMP;
    BEGIN
        --Controlliamo prima se il nuovo dirigente lavora in quell'ufficio
        SELECT count(*)
        FROM assegnazione_ufficio
        WHERE assegnazione_ufficio.ufficio = new.ufficio
        AND assegnazione_ufficio.segretario = new.segretario
        AND assegnazione_ufficio.fine IS NULL;

        IF valore = 0 THEN
            RAISE EXCEPTION 'Il nuovo dirigente non lavora in quell''ufficio';
        end if;

        --Se si, ci prendiamo il vecchio direttore (se esiste)
        SELECT segretario, inizio INTO vecchio_capo, old_inizio
        FROM direzione_ufficio
        WHERE ufficio = new.ufficio
        AND fine IS NULL;
        --E se esiste impostiamo la fine del suo mandato
        IF segretario IS NOT NULL THEN
            UPDATE segretario
            SET fine = current_timestamp
            WHERE ufficio = new.ufficio
            AND inizio = old_inizio
            AND ufficio = new.ufficio;
        end if;
        return new;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER insert_direzione
    BEFORE INSERT
    ON direzione_ufficio
    FOR EACH ROW
EXECUTE PROCEDURE insert_direzione();
```

FERROVIERE

-Controlliamo se il ferroviere è presente nella lista dei ferrovieri attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)

```
CREATE FUNCTION insert_ferroviere() RETURNS trigger
    LANGUAGE plpgsql
AS
$$
DECLARE
        valore INTEGER;
    BEGIN
        SELECT count(*) INTO valore
        FROM ferrovieri_attuali
        where cf = new.impiegato;
    IF valore > 0 THEN
        RETURN new;
    ELSE
        RAISE EXCEPTION 'L impiegato non è un ferroviere';
    END IF;
END
$$;
```

```
CREATE TRIGGER insert_ferroviere
    AFTER INSERT
    ON ferroviere
    FOR EACH ROW
EXECUTE PROCEDURE insert_ferroviere();
```

MANUTENZIONE

-Controllare che il treno di cui si sta iniziando la manutenzione sia effettivamente presente nel deposito

```
CREATE FUNCTION controllo_deposito() RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$BODY$
    DECLARE
        USCITA TIME;
        FINE VARCHAR(100);
        POSIZIONE_BANCO VARCHAR(100);
    BEGIN
        SELECT uscita, tratta_fine AS USCITA, FINE
        from sezione_effettiva
        where identificativo =
              (SELECT max(sezione_effettiva.identificativo)
               FROM sezione_effettiva, spostamento
               WHERE spostamento.treno = NEW.idtreno
               AND sezione_effettiva.percorso = spostamento.percorso
               AND sezione_effettiva.giorno = spostamento.giorno);

        IF USCITA IS NULL THEN
            RAISE EXCEPTION 'Treno in Viaggio';
        end if;

        SELECT deposito as POSIZIONE_BANCO
        FROM banco_manutenzione
        WHERE banco_manutenzione.identificativo = NEW.banco;

        IF POSIZIONE_BANCO = FINE THEN
            RETURN NEW;
        ELSE
            RAISE EXCEPTION 'Posizione errata';
        end if;
    END;
$BODY$;

CREATE TRIGGER controllo_posizione_treno
BEFORE INSERT ON manutenzione
FOR EACH ROW
EXECUTE PROCEDURE controllo_deposito();
```

-Controllare che il banco che si sta cercando di utilizzare non sia già occupato

```
CREATE FUNCTION assegnazione_banco() RETURNS trigger AS
$BODY$
    DECLARE
        valore integer;
    BEGIN
        SELECT count(*) INTO valore
        FROM manutenzione
        where manutenzione.banco = new.banco AND fine IS NULL;
    IF (valore = 0) then
        new.inizio = now()::timestamp;
```

```
        return (NEW);
    else
        RAISE EXCEPTION 'Doppia_Assegnazione' USING HINT = 'Questo banco è già
stato assegnato ad un altro treno';
    end if;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_banco
    BEFORE INSERT ON manutenzione
    FOR EACH ROW
    EXECUTE PROCEDURE assegnazione_banco()
```

OPERAIO

-Controlliamo se l'operaio è presente nella lista degli operai attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)

```
CREATE FUNCTION check_operaio() RETURNS trigger AS
$BODY$
    DECLARE
        valore integer;
    BEGIN
        SELECT count(*) INTO valore
        FROM operai_attuali
        where cf = new.impiegato;
    IF valore > 0 THEN
        return new;
    else
        RAISE EXCEPTION 'L' impiegato non è un operaio';
    end if;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_operaio
    AFTER INSERT ON operaio
    FOR EACH ROW
    EXECUTE PROCEDURE check_operaio()
```

PERCORSO

-Non modificare un percorso se è già associato ad uno spostamento e aggiornare il percorso solo con dati validi

```
CREATE FUNCTION modifica_percorso() RETURNS trigger AS
$BODY$
DECLARE
        valore INTEGER;
        uscita TIME;
        arrivo VARCHAR(100);
    BEGIN
        SELECT count(*) INTO valore
        FROM spostamento
        WHERE spostamento.percorso = new.identificativo;
        IF (valore > 0) THEN
            RAISE EXCEPTION 'Il percorso è già associato ad uno spostamento';
        END IF;
```

```
        IF new.tipologia != old.tipologia THEN
            SELECT count(*) INTO valore
            FROM sezione
            WHERE sezione.percorso = new.identificativo;

            IF valore > 0 THEN
                RAISE EXCEPTION 'Non puoi modificare la tipologia di un percorso
che ha già delle sezioni';
            end if;
        end if;


        SELECT count(*) INTO valore
        FROM sezione
        WHERE sezione.percorso = new.identificativo;

        IF valore = 0 AND new.ora_arrivo = null AND new.ora_partenza = null
               AND new.arrivo = null and new.partenza = null THEN
            RETURN new;
        end if;

        IF (new.partenza IS NOT NULL AND new.ora_partenza IS NOT NULL) THEN
            SELECT sezione.tratta_inizio, sezione.entrata_programmata INTO
arrivo, uscita
            FROM sezione
            WHERE sezione.percorso = new.identificativo
            AND sezione.ordine = 0;

            IF (uscita != new.ora_partenza OR arrivo != new.partenza) THEN
                RAISE EXCEPTION 'Dati non validi % % % %', arrivo, new.partenza,
uscita, new.ora_partenza;
            end if;
        end if;

        IF (new.ora_arrivo IS NOT NULL AND new.arrivo IS NOT NULL) THEN
            SELECT sezione.uscita_programmata, sezione.tratta_fine INTO uscita,
arrivo
            FROM sezione
            WHERE sezione.percorso = new.identificativo
            AND sezione.ordine =    (SELECT max(ordine)
                                     FROM sezione
                                     WHERE sezione.percorso =
new.identificativo);
            IF (uscita != new.ora_arrivo and arrivo != new.arrivo) THEN
                RAISE EXCEPTION 'Dati non validi % % % %', arrivo, new.arrivo,
uscita, new.ora_arrivo;
            end if;
        end if;
        RETURN new;
    END
$$;
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_possibile_modifica_percorso
    AFTER UPDATE ON percorso
    FOR EACH ROW
    EXECUTE PROCEDURE modifica_percorso()
```

-In fase iniziale di creazione, non inserire l'ora di arrivo e l'ora di partenza

```
CREATE FUNCTION reset_arrivo_percorso() RETURNS trigger AS
$BODY$
BEGIN
        new.ora_arrivo = NULL;
        new.arrivo = NULL;
        new.ora_partenza = NULL;
        new.partenza = NULL;
        RETURN new;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER reset_arrivo
    BEFORE INSERT ON percorso
    FOR EACH ROW
    EXECUTE PROCEDURE reset_arrivo_percorso()
```

PERSONALE DI BORDO

-Controllare che il capotreno sia un capotreno e il macchinista sia un macchinista (anche durante un eventuale aggiornamento dei dati)

```
CREATE FUNCTION check_ferroviere() RETURNS trigger AS
$BODY$
    DECLARE
        macchinista INTEGER;
        capotreno INTEGER;
    BEGIN
        SELECT count(*) INTO macchinista
        FROM macchinisti_attuali
        WHERE impiegato = new.macchinista;

        SELECT count(*) INTO capotreno
        FROM capotreni_attuali
        WHERE impiegato = new.capotreno;

        IF (macchinista > 0) then
            IF (capotreno > 0) then
                return (NEW);
            else
                RAISE EXCEPTION 'Capotreno Error' USING HINT = 'Il capotreno
definito non è un capotreno';
            end if;
        else
            RAISE EXCEPTION 'Macchinista Error' USING HINT = 'Il macchinista
definito non è un macchinista';
        end if;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_ferrovieri
    BEFORE INSERT ON personale_bordo
    FOR EACH ROW
    EXECUTE PROCEDURE check_ferroviere()

