Basi di Dati

Progetto A.A. 2023/2024

TITOLO DEL PROGETTO

Matricola

0307070

Nome e Cognome

Luca Cupellaro

Indice

1. Descrizione del Minimondo

Sistema di gestione di trasporto ferroviario

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

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

I gestori del servizio associano anche un macchinista e un capotreno a ciascun treno. La gestione dei turni avviene da parte dei gestori del servizio su base mensile. Ogni lavoratore ha la possibilità di generare un report sui propri turni di lavoro, su base settimanale, riportante l'indicazione degli orari e dei treni in cui esso è coinvolto.

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

2. 25Analisi dei Requisiti

Identificazione dei termini ambigui e correzioni possibili

Linea	Termine	Nuovo termine	Motivo correzione
4,12	associato	si vuole salvare	associato potrebbe apparire come una associazione
7	Sono memorizzati	Per ogni treno sono memorizzati.	Termine più chiaro.
4	veicolo	Treno.	Termine più chiaro.
3	servizio	Sistema.	Termine più chiaro.
8	carrozza	Vagone.	Termine più chiaro.
15	gestori	Gestori del servizio.	Termine più chiaro.

Specifica disambiguata

Sistema di gestione di trasporto ferroviario

Si vuole realizzare un sistema informativo per la gestione dell'operatività di un'azienda di trasporto pubblico ferroviario ad alta velocità per passeggeri. I treni gestiti dal sistema sono caratterizzati da una matricola (codice univoco numerico di quattro cifre). Di ogni treno si vuole salvare una data di acquisto e ad uno storico di manutenzione. È inoltre di interesse mantenere informazioni legate alla marca e modello delle locomotrici e vagoni. Di ciascun treno è di interesse conoscere anche da quanti vagoni questo è composto. Sono memorizzati il numero di vagoni di prima e seconda classe, il numero massimo di passeggeri che possono viaggiare in ciascun vagone (con riferimento ai singoli posti).

Ciascuna tratta ha un insieme di fermate identificate dal nome della stazione, dalla città e dalla provincia in cui si trova. Inoltre, per ciascuna tratta, vengono mantenuti i capilinea di partenza ed arrivo. Ciascuna fermata si salva l'orario di arrivo e di partenza previsti, eccezion fatta per i capilinea in cui uno solo dei due orari è disponibile. Ogni tratta viene coperta da un numero predefinito di treni, la cui associazione viene gestita dai gestori del servizio. I gestori del servizio possono, su base periodica, modificare il numero di treni operanti su ciascuna tratta.

I gestori del servizio associano anche un macchinista e un capotreno a ciascun treno. La gestione dei turni avviene da parte dei gestori del servizio su base mensile. Ogni lavoratore ha la possibilità di generare un report sui propri turni di lavoro, su base settimanale, riportante l'indicazione degli orari e dei treni in cui esso è coinvolto.

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

Glossario dei Termini

Termine	Descrizione	Sinonimi	Collegamenti
Treno	Mezzo di Trasporto pubblico utilizzato per coprire una determinate tratta.	veicolo	Vagoni, Locomotrici, Turni, Tratta
Tratta	Si riporta il punto di partenza e il punto di arrivo di ogni treno.	Viaggio	Prenotazione, Fermate
Fermata	Dove il treno si ferma	Stazione	Tratta
Acquirente	È colui che acquista un biglietto.	Utente, prenotato	Prenotazione
Prenotazione	Strumento utilizzato per accedere alla corsa del treno.	Biglietto.	Tratta
Vagoni	Struttura su ruote che trasporta passeggeri.	Carrozza.	Treno
Locomotrici	Utilizzata per trainare treni di vagoni o carri ferroviari.		Treno
Lavoratore	Personale coinvolto.	Personale	Treno

Raggruppamento dei requisiti in insiemi omogenei

Frasi relative al Treno
I treni gestiti dal servizio sono caratterizzati da una matricola (codice univoco numerico di quattro cifre). Ogni veicolo è anche associato ad una data di acquisto e ad uno storico di manutenzione. Di ciascun treno è di interesse conoscere anche da quanti vagoni questo è composto. Sono memorizzati il numero di carrozze di prima e seconda classe

Frasi relative alle Tratta
Per ciascuna tratta, vengono mantenuti i capilinea di partenza e di arrivo. Ogni tratta viene coperta da un numero predefinito di treni.