CREATE TRIGGER controllo_ferrovieri _update
    BEFORE UPDATE
    ON personale_bordo
    FOR EACH ROW
EXECUTE PROCEDURE check_ferroviere();
```

SEGRETARIO

- Controlliamo se il segretario è presente nella lista dei segretari attuali (e che quindi abbia un contratto valido e non abbia altre mansioni)

```
CREATE FUNCTION check_segretario() RETURNS trigger AS
$BODY$
    DECLARE
        valore integer;
    BEGIN
        SELECT count(*) INTO valore
        FROM segretari_attuali
        where cf = new.impiegato;
    IF valore > 0 THEN
        return new;
    else
        RAISE EXCEPTION 'L' impiegato non è un segretario';
    end if;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_segretario
    AFTER INSERT ON segretario
    FOR EACH ROW
    EXECUTE PROCEDURE check_segretario()
```

SPOSTAMENTO

- L'ultima stazione referenziata dal percorso deve essere anche una fermata se la tipologia del percorso è impostata a "Viaggio"

```
CREATE FUNCTION controllo_fermata_finale() RETURNS trigger AS
$BODY$
    DECLARE
        valore BOOLEAN;
        tipologia tipo_spostamento;
    BEGIN

        SELECT percorso.tipologia INTO tipologia
        FROM percorso
        WHERE percorso.identificativo = new.percorso;

        IF tipologia = 'Spostamento' THEN
            RETURN NEW;
        END IF;

        SELECT "fermata?" INTO valore
        FROM sezione
        WHERE sezione.percorso = new.percorso
        AND sezione.ordine = (SELECT max(ordine)
                              FROM sezione
                              WHERE sezione.percorso = new.percorso);

        IF NOT valore THEN
            RAISE EXCEPTION 'Lo spostamento non fa fermata nella ultima
sezione';
        END IF;
```

```
        RETURN NEW;
END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_fermata_finale
    BEFORE INSERT ON spostamento
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_fermata_finale()
```

-Controllare che i ferrovieri stiano lavorando durante il loro turno e che non siano già assegnati ad un altro spostamento

```
CREATE FUNCTION controllo_turno_ferroviere() RETURNS trigger AS
$BODY$
    DECLARE
        assistenti RECORD;
        inizio_percorso TIME;
        fine_percorso TIME;
        inizio_turno TIME;
        fine_turno TIME;
        gruppi INTEGER;
    BEGIN
        SELECT count(*) INTO gruppi
        FROM spostamento
        WHERE idgruppo = NEW.idgruppo;
        IF gruppi > 0 THEN
                RAISE EXCEPTION 'Questo gruppo è già stato assegnato';
        END IF;


        SELECT ora_arrivo, ora_partenza INTO fine_percorso, inizio_percorso
        FROM percorso
        WHERE new.percorso = percorso.identificativo;

        SELECT inizio, fine INTO inizio_turno, fine_turno
        FROM turno, personale_bordo
        WHERE new.idgruppo = personale_bordo.identificativo
        AND turno.impiegato = personale_bordo.capotreno;


        IF (inizio_turno > inizio_percorso OR inizio_turno IS NULL) OR
(fine_turno < fine_percorso OR fine_turno IS NULL) THEN
                    RAISE EXCEPTION 'Il capotreno non ha il turno in quel
periodo';
                end if;

        SELECT inizio, fine INTO inizio_turno, fine_turno
        FROM turno, personale_bordo
        WHERE new.idgruppo = personale_bordo.identificativo
        AND turno.impiegato = personale_bordo.macchinista;

        RAISE NOTICE 'Inizio Percorso %, Fine Percorso %', inizio_percorso,
fine_percorso;
        RAISE NOTICE 'inizio_turno %, fine_turno %', inizio_turno, fine_turno;

        IF (inizio_turno > inizio_percorso OR inizio_turno IS NULL) OR
(fine_turno < fine_percorso OR fine_turno IS NULL) THEN
                    RAISE EXCEPTION 'Il macchinista non ha il turno in quel
periodo';
```

```
                end if;

        FOR assistenti IN
                        SELECT servizio_assistente.assistente
                        FROM personale_bordo, servizio_assistente
                        WHERE servizio_assistente.idgruppo = new.idgruppo
            loop
                SELECT inizio, fine INTO inizio_turno, fine_turno
                FROM turno, personale_bordo
                WHERE new.idgruppo = personale_bordo.identificativo
                AND turno.impiegato = assistenti.assistente;

        IF (inizio_turno > inizio_percorso OR inizio_turno IS NULL) OR
(fine_turno < fine_percorso OR fine_turno IS NULL) THEN
                    RAISE EXCEPTION 'L''assistente % non ha il turno in quel
periodo', assistenti.assistente;
                end if;
            end loop;
        RETURN new;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_turno_ferroviere
    BEFORE UPDATE ON spostamento
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_turno_ferroviere()

CREATE TRIGGER controllo_turno_ferroviere_insert
    BEFORE INSERT
    ON spostamento
    FOR EACH ROW
EXECUTE PROCEDURE controllo_turno_ferroviere();
```

-Controllare che gli spostamenti non si vadano ad accavallare su una determinata tratta

```
CREATE FUNCTION controllo_accavallamento() RETURNS trigger AS
$BODY$
    DECLARE
        f RECORD;
        valore INTEGER;
    BEGIN
        FOR f IN
                SELECT *
                FROM sezione
                WHERE sezione.percorso = new.percorso
            LOOP
                SELECT count(*) INTO valore
                FROM sezione, percorso p1, spostamento
                WHERE spostamento.giorno = new.giorno
                AND spostamento.percorso = p1.identificativo
                AND sezione.percorso = p1.identificativo
                AND uscita_programmata <= f.uscita_programmata
                AND entrata_programmata >= f.entrata_programmata;

                IF valore > 0 THEN
                    RAISE EXCEPTION 'Il treno durante questo viaggio si scontra
durante il viaggio con un altro treno nella sezione numero %', f.ordine;
                end if;
            end loop;
```

```
        RETURN new;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_accavallamento_spostamento
    BEFORE INSERT ON spostamento
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_accavallamento()
```

-Un treno può partire solo dal punto di arrivo dell'ultimo viaggio e solo se il percorso è stato definito

```
CREATE FUNCTION controllo_partenza_treno() RETURNS trigger AS
$BODY$
DECLARE
        ora_arrivo_percorso TIME;
        partenza_percorso VARCHAR(100);
        ora_partenza_check TIME;
        arrivo_check VARCHAR(100);
        giorno_check DATE;
        valore INTEGER;
    BEGIN
        SELECT partenza, ora_partenza INTO partenza_percorso
,ora_arrivo_percorso
        FROM percorso
        WHERE identificativo = new.percorso;

        -- Se il percorso non contiene tratte, allora impossibile
        IF ora_partenza_check = NULL THEN
            RAISE EXCEPTION 'Il percorso non è ancora stato definito, definisci
prima un percorso';
        end if;

        -- Se il treno è un nuovo treno, basta che parta da un deposito
        SELECT count(*) INTO valore
        FROM spostamento
        WHERE spostamento.treno = new.treno;

        IF valore = 0 THEN
            SELECT count(*) INTO valore
            FROM deposito, percorso
            WHERE percorso.identificativo = new.percorso
            AND deposito.nome = percorso.partenza;
            IF valore > 0 THEN
                RETURN new;
            end if;
        end if;

        -- Altrimenti ci prendiamo i dati dell'ultimo spostamento del treno
        SELECT p0.arrivo, p0.ora_arrivo, s0.giorno INTO arrivo_check,
ora_partenza_check, giorno_check
        FROM spostamento s0, percorso p0
        WHERE s0.percorso = p0.identificativo
        AND s0.treno = new.treno
        AND s0.giorno = (SELECT max(s1.giorno)
                         FROM spostamento s1
                         WHERE s1.treno = new.treno)
        AND p0.ora_arrivo >= all (SELECT ora_partenza
                                  FROM percorso p1, spostamento s1
                                  WHERE s1.percorso = p1.identificativo
                                  AND s1.treno = new.treno
```

```
                                        AND s1.giorno = (SELECT MAX(s2.giorno)
                                                         FROM spostamento s2
                                                         WHERE s2.treno = new.treno));

        IF (arrivo_check = partenza_percorso) THEN
            IF (giorno_check < new.giorno) THEN
                RETURN new;
            ELSE
                IF (giorno_check = new.giorno AND ora_arrivo_percorso >
ora_partenza_check) THEN
                    RETURN NEW;
                ELSE
                    RAISE EXCEPTION 'Il treno a quella ora è ancora in viaggio';
                end if;
            end if;
        end if;
        RAISE EXCEPTION 'Il percorso non è compatibile con la ultima posizione
finale del treno';
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_partenza_treno
    BEFORE INSERT ON spostamento
    FOR EACH ROW
    EXECUTE PROCEDURE controllo_partenza_treno()

CREATE TRIGGER controllo_partenza_treno_update
    BEFORE UPDATE
    ON spostamento
    FOR EACH ROW
EXECUTE PROCEDURE controllo_partenza_treno();
```

-Non cancellare gli spostamenti effettuati nel passato o che non siano l'ultimo spostamento programmato per quel treno

```
CREATE FUNCTION check_delete_spostamento() RETURNS trigger AS
$BODY$
    DECLARE
        orario TIME;
        giorno_max DATE;
        orario_max TIME;
    BEGIN
        --Controlliamo che la data non sia nel passato (vuol dire che lo
spostamento è stato già effettuato)
        IF old.giorno < current_date THEN
            RAISE EXCEPTION 'Non puoi cancellare spostamenti effettuati nel
passato';
        end if;
        --Se il giorno è lo stesso, controlliamo che l'orario sia maggiore
dell'ora attuale
        -- (altrimenti vorrebbe dire che il treno è già partito)
        IF old.giorno = current_date THEN
            SELECT ora_partenza INTO orario
            FROM percorso
            WHERE identificativo = old.percorso;
            IF orario < current_time THEN
                RAISE EXCEPTION 'Non puoi cancellare spostamenti effettuati nel
passato';
            end if;
        end if;
```

```
        SELECT max(spostamento.giorno), max(percorso.ora_partenza) INTO
giorno_max, orario_max

        FROM spostamento, percorso
        WHERE spostamento.percorso = percorso.identificativo
        AND spostamento.treno = new.treno;

        --Infine controlliamo che sia l'ultimo viaggio programmato per quel
treno, altrimenti dare errore
        IF giorno_max > new.giorno THEN
            RAISE EXCEPTION 'Non puoi cancellare uno spostamento con degli
spostamenti futuri che dipendono da questo';
        end if;
        IF giorno_max = new.giorno THEN
            IF orario_max > orario THEN
                RAISE EXCEPTION 'Non puoi cancellare uno spostamento con degli
spostamenti futuri che dipendono da questo';
            end if;
        end if;
        return old;
    END
  LANGUAGE plpgsql;

create trigger check_delete_spostamento
    before delete
    on spostamento
    for each row
execute procedure check_delete_spostamento();
```

SEZIONE EFFETTIVA

-Quando si inserisce un record, il valore di entrata deve essere maggiore del valore dell'uscita precedente, se esiste

```
CREATE FUNCTION check_insert_sezione_effettiva() RETURNS trigger AS
$BODY$
    DECLARE
        partenza TIME;
    BEGIN
        IF NEW.sezione_ordine = 0 THEN
            SELECT entrata_programmata INTO partenza
            FROM sezione
            WHERE sezione.percorso = new.percorso
            AND sezione.ordine = 0;

            IF new.entrata >= partenza THEN
                RETURN new;
            ELSE
                RAISE EXCEPTION 'Non puoi partire prima della partenza';
            end if;
        end if;

        SELECT uscita INTO partenza
        FROM sezione_effettiva
        WHERE sezione_ordine = new.sezione_ordine-1
        AND sezione_effettiva.percorso = new.percorso
        AND sezione_effettiva.giorno = new.giorno;

        IF new.entrata >= partenza THEN
                RETURN new;
            ELSE
```

```
                RAISE EXCEPTION 'Inconsistenza sugli orari di uscita e entrata
tra due sezioni';
            end if;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER check_insert_sezione_effettiva
    BEFORE INSERT ON sezione_effettiva
    FOR EACH ROW
    EXECUTE PROCEDURE check_insert_sezione_effettiva()
```

-Quando si aggiorna un record, il valore di uscita deve essere maggiore del valore di entrata

```
CREATE FUNCTION check_update_sezione_effettiva() RETURNS trigger AS
$BODY$
    DECLARE
        arrivo TIME;
    BEGIN
        SELECT entrata INTO arrivo
        FROM sezione_effettiva
        WHERE sezione_ordine = new.sezione_ordine-1
        AND sezione_effettiva.percorso = new.percorso
        AND sezione_effettiva.giorno = new.giorno;

        IF new.uscita >= arrivo THEN
                RETURN new;
            ELSE
                RAISE EXCEPTION 'Inconsistenza sugli orari di uscita e entrata
nella sezione';
            end if;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER check_update_sezione_effettiva
    BEFORE UPDATE ON sezione_effettiva
    FOR EACH ROW
    EXECUTE PROCEDURE check_update_sezione_effettiva()
```

-Non si può cancellare un record di sezione_effettiva

```
CREATE FUNCTION stop_remove() RETURNS trigger AS
$BODY$
    BEGIN
        RAISE EXCEPTION 'Questo record non può essere rimosso';
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER block_remove_sezione_effettiva
    BEFORE DELETE ON sezione_effettiva
    FOR EACH ROW
    EXECUTE PROCEDURE stop_remove()
```

-Quando si aggiorna il campo uscita_effettiva di una sezione bisogna controllare che sia l'ultima dell'itinerario. Nel caso sia l'ultima, controllare che i ferrovieri interessati non abbiano superato l'orario del turno. Nel caso, aggiungere lo straordinario

```
CREATE FUNCTION check_straordinario() RETURNS trigger AS
$BODY$
    DECLARE
        ordine_massimo INTEGER;
        ora_arrivo TIME;
        fine_turno TIME;
```

```
        impiegati RECORD;
        valore INTEGER;
    BEGIN
        SELECT max(ordine) INTO ordine_massimo
        FROM sezione
        WHERE sezione.percorso = new.percorso;

        IF new.sezione_ordine != ordine_massimo THEN
            RETURN new;
        END IF;

        SELECT sezione.uscita_programmata INTO ora_arrivo
        FROM sezione
        WHERE sezione_ordine = ordine_massimo
        AND sezione.percorso = new.percorso;

        IF ora_arrivo >= new.uscita THEN
            RETURN new;
        END IF;
        --Ci prendiamo i codici fiscali di tutti gli impiegati presenti sul
treno durante lo spostamento e iteriamo su essi
        FOR impiegati IN SELECT impiegato.cf
                         FROM impiegato
                         WHERE cf = (SELECT ferroviere.impiegato
                         FROM ferroviere, spostamento, personale_bordo
                         WHERE new.percorso = spostamento.percorso
                         AND new.giorno = spostamento.giorno
                         AND spostamento.idgruppo = personale_bordo.identificativo
                         AND (capotreno = ferroviere.impiegato
                         OR macchinista = ferroviere.impiegato))
                         OR cf =     (SELECT assistente.impiegato
                          FROM assistente, spostamento, servizio_assistente
                          WHERE new.percorso = spostamento.percorso
                          AND new.giorno = spostamento.giorno
                          AND spostamento.idgruppo = servizio_assistente.idgruppo
                        AND servizio_assistente.assistente = assistente.impiegato)
LOOP
        --Ci prendiamo la fine del turno dell'impiegato che stiamo analizzando
            SELECT fine INTO fine_turno
            FROM turno
            WHERE impiegati.cf = turno.impiegato
            AND new.giorno = turno.giorno
            AND turno.straordinario = false;

            --Nel caso ci sia bisogno di aggiungere uno straordinario
            IF fine_turno < new.uscita THEN
                SELECT count(*) INTO valore
                FROM turno
                WHERE impiegati.cf = turno.impiegato
                AND new.giorno = turno.giorno
                AND turno.straordinario = true;
                --Rimuoviamo l'eventuale già presente straordinario...
                IF valore > 0 THEN
                    DELETE FROM turno
                    WHERE impiegati.cf = turno.impiegato
                    AND turno.giorno = new.giorno
                    AND turno.straordinario = true;
                END IF;
                --E lo "aggiorniamo" aggiungendo quello nuovo (o semplicemente
ne aggiungiamo uno nuovo)
                INSERT INTO turno (impiegato, giorno, inizio, fine,
straordinario)
```

```
                VALUES (impiegati.cf, new.giorno, fine_turno, new.uscita, true);
            END IF;
        END LOOP;

        RETURN NEW;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER check_straordinario
    BEFORE UPDATE ON sezione_effettiva
    FOR EACH ROW
    EXECUTE PROCEDURE check_straordinario()
```

SEZIONE

-Impostiamo correttamente l'ordine della sezione

```
CREATE FUNCTION set_ordine() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
    BEGIN
        SELECT count(*) INTO valore
        FROM sezione
        WHERE percorso = new.percorso;
        new.ordine = valore;
        RETURN new;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER set_ordine
    BEFORE INSERT ON sezione
    FOR EACH ROW
    EXECUTE PROCEDURE set_ordine()
```

-Non si possono modificare le sezioni

```
CREATE FUNCTION update_sezione() RETURNS trigger AS
$BODY$
    DECLARE
    BEGIN
        RAISE EXCEPTION 'Non puoi modificare le sezioni, devi prima
cancellarle';
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER modify_sezione
    AFTER UPDATE ON sezione
    FOR EACH ROW
    EXECUTE PROCEDURE update_sezione()
```

-Non si può cancellare una sezione che è già associata ad uno spostamento o che abbia valore di ordine minore del massimo valore per quel percorso.
Nel caso si possa cancellare la sezione, modificare i dati del percorso.

```
CREATE FUNCTION delete_sezione() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
        sezione_precedente VARCHAR(100);
```

```
        uscita_precedente TIME;
    BEGIN
        SELECT count(*) INTO valore
        FROM spostamento
        WHERE percorso = new.percorso;

        IF valore > 0 THEN
            RAISE EXCEPTION 'Non puoi rimuovere sezioni a questo percorso';
        end if;

        SELECT count(*) INTO valore
        FROM sezione
        WHERE sezione.percorso = old.percorso;

        IF valore = 0 THEN
            UPDATE percorso
            SET partenza = null, ora_partenza = null, arrivo = null, ora_arrivo
= null
            WHERE percorso.identificativo = old.percorso;
        end if;

        IF valore = old.ordine THEN
            SELECT sezione.tratta_fine, uscita_programmata INTO
sezione_precedente, uscita_precedente
            FROM sezione
            WHERE sezione.ordine = valore - 1
            AND sezione.percorso = old.percorso;

            UPDATE percorso
            SET arrivo = sezione_precedente, ora_arrivo = uscita_precedente
            WHERE identificativo = old.percorso;
        ELSE
            RAISE EXCEPTION 'Non puoi rimuovere questa sezione';
        end if;
        RETURN old;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER delete_sezione
    AFTER DELETE ON sezione
    FOR EACH ROW
    EXECUTE PROCEDURE delete_sezione()
```

-Ogni volta che si aggiunge una sezione, aggiornare l'ora di arrivo e il punto di interesse di arrivo e la compatibilità con la precedente

Inoltre non si possono aggiungere sezioni ad un percorso che ha già uno spostamento

Inoltre controllare se la sezione è compatibile con la tipologia di percorso specificato

Inoltre se si passa in una stazione, controllare che si sia specificato un binario dove passare

```
CREATE FUNCTION insert_sezione() RETURNS trigger AS
$BODY$
    DECLARE
        valore INTEGER;
        sezione_precedente VARCHAR(100);
        uscita_precedente TIME;
        sosta_precedente INTEGER;
        tipologia2 tipo_spostamento;
    BEGIN
        SELECT count(*) INTO valore
        FROM spostamento
        WHERE percorso = new.percorso;
```

```
        IF valore > 0 THEN
            RAISE EXCEPTION 'Non puoi aggiungere sezioni a questo percorso';
        end if;

        IF new.ordine > 0 THEN
            SELECT sezione.tratta_fine, uscita_programmata, sosta INTO
sezione_precedente, uscita_precedente, sosta_precedente
            FROM sezione
            WHERE ordine = new.ordine - 1
            AND sezione.percorso = new.percorso;

            uscita_precedente = uscita_precedente + ((sosta_precedente ||
'MINUTE')::INTERVAL);
            IF sezione_precedente != new.tratta_inizio OR uscita_precedente !=
new.entrata_programmata THEN
                RAISE EXCEPTION 'La sezione non è compatibile con la
precedente';
            end if;
        ELSE
            UPDATE percorso
            SET partenza = new.tratta_inizio, ora_partenza =
new.entrata_programmata
            WHERE identificativo = new.percorso;
        end if;

        SELECT count(*) INTO valore
        FROM deposito
        WHERE nome = new.tratta_inizio or nome = new.tratta_fine;

        SELECT percorso.tipologia INTO tipologia2
        FROM percorso
        WHERE percorso.identificativo = new.percorso;

        IF valore > 0 AND tipologia2 = 'Viaggio' THEN
            RAISE EXCEPTION 'Non puoi visitare dei depositi mentre fai servizio
viaggiatori';
        end if;

        SELECT count(*) INTO valore
        FROM stazione
        WHERE nome = new.tratta_fine;

        IF valore > 0 AND new.binario IS NULL THEN
            RAISE EXCEPTION 'Specificare su quale binario il treno farà
sosta/passerà';
        END IF;

        IF new."fermata?" THEN
            IF valore = 0 THEN
                RAISE EXCEPTION 'La tratta finale non è una stazione, pertanto
non possiamo fare fermata viaggiatori';
            end if;

            IF tipologia2 = 'Spostamento' THEN
                RAISE EXCEPTION 'Non puoi fare fermate viaggiatori durante uno
spostamento';
            end if;
        end if;

        UPDATE percorso
        SET arrivo = new.tratta_fine, ora_arrivo = new.uscita_programmata
```

```
        WHERE identificativo = new.percorso;

        RETURN new;
    END
$$;
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER insert_sezione
    AFTER INSERT ON sezione
    FOR EACH ROW
    EXECUTE PROCEDURE insert_sezione()
```

TURNO

-Controllare che il turno di un impiegato ricada in un periodo in cui lui ha il contratto

```
CREATE FUNCTION contratto_valido() RETURNS TRIGGER AS
$BODY$
DECLARE
        contratti INTEGER;
        tempo_fine TIME;
        tipol tipo_contratto;
    BEGIN
        SELECT MAX(fine) INTO tempo_fine
        FROM turno
        WHERE impiegato = new.impiegato
        AND giorno = new.giorno
        GROUP BY giorno;

        IF (tempo_fine IS NOT NULL AND new.fine < tempo_fine) then
            RAISE EXCEPTION 'Accavallameno Turni';
        END IF;

        SELECT contratto.identificativo, contratto.tipo INTO contratti, tipol
        FROM contratto
        WHERE contratto.impiegato = new.impiegato
        AND new.giorno > contratto.inizio
        AND new.giorno < contratto.fine;

        IF (contratti IS NOT NULL) THEN
            IF tipol = 'Part_Time' AND (new.inizio - new.fine) > interval '4
hours' THEN
                RAISE EXCEPTION 'Un part_time non può lavorare più di 4 ore';
            end if;
            RETURN NEW;
        ELSE RAISE EXCEPTION 'No Contratto';
        END IF;
    END
$BODY$
  LANGUAGE plpgsql;

CREATE TRIGGER controllo_contratto_trigger
    BEFORE INSERT ON turno
    FOR EACH ROW
    EXECUTE PROCEDURE contratto_valido()
```

-Non cancellare turni nel passato o uno nel futuro in cui un ferroviere è impegnato

```
CREATE FUNCTION check_delete_turno() RETURNS trigger AS
$BODY$
DECLARE
```

```
    valore INTEGER;
BEGIN
        --Controlliamo che la data non sia nel passato (vuol dire che il turno è
stato già effettuato)
        IF old.giorno < current_date THEN
            RAISE EXCEPTION 'Non puoi cancellare un turno effettuato nel
passato';
        end if;
        --Se il giorno è lo stesso, controlliamo che l'orario di uscita sia
maggiore dell'ora attuale
        -- (altrimenti vorrebbe dire che il turno è finito)
        IF old.giorno = current_date THEN
            IF old.fine < current_time THEN
                IF old.straordinario = false THEN
     RAISE EXCEPTION 'Non puoi cancellare un turno effettuato nel passato';
END IF;
            end if;
        end if;

        SELECT count(*) INTO valore
        FROM ((spostamento
        INNER JOIN personale_bordo on spostamento.idgruppo =
personale_bordo.identificativo)
        INNER JOIN servizio_assistente on spostamento.idgruppo =
servizio_assistente.idgruppo)
        WHERE spostamento.giorno=old.giorno AND
(personale_bordo.capotreno=old.impiegato OR
personale_bordo.macchinista=old.impiegato OR
servizio_assistente.assistente=old.impiegato);
        IF valore>0 THEN
            RAISE EXCEPTION 'Non puoi cancellare un turno futuro già assegnato';
        end if;

        return old;
    END
$BODY$
  LANGUAGE plpgsql;

create trigger check_delete_turno
    before delete
    on turno
    for each row
execute procedure check_delete_turno();
```

-Non modificare turni nel futuro in cui un ferroviere è occupato

```
CREATE FUNCTION check_update_turno() RETURNS trigger
    LANGUAGE plpgsql
AS
$$
DECLARE
        valore INTEGER;
    BEGIN

        SELECT count(*) INTO valore
        FROM spostamento
        INNER JOIN personale_bordo on spostamento.idgruppo =
personale_bordo.identificativo
        INNER JOIN servizio_assistente on spostamento.idgruppo =
servizio_assistente.idgruppo
        INNER JOIN percorso on spostamento.percorso =
public.percorso.identificativo
        WHERE spostamento.giorno=old.giorno AND
```

```
(personale_bordo.capotreno=old.impiegato OR
personale_bordo.macchinista=old.impiegato OR
servizio_assistente.assistente=old.impiegato)
        AND (new.fine<old.fine AND new.fine<public.percorso.ora_arrivo) OR
(new.inizio>old.inizio AND new.inizio>public.percorso.ora_partenza );
        IF valore>0 THEN
            RAISE EXCEPTION 'Non puoi modificare un turno rimuovendo orari già
assegnati';
        end if;

        return new;
    END

$$;
CREATE TRIGGER check_update_turno
    BEFORE UPDATE
    ON turno
    FOR EACH ROW
EXECUTE PROCEDURE check_update_turno();
```

Operazioni

Inserimento Persona

```
INSERT INTO persona (nome, cognome, email, cellulare) VALUES ('...', '...',
'...', '...');
```

Creazione di un account

```
INSERT INTO account (codice_fedeltà, account_password, email)
VALUES ('...', '...', '...', '...');
```

Acquisto Biglietto

```
CREATE FUNCTION acquisto_biglietto(emailnew email, spostgiornonew date,
spostpercorsonew integer, classeviaggionew character varying, partenzaf
character varying, arrivof character varying)

RETURNS TABLE("IdBiglietto" integer, "Nome" character varying, "Cognome"
character varying, "Treno" smallint, "Carrozza" smallint, "Posto" smallint,
"Partenza" character varying, "Ora Partenza" time without time zone, "Arrivo"
character varying, "Ora Arrivo" time without time zone, "Data_Viaggio" date,
"Prezzo Biglietto" numeric)
    LANGUAGE plpgsql
AS
$$
DECLARE
        numeroPosto integer;
        numeroCarrozza integer;
        ordinePartenza integer;
        ordineArrivo integer;
        costoKM integer;
        metriTot integer;
        accountUt integer;
        trenoSpost integer;
        valore INTEGER;
    BEGIN
        SELECT spos.treno INTO trenoSpost
        FROM spostamento spos
        WHERE spos.giorno=spostGiornoNew AND spos.percorso=spostPercorsoNew;

        SELECT ordine INTO ordinePartenza
        FROM sezione
        WHERE sezione.percorso = spostpercorsonew
        AND sezione.tratta_inizio = partenzaf;

        SELECT ordine INTO ordineArrivo
        FROM sezione
        WHERE sezione.percorso = spostpercorsonew
        AND sezione.tratta_fine = arrivof;

        SELECT MIN(c.numero) INTO numeroCarrozza
        FROM carrozza c
        WHERE c.classe_viaggio=classeViaggioNew
        AND c.idtreno = trenoSpost
        AND (SELECT count(*)
             FROM posto
             WHERE posto.idtreno = c.idtreno
             AND c.numero = posto.carrozza) != (SELECT count(DISTINCT
biglietto.posto)
                        FROM biglietto, sezione partenza, sezione arrivo
                        WHERE biglietto.spostamento_percorso = spostPercorsoNew
                        AND biglietto.data_viaggio = spostgiornonew
                        AND biglietto.carrozza = c.numero
```

```
                        AND partenza.percorso = biglietto.spostamento_percorso
                        AND arrivo.percorso = biglietto.spostamento_percorso
                        AND partenza.ordine = biglietto.partenza_ordine
                        AND arrivo.ordine = biglietto.arrivo_ordine
                        AND ((arrivo.ordine >= ordinePartenza AND
partenza.ordine <= ordineArrivo)
                        OR (partenza.ordine <= ordineArrivo AND arrivo.ordine >=
ordinePartenza)));
        IF numeroCarrozza IS NULL THEN
            RAISE EXCEPTION 'Non sono rimasti posti';
        END IF;

        SELECT MIN(p.numero) INTO numeroPosto
                    from posto p
                    where p.numero NOT IN (SELECT DISTINCT biglietto.posto
                        FROM biglietto, sezione partenza, sezione arrivo
                        WHERE biglietto.spostamento_percorso = spostPercorsoNew
                        AND biglietto.data_viaggio = spostgiornonew
                        AND biglietto.carrozza = numeroCarrozza
                        AND partenza.percorso = biglietto.spostamento_percorso
                        AND arrivo.percorso = biglietto.spostamento_percorso
                        AND partenza.ordine = biglietto.partenza_ordine
                        AND arrivo.ordine = biglietto.arrivo_ordine
                        AND ((arrivo.ordine >= ordinePartenza AND
partenza.ordine <= ordineArrivo)
                        OR (partenza.ordine <= ordineArrivo AND arrivo.ordine >=
ordinePartenza)))
                        AND p.idtreno = trenoSpost
                        AND p.carrozza = numeroCarrozza;

        IF numeroPosto IS NULL THEN
            RAISE EXCEPTION 'Cè stato un problema';
        END IF;


        SELECT ac.codice_fedeltà INTO accountUt
        FROM account ac
        WHERE emailNew = ac.email;

        SELECT prezzo INTO costoKM
        FROM classe_viaggio
        WHERE classeViaggioNew = classe_viaggio.nome;

        SELECT sum(tratta.lunghezza) INTO metriTot
                FROM tratta, sezione
                WHERE tratta.fine = sezione.tratta_fine
                AND tratta.inizio = sezione.tratta_inizio
                AND tratta.limite_velocità = sezione.tratta_limit
                AND sezione.percorso = spostPercorsoNew
                AND sezione.ordine >= (SELECT ordine
                                       FROM sezione s1
                                       WHERE s1.percorso = spostPercorsoNew
                                       AND s1.tratta_inizio = partenzaF)
                AND sezione.ordine <= (SELECT ordine
                                       FROM sezione s1
                                       WHERE s1.percorso = spostPercorsoNew
                                       AND s1.tratta_fine = arrivoF);

        IF metriTot = 0 OR metriTot IS NULL THEN
            RAISE EXCEPTION 'Errore sul calcolo della lunghezza della tratta';
        END IF;
```

```
        INSERT INTO biglietto(prezzo, rimborsato, metodopagamento, id_pagamento,
account, spostamento_percorso, posto, carrozza, email, treno, data_acquisto,
partenza_ordine, arrivo_ordine, data_viaggio)
        VALUES ((metriTot/1000)*costoKM, false, 'Mastercard', '...', accountUt,
spostPercorsoNew, numeroPosto, numeroCarrozza, emailNew, trenoSpost,
current_date, ordinePartenza, ordineArrivo, spostgiornonew)
        RETURNING identificativo INTO valore;

        RETURN QUERY
        SELECT biglietto.identificativo, persona.nome, persona.cognome,
biglietto.treno, biglietto.carrozza, biglietto.posto, percorso.partenza,
percorso.ora_partenza, percorso.arrivo, percorso.ora_arrivo,
biglietto.data_viaggio, biglietto.prezzo
        FROM biglietto, percorso, persona
        WHERE biglietto.identificativo = valore
        AND percorso.identificativo = spostpercorsonew
        AND persona.email = emailnew;
    end;
$$;
```

Visualizzazione Viaggi

```
CREATE FUNCTION cerca_treno(partenzaf character varying, arrivof character
varying, classe character varying)
    RETURNS TABLE(percorso integer, giorno date, ora_partenza time without time
zone, ora_arrivo time without time zone, costo numeric)
    LANGUAGE plpgsql
AS
$$
DECLARE
    BEGIN
    RETURN QUERY
    SELECT spostamento.percorso, spostamento.giorno,
entrata.entrata_programmata, uscita.uscita_programmata,
(sum(tratta.lunghezza)/1000)*(SELECT prezzo

FROM classe_viaggio

WHERE classe_viaggio.nome = classe) as costo
    FROM spostamento, percorso p0, sezione viaggio, sezione entrata, sezione
uscita, tratta
    WHERE spostamento.percorso = p0.identificativo
    AND spostamento.percorso = p0.identificativo
    AND viaggio.percorso = p0.identificativo
    AND viaggio.tratta_fine = tratta.fine
    AND viaggio.tratta_inizio = tratta.inizio
    AND viaggio.tratta_limit = tratta.limite_velocità
    AND (entrata.percorso,entrata.ordine) IN (SELECT s1.percorso, s1.ordine
         FROM percorso p1, sezione s1
         WHERE p1.identificativo = p0.identificativo
         AND s1.percorso = p0.identificativo
         AND s1.tratta_inizio = partenzaf
         AND (s1.ordine = 0 OR (SELECT s2."fermata?"
                                FROM sezione s2
                                WHERE s2.ordine = s1.ordine - 1
                                AND s2.percorso = p0.identificativo)))
    AND (uscita.percorso,uscita.ordine) IN (SELECT s3.percorso, s3.ordine
         FROM percorso p3, sezione s3
         WHERE p3.identificativo = p0.identificativo
         AND s3.percorso = p0.identificativo
         AND s3.tratta_fine = arrivof
         AND s3."fermata?" = true)
    AND viaggio.ordine >= entrata.ordine
```

```
    AND viaggio.ordine <= uscita.ordine
    GROUP BY spostamento.percorso, spostamento.giorno,
entrata.entrata_programmata, uscita.uscita_programmata;
    ORDER BY spostamento.giorno, entrata.entrata_programmata;
END;
$$;
```

Inserimento Percorso

```
INSERT INTO percorso (identificativo, ora_arrivo, ora_partenza, partenza,
arrivo, tipologia)
VALUES ('...', '...', '...', '...', '...', '...');
```

Inserimento Spostamento

```
INSERT INTO spostamento (percorso, treno, giorno, idgruppo)
VALUES ('...', '...', '...', '...');
```

Nuovo Contratto

```
INSERT INTO contratto (identificativo, tipo, inizio, rescisso, paga_oraria,
fine)
VALUES ('...', '...', '...', '...', '...', '...');
```

Inserimento Nuovo Impiegato

```
INSERT INTO impiegato (cf, email, email_aziendale, data_nascita, iban, sesso,
tipologia)
VALUES ('...', '...', '...', '...', '...', '...', '...');
```

Calcolo Ritardo medio

```
CREATE FUNCTION calcolo_ritardo(giornata character varying)
    RETURNS TABLE(avg interval)
    LANGUAGE plpgsql
AS
$$
BEGIN
        return QUERY
        SELECT avg(sez_eff.uscita - sez.uscita_programmata)
        FROM sezione sez, sezione_effettiva sez_eff
        WHERE sez_eff.giorno = giornata::DATE AND (sez.percorso=sez_eff.percorso
AND sez.ordine = sez_eff.sezione_ordine);
    end;
$$;
```

Calcolo Guadagno

```
create function guadagno_giornata() returns bigint
    language plpgsql
as
$$
DECLARE
        somma int8;
    BEGIN
        SELECT SUM(prezzo) INTO somma
        from biglietto
        where biglietto.data_acquisto::date = current_date AND NOT
biglietto.rimborsato;

        RETURN somma;
    end;
$$;
```

Inserimento Manutenzione

```
INSERT INTO manutenzione (descrizione, idtreno, fine, inizio, banco,
descrizione_guasto)
VALUES ('...', '...', '...', '...', '...', '...');
```

Inserimento Personale di Bordo

```
create function inserimento_personale_bordo(capotrenonew codice_fiscale,
macchinistanew codice_fiscale, assistenti codice_fiscale[]) returns void
    language plpgsql
as
$$
DECLARE
    valore integer;
    cf codice_fiscale;
BEGIN
        INSERT INTO personale_bordo (capotreno, macchinista) VALUES
(capotrenoNew, macchinistaNew);
        SELECT MAX(identificativo) into valore from personale_bordo;
        foreach cf in array assistenti
            LOOP
            INSERT INTO servizio_assistente (assistente, idgruppo) values (cf,
valore);
            end loop;
    END
$$;
```

Visualizza Arrivi in Stazione

```
CREATE FUNCTION visualizza_arrivi(stazione character varying)
    RETURNS TABLE("ID Treno" smallint, "ID Percorso" integer, "Provenienza"
character varying, "Giorno Arrivo" date, "Ora Arrivo" time without time zone)
    LANGUAGE plpgsql
AS
$$
BEGIN
    RETURN QUERY
    SELECT t1.identificativo, p1.identificativo, p1.partenza, s1.giorno,
sez1.uscita_programmata, sez1.binario
    FROM treno t1, percorso p1, spostamento s1, sezione sez1
    WHERE s1.treno = t1.identificativo
    AND s1.percorso = p1.identificativo
    AND sez1.percorso = p1.identificativo
    AND sez1.tratta_fine = stazione
    AND sez1."fermata?"
    AND (s1.giorno > current_date OR (s1.giorno = current_date AND
sez1.uscita_programmata > current_time))
   ORDER BY s1.giorno, uscita_programmata ASC;
END

$$;
```

Visualizza Partenze in Stazione

```
CREATE FUNCTION visualizza_partenze(stazione character varying)
    RETURNS TABLE("ID Treno" smallint, "ID Percorso" integer, "Destinazione"
character varying, "Giorno Partenza" date, "Ora Partenza" time without time
zone)
    LANGUAGE plpgsql
AS
$$
BEGIN
    RETURN QUERY
  SELECT t1.identificativo, p1.identificativo, p1.arrivo,  s1.giorno,
sez1.entrata_programmata, binario.binario
FROM treno t1, percorso p1, spostamento s1, sezione sez1, sezione binario
WHERE s1.treno = t1.identificativo
AND s1.percorso = p1.identificativo
AND sez1.percorso = p1.identificativo
```

```
AND sez1.tratta_inizio = stazione
AND (sez1.ordine = 0 OR (SELECT "fermata?"
  FROM sezione
  WHERE sezione.percorso = sez1.percorso
  AND sezione.ordine = sez1.ordine - 1))
--Comando per trovare il binario (visto che si riferisce alla sezione
precendente (che potrebbe anche essere in un altro
--spostamento del tutto)
AND (binario.ordine, binario.percorso) IN (SELECT sez2.ordine, sez2.percorso
              FROM sezione sez2, percorso p2, spostamento s2
              WHERE
                 (   sez2.percorso = sez1.percorso
                 AND sez1.ordine > 0
                 AND sez2.tratta_fine = stazione)
              OR
                 (   sez1.ordine = 0
                 AND p2.identificativo = (SELECT p3.identificativo
                          FROM percorso p3, spostamento s3
                          WHERE s3.treno = t1.identificativo
                          AND p3.identificativo = s3.percorso
                          AND p3.ora_arrivo < p1.ora_partenza
                          AND p3.ora_arrivo >= ALL (SELECT p4.ora_arrivo
                                       FROM percorso p4, spostamento s4
                                       WHERE p4.identificativo = s4.percorso
                                       AND s4.treno = t1.identificativo
                                       AND p4.ora_arrivo < p1.ora_partenza))
                                    AND sez2.ordine = (SELECT max(sez3.ordine)
                                    FROM sezione sez3
                                    WHERE sez3.percorso = sez2.percorso))))
AND (s1.giorno > current_date OR (s1.giorno = current_date AND
sez1.entrata_programmata > current_time))
ORDER BY s1.giorno, sez1.entrata_programmata ASC;
END

$$;
```

Calcolo Retribuzione di un Impiegato

```
create function calcolo_retribuzione_impiegato(cfnew codice_fiscale,
inizio_calcolo date) returns bigint
    language plpgsql
as
$$
DECLARE
        somma int8;
    BEGIN
    SELECT SUM(extract(HOUR from (t.fine-t.inizio))) * (SELECT c.paga_oraria
                                    FROM contratto c,
                                    WHERE cfnew=c.impiegato
                                    AND c.rescisso = false
                                    AND c.fine >= Inizio_Calcolo
                                    AND c.inizio <= Inizio_Calcolo) INTO somma
    from turno t, contratto c
    where extract(MONTH FROM t.giorno) = extract(MONTH FROM Inizio_Calcolo)
    AND t.impiegato = c.impiegato
    AND t.giorno <= Inizio_Calcolo
    AND t.giorno >= c.inizio
    AND t.giorno <= c.fine
    AND t.impiegato = cfnew;
    RETURN somma;
    end;
$$;
```

Query

-Visualizza in ordine temporale tutti i treni che vanno da un punto a ad un punto b e il loro relativi costi data una classe di viaggio

```
SELECT spostamento.percorso, spostamento.giorno, entrata.entrata_programmata,
uscita.uscita_programmata,
sum((tratta.lunghezza)/1000)*(SELECT prezzo
                              FROM classe_viaggio
                              WHERE classe_viaggio.nome = ClasseViaggio) as costo

FROM spostamento, percorso p0, sezione viaggio, sezione entrata, sezione uscita,
tratta
WHERE spostamento.percorso = p0.identificativo
AND spostamento.percorso = p0.identificativo
AND viaggio.percorso = p0.identificativo
AND viaggio.tratta_fine = tratta.fine
AND viaggio.tratta_inizio = tratta.inizio
AND viaggio.tratta_limit = tratta.limite_velocità
AND (entrata.percorso,entrata.ordine) IN (SELECT s1.percorso, s1.ordine
     FROM percorso p1, sezione s1
     WHERE p1.identificativo = p0.identificativo
     AND s1.percorso = p0.identificativo
     AND s1.tratta_inizio = partenzaA
     AND (s1.ordine = 0 OR (SELECT s2."fermata?"
                            FROM sezione s2
                            WHERE s2.ordine = s1.ordine - 1
                            AND s2.percorso = p0.identificativo)))
AND (uscita.percorso,uscita.ordine) IN (SELECT s3.percorso, s3.ordine
     FROM percorso p3, sezione s3
     WHERE p3.identificativo = p0.identificativo
     AND s3.percorso = p0.identificativo
     AND s3.tratta_fine = arrivoB
     AND s3."fermata?" = true)
AND viaggio.ordine >= entrata.ordine
AND viaggio.ordine <= uscita.ordine
GROUP BY spostamento.percorso, spostamento.giorno, entrata.entrata_programmata,
uscita.uscita_programmata
ORDER BY spostamento.giorno, entrata.entrata_programmata
```

-Visualizza il numero della carrozza con valore intero minore che ha ancora posti disponibili per un dato viaggio

```
SELECT MIN(c.numero) INTO numeroCarrozza
        FROM carrozza c
        WHERE c.classe_viaggio= ClasseViaggio
        AND c.idtreno = IdentificativoTreno
        AND (SELECT count(*)
             FROM posto
             WHERE posto.idtreno = c.idtreno
      AND c.numero = posto.carrozza) != (SELECT count(DISTINCT biglietto.posto)
               FROM biglietto, sezione partenza, sezione arrivo
               WHERE biglietto.spostamento_percorso = IdentificativoPercorso
               AND biglietto.data_viaggio = DataViaggio
               AND biglietto.carrozza = c.numero
               AND partenza.percorso = biglietto.spostamento_percorso
               AND arrivo.percorso = biglietto.spostamento_percorso
               AND partenza.ordine = biglietto.partenza_ordine
               AND arrivo.ordine = biglietto.arrivo_ordine
     AND ((arrivo.ordine >= ordinePartenza AND partenza.ordine <= ordineArrivo)
     OR (partenza.ordine <= ordineArrivo AND arrivo.ordine >= ordinePartenza)))
```

-Visualizza il numero del posto con valore intero minore libero per un dato viaggio

```
SELECT MIN(p.numero) INTO numeroPosto
                from posto p
                where p.numero NOT IN (SELECT DISTINCT biglietto.posto
                FROM biglietto, sezione partenza, sezione arrivo
                WHERE biglietto.spostamento_percorso = IdentificativoPercorso
                AND biglietto.data_viaggio = DataViaggio
                AND biglietto.carrozza = NumeroCarrozza
                AND partenza.percorso = biglietto.spostamento_percorso
                AND arrivo.percorso = biglietto.spostamento_percorso
                AND partenza.ordine = biglietto.partenza_ordine
                AND arrivo.ordine = biglietto.arrivo_ordine
     AND ((arrivo.ordine >= ordinePartenza AND partenza.ordine <= ordineArrivo)
     OR (partenza.ordine <= ordineArrivo AND arrivo.ordine >= ordinePartenza)))
                        AND p.idtreno = IdentificativoTreno
                        AND p.carrozza = NumeroCarrozza;
```

-Generalità dei segretari che hanno risposto ad una assistenza clienti relativa al biglietto più costoso venduto in quella giornata

```
SELECT impiegato.*, persona.*
from persona, impiegato
WHERE persona.email = impiegato.email
AND impiegato.cf IN (SELECT impiegato
         FROM segretario, contenuto_assistenza, assistenza_clienti
         WHERE contenuto_assistenza.segretario = segretario.impiegato
         AND assistenza_clienti.identificativo = contenuto_assistenza.assistenza
         AND assistenza_clienti.biglietto IN (SELECT b1.identificativo
                        FROM biglietto b1
                        WHERE b1.identificativo = assistenza_clienti.biglietto
                        AND b1.prezzo >= ALL (SELECT b2.prezzo
                                        FROM biglietto b2
                                        WHERE b2.data_acquisto = b1.data_acquisto)))
```

Con View

```
CREATE VIEW Biglietti_Costosi AS
    SELECT b1.identificativo, b1.data_acquisto
    FROM biglietto b1
    WHERE b1.prezzo >= ALL (SELECT b2.prezzo
                            FROM biglietto B2
                            WHERE b1.data_acquisto = b2.data_acquisto);

SELECT impiegato.*, persona.*
from persona, impiegato
WHERE persona.email = impiegato.email
AND impiegato.cf IN (SELECT impiegato
         FROM segretario, contenuto_assistenza, assistenza_clienti
         WHERE contenuto_assistenza.segretario = segretario.impiegato
         AND assistenza_clienti.identificativo = contenuto_assistenza.assistenza
         AND assistenza_clienti.biglietto IN (SELECT identificativo FROM
Biglietti_Costosi));
```

-Biglietto più costoso acquistato da ogni account

```
SELECT a1.codice_fedeltà, b1.identificativo, b1.prezzo
FROM account a1, biglietto b1
WHERE b1.account = a1.codice_fedeltà
AND b1.prezzo >= all (SELECT b2.prezzo
                        FROM biglietto b2
                        WHERE b2.account = a1.codice_fedeltà)
```

-Posizione finale del treno nel futuro dopo aver fatto tutti gli spostamenti programmati

```
SELECT p0.arrivo, p0.ora_arrivo, s0.giorno
FROM spostamento s0, percorso p0
WHERE s0.percorso = p0.identificativo
```

```
AND s0.treno = new.treno
AND s0.giorno = (SELECT max(s1.giorno)
                 FROM spostamento s1
                 WHERE s1.treno = new.treno)
AND p0.ora_arrivo >= all (SELECT ora_partenza
                                FROM percorso p1, spostamento s1
                                WHERE s1.percorso = p1.identificativo
                                AND s1.treno = new.treno
                                AND s1.giorno = (SELECT MAX(s2.giorno)
                                                 FROM spostamento s2
                                                 WHERE s2.treno = new.treno));
```

- Treni in partenza in una stazione, il loro orario di partenza e la loro destinazione

```
SELECT t1.identificativo, p1.identificativo, p1.arrivo,  s1.giorno,
sez1.entrata_programmata, binario.binario
FROM treno t1, percorso p1, spostamento s1, sezione sez1, sezione binario
WHERE s1.treno = t1.identificativo
AND s1.percorso = p1.identificativo
AND sez1.percorso = p1.identificativo
AND sez1.tratta_inizio = 'Pescara'
AND (sez1.ordine = 0 OR (SELECT "fermata?"
  FROM sezione
  WHERE sezione.percorso = sez1.percorso
  AND sezione.ordine = sez1.ordine - 1))
--Comando per trovare il binario (visto che si riferisce alla sezione
precendente (che potrebbe anche essere in un altro
--spostamento del tutto)
AND (binario.ordine, binario.percorso) IN (SELECT sez2.ordine, sez2.percorso
              FROM sezione sez2, percorso p2, spostamento s2
              WHERE
                 (   sez2.percorso = sez1.percorso
                 AND sez1.ordine > 0
                 AND sez2.tratta_fine = 'Pescara')
              OR
                 (   sez1.ordine = 0
                 AND p2.identificativo = (SELECT p3.identificativo
                         FROM percorso p3, spostamento s3
                         WHERE s3.treno = t1.identificativo
                         AND p3.identificativo = s3.percorso
                         AND p3.ora_arrivo < p1.ora_partenza
                         AND p3.ora_arrivo >= ALL (SELECT p4.ora_arrivo
                                         FROM percorso p4, spostamento s4
                                         WHERE p4.identificativo = s4.percorso
                                         AND s4.treno = t1.identificativo
                                         AND p4.ora_arrivo < p1.ora_partenza))
                                    AND sez2.ordine = (SELECT max(sez3.ordine)
                                    FROM sezione sez3
                                    WHERE sez3.percorso = sez2.percorso))))
AND (s1.giorno > current_date OR (s1.giorno = current_date AND
sez1.entrata_programmata > current_time))
ORDER BY s1.giorno, sez1.entrata_programmata ASC;
```

-Treni in arrivo in una stazione, il loro orario di arrivo e la loro provenienza

```
SELECT t1.identificativo, p1.identificativo, p1.partenza, s1.giorno,
sez1.uscita_programmata, sez1.binario
    FROM treno t1, percorso p1, spostamento s1, sezione sez1
    WHERE s1.treno = t1.identificativo
    AND s1.percorso = p1.identificativo
    AND sez1.percorso = p1.identificativo
    AND sez1.tratta_fine = stazione
    AND sez1."fermata?"
    AND (s1.giorno > current_date OR (s1.giorno = current_date AND
```

```
sez1.uscita_programmata > current_time))
   ORDER BY s1.giorno, uscita_programmata ASC;
```

-Treni che arrivano a Roma provenienti da Milano che hanno più di 300 passeggeri

```
SELECT treno.identificativo, treno.modello_treno, spostamento.percorso,
spostamento.giorno
FROM treno, spostamento, percorso
WHERE spostamento.treno = treno.identificativo
AND spostamento.percorso = percorso.identificativo
AND (SELECT count(*)
     FROM biglietto
     WHERE biglietto.data_viaggio = spostamento.giorno
     AND biglietto.spostamento_percorso = spostamento.percorso) > 300
AND (spostamento.percorso, spostamento.giorno) IN (SELECT s1.percorso, s1.giorno
                                                   FROM spostamento s1, percorso
p1
                                                   WHERE s1.percorso =
p1.identificativo
                                                   AND p1.partenza = 'Milano'
                                                   AND p1.arrivo = 'Roma')
```

Con view

```
CREATE VIEW Spostamenti_Popolati AS
    SELECT biglietto.spostamento_percorso, biglietto.data_viaggio
    FROM biglietto
    GROUP BY biglietto.spostamento_percorso
    HAVING count(*) > 300;

SELECT treno.identificativo, treno.modello_treno, spostamento.percorso,
spostamento.giorno
FROM treno, spostamento, percorso, Spostamenti_Popolati
WHERE spostamento.treno = treno.identificativo
AND spostamento.percorso = percorso.identificativo
AND spostamento.giorno = Spostamenti_Popolati.data_viaggio
AND spostamento.percorso = Spostamenti_Popolati.spostamento_percorso
AND (spostamento.percorso, spostamento.giorno) IN (SELECT s1.percorso, s1.giorno
                                                   FROM spostamento s1, percorso p1
                                                   WHERE s1.percorso = p1.identificativo
                                                   AND p1.partenza = 'Milano'
                                                   AND p1.arrivo = 'Roma')
```

-Dati degli impiegati che lavorano in un dato spostamento

```
SELECT *
FROM impiegato
WHERE cf IN (SELECT ferroviere.impiegato
             FROM ferroviere, spostamento, personale_bordo
             WHERE 'percorso'= spostamento.percorso
             AND 'giorno' = spostamento.giorno
             AND spostamento.idgruppo = personale_bordo.identificativo
             AND (capotreno = ferroviere.impiegato OR macchinista =
ferroviere.impiegato))
OR cf IN     (SELECT assistente.impiegato
             FROM assistente, spostamento, servizio_assistente
             WHERE 'percorso'= spostamento.percorso
             AND 'giorno' = spostamento.giorno
             AND spostamento.idgruppo = servizio_assistente.idgruppo
             AND servizio_assistente.assistente = assistente.impiegato)
```

-Totale da pagare a partire da un giorno a scelta fino a fine mese per un impiegato (fino alla fine del contratto attuale nel caso finisca nel mezzo del mese)

```
SELECT SUM(extract(HOUR from (t.fine-t.inizio))) * (SELECT c.paga_oraria
                                    FROM contratto c
```

```
                                    WHERE CodiceFiscale=c.impiegato
                                    AND c.rescisso = false
                                    AND c.fine >= Inizio_Calcolo
                                    AND c.inizio <= Inizio_Calcolo) INTO somma
    from turno t, contratto c
    where extract(MONTH FROM t.giorno) = extract(MONTH FROM Inizio_Calcolo)
    AND t.impiegato = c.impiegato
    AND t.giorno <= InizioCalcolo
    AND t.giorno >= c.inizio
    AND t.giorno <= c.fine
    AND t.impiegato = CodiceFiscale;
```

Con view (leggermente diversa, invece di specificare un giorno a scelta, sceglie il giorno attuale. Utile nel caso si voglia eseguire il calcolo automaticamente ogni ultimo del mese)

```
CREATE VIEW paga_attuale AS
SELECT contratto.impiegato, contratto.paga_oraria
FROM contratto
WHERE rescisso = false
AND contratto.fine >= current_date
AND contratto.inizio <= current_date;

SELECT SUM(extract(HOUR from (t.fine-t.inizio))) * (SELECT p.paga_oraria
                                    FROM paga_attuale p
                                    WHERE p.impiegato = CodiceFiscale) INTO
somma
    from turno t, contratto c
    where extract(MONTH FROM t.giorno) = extract(MONTH FROM current_date)
    AND t.impiegato = c.impiegato
    AND t.giorno <= current_date
    AND t.giorno >= c.inizio
    AND t.giorno <= c.fine
    AND t.impiegato = CodiceFiscale;
```

-Lista degli impiegati con contratto valido che non hanno un turno impostato nel futuro

```
SELECT impiegato.*
FROM impiegato
WHERE cf in (SELECT contratto.impiegato
             FROM contratto
             WHERE contratto.inizio < current_date
             AND contratto.fine > current_date)
AND cf not in (SELECT turno.impiegato
           FROM turno
           WHERE turno.giorno > current_date)
```

Con view

```
CREATE VIEW impiegato_corrente AS
    SELECT impiegato.*
    FROM impiegato
    WHERE cf in (SELECT contratto.impiegato
             FROM contratto
             WHERE contratto.inizio < current_date
             AND contratto.fine > current_date);

SELECT impiegato_corrente.*
FROM impiegato_corrente
WHERE impiegato_corrente.cf not in (SELECT turno.impiegato
                                    FROM turno
                                    WHERE turno.giorno > current_date)
```

-Dati personali e lavorativi delle persone che posseggono un account di livello Platino che sono anche operai che hanno effettuato più di 10 manutenzioni

```
SELECT persona.*, impiegato.*
FROM persona, account, impiegato, operaio o
WHERE persona.email = impiegato.email
AND o.impiegato = impiegato.cf
AND account.email = persona.email
AND account.livello = 'Platinum'
AND (SELECT count(*)
     FROM riparatori
     WHERE riparatori.operaio = o.impiegato) > 10;
```

Con view

```
CREATE VIEW persona_impiegato as
SELECT persona.*, impiegato.*
FROM persona, impiegato
WHERE persona.email = impiegato.email;

CREATE VIEW account_platino AS
SELECT account.email
FROM account
WHERE account.livello = 'Platinum';

SELECT persona_impiegato.*
FROM persona_impiegato, operaio o, account_platino
WHERE persona_impiegato.email = account_platino.email
AND o.impiegato = persona_impiegato.cf
AND (SELECT count(*)
     FROM riparatori
     WHERE riparatori.operaio = o.impiegato) > 10;
```

- La manutenzione, il treno e il numero di operai alla quale hanno lavorato più operai per ogni treno che ha avuto almeno una manutenzione

```
SELECT m1.identificativo, m1.idtreno, count(riparatori.operaio)
FROM riparatori, manutenzione m1
WHERE riparatori.manutenzione = m1.identificativo
GROUP BY riparatori.manutenzione, m1.idtreno
HAVING count(riparatori.operaio) >= ALL (SELECT count(r2.operaio)
                                         FROM riparatori r2, manutenzione m2
                                         WHERE r2.manutenzione = m2.identificativo
                                         AND m2.idtreno = m1.idtreno);
```

Con view

```
CREATE VIEW numero_operai_manutenzione AS
    SELECT count(riparatori.operaio) as Numero_Operai, m.identificativo
    FROM riparatori, manutenzione m
    WHERE riparatori.manutenzione = m.identificativo
    GROUP BY m.identificativo;

SELECT manutenzione.identificativo, manutenzione.idtreno,
numero_operai_manutenzione.Numero_Operai
FROM numero_operai_manutenzione, manutenzione
WHERE numero_operai_manutenzione.identificativo = manutenzione.identificativo
AND numero_operai_manutenzione.Numero_Operai >= all (SELECT n2.Numero_Operai
                              FROM numero_operai_manutenzione n2, manutenzione m2
                              WHERE n2.identificativo = m2.identificativo
                              AND m2.idtreno = manutenzione.idtreno)
```

-Coppie di capotreni che hanno un contratto valido e hanno effettuato gli stessi percorsi

```
SELECT f1.impiegato, f2.impiegato
FROM ferroviere f1, ferroviere f2, contratto c1, contratto c2
```

```
WHERE f1.impiegato = c1.impiegato
AND c1.inizio > current_date
AND c1.fine < current_date
AND NOT c1.rescisso
AND f1.ruolo = 'Capotreno'
AND f2.impiegato = c2.impiegato
AND c2.inizio > current_date
AND c2.fine < current_date
AND NOT c2.rescisso
AND f2.ruolo = 'Capotreno'
AND NOT EXISTS(
    (SELECT percorso.identificativo
     FROM percorso, spostamento, personale_bordo
     WHERE spostamento.percorso = percorso.identificativo
     AND spostamento.idgruppo = personale_bordo.identificativo
     AND capotreno = f1.impiegato) EXCEPT (SELECT percorso.identificativo
                        FROM percorso, spostamento, personale_bordo
                        WHERE spostamento.percorso = percorso.identificativo
                        AND spostamento.idgruppo = personale_bordo.identificativo
                        AND capotreno = f2.impiegato))
AND NOT EXISTS(
    (SELECT percorso.identificativo
     FROM percorso, spostamento, personale_bordo
     WHERE spostamento.percorso = percorso.identificativo
     AND spostamento.idgruppo = personale_bordo.identificativo
     AND capotreno = f2.impiegato) EXCEPT (SELECT percorso.identificativo
                        FROM percorso, spostamento, personale_bordo
                        WHERE spostamento.percorso = percorso.identificativo
                        AND spostamento.idgruppo = personale_bordo.identificativo
                        AND capotreno = f1.impiegato));
```

Con view

```
SELECT f1.impiegato, f2.impiegato
FROM capotreni_attuali f1, capotreni_attuali f2
WHERE NOT EXISTS(
    (SELECT percorso.identificativo
     FROM percorso, spostamento, personale_bordo
     WHERE spostamento.percorso = percorso.identificativo
     AND spostamento.idgruppo = personale_bordo.identificativo
     AND capotreno = f1.impiegato) EXCEPT (SELECT percorso.identificativo
                        FROM percorso, spostamento, personale_bordo
                        WHERE spostamento.percorso = percorso.identificativo
                        AND spostamento.idgruppo = personale_bordo.identificativo
                        AND capotreno = f2.impiegato))
AND NOT EXISTS(
    (SELECT percorso.identificativo
     FROM percorso, spostamento, personale_bordo
     WHERE spostamento.percorso = percorso.identificativo
     AND spostamento.idgruppo = personale_bordo.identificativo
     AND capotreno = f2.impiegato) EXCEPT (SELECT percorso.identificativo
                       FROM percorso, spostamento, personale_bordo
                       WHERE spostamento.percorso = percorso.identificativo
                       AND spostamento.idgruppo = personale_bordo.identificativo
                       AND capotreno = f1.impiegato))
```