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

Frasi relative ai Vagoni
È inoltre di interesse mantenere informazioni legate alla marca e modello delle locomotrici e vagoni. Di ciascun treno è di interesse conoscere anche da quanti vagoni questo è composto. Di ogni vagone sono memorizzati il numero di carrozze di prima e seconda classe, il numero massimo di passeggeri che possono viaggiare in ciascun vagone (con riferimento ai singoli posti).

Frasi relative alle Locomotrici
È inoltre di interesse mantenere informazioni legate alla marca e modello delle locomotrici

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

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

Frasi relative agli Acquirenti
L'acquirente deve indicare il proprio nome, cognome data di nascita, codice fiscale e numero di carta di credito.

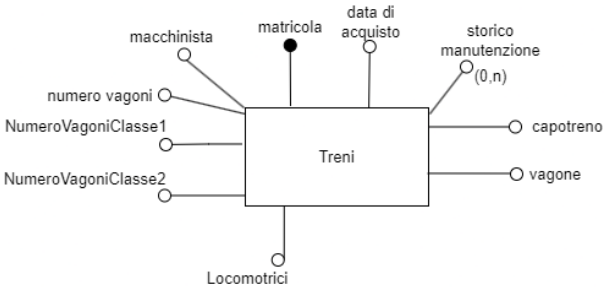
Frasi relative ai Lavoratori
I gestori del servizio associano anche un macchinista e un capotreno a ciascun treno. La gestione dei turni avviene da parte dei gestori del servizio su base mensile.

3. Progettazione concettuale

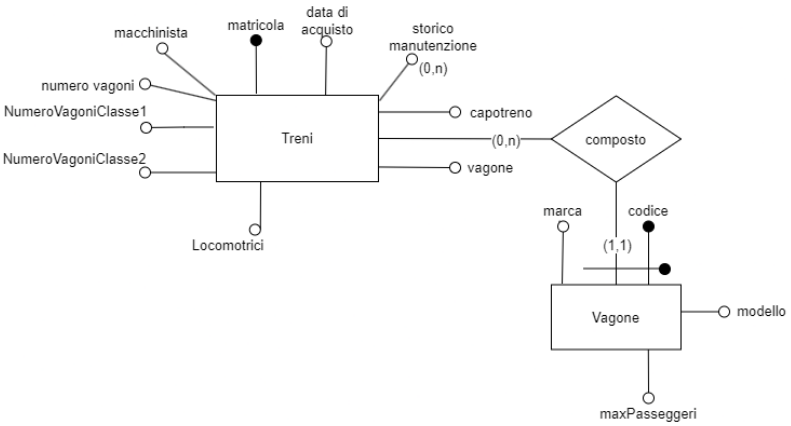
Costruzione dello schema E-R

Organizziamo la progettazione concettuale della nostra base dati adottando la strategia inside-out, che consiste nello sviluppare prima i concetti più importanti (primari) in bottom-up. Gli altri concetti secondari verranno successivamente sviluppati per collegamento ai concetti primari.

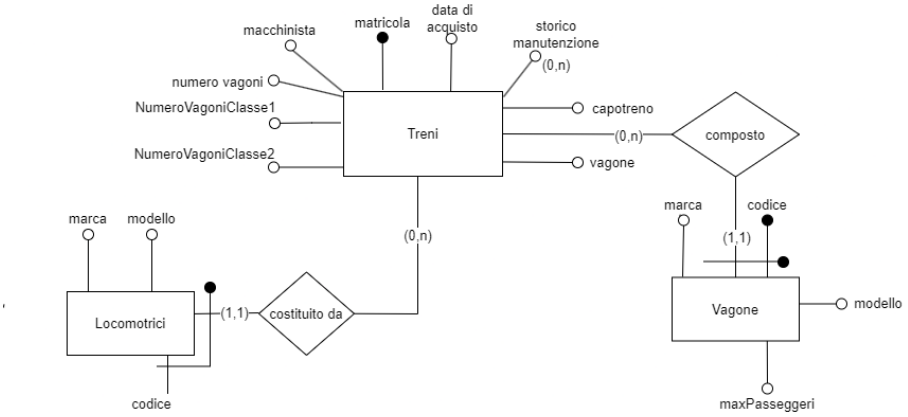
- Il primo concetto del mini-mondo di riferimento è racchiuso nell'entità Treni. Esso è identificato tramite l'attributo matricola. Di esso terremo traccia anche della data di acquisto, numero di vagoni totale da cui è composto, numero vagone di prima classe, di seconda classe e di un attributo multi-valore storico manutenzione. Raggruppando le specifiche che descrivono l'entità Treni si ottiene il seguente schema:



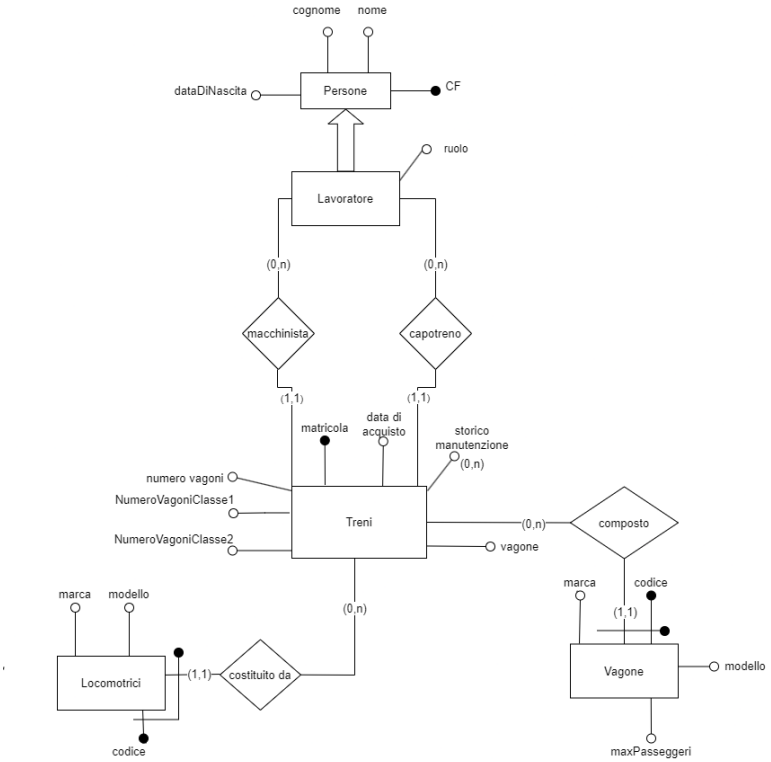
-Poiché è necessario mantenere diverse specifiche per i vagoni, come modello, marca e il numero massimo di passeggeri che possono contenere ogni vagone, applichiamo il pattern "Part of". Un treno può essere composto da 0 a n vagoni, e identifichiamo ciascun vagone tramite un codice univoco, poiché le sole informazioni di marca, modello e numero massimo di passeggeri non sono sufficienti a garantire l'unicità del vagone. La relazione "composto" è di tipo uno a molti, poiché un treno può essere associato a 0 o più vagoni, mentre un vagone deve essere associato a un solo treno.
Raggruppando le specifiche che descrivono le entità Vagoni e Treni si ottiene il seguente schema:



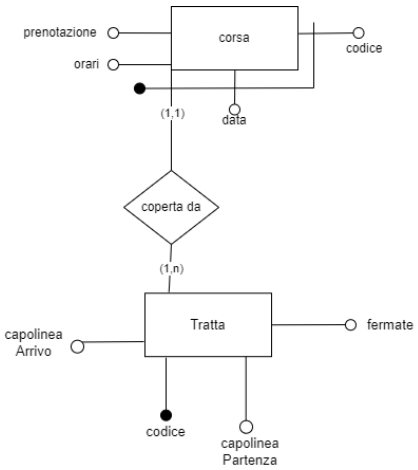
-Successivamente anche per Locomotrici applichiamo il pattern "Part of", in quanto si vogliono tenere traccia di varie specifiche come marca e modello. Viene identificata da un codice per lo stesso motivo dell'entità vagone. Anche in questo caso l'unità Locomotrice è legata tramite una relazione all'entità treno tramite usa associazione uno a molti poiché un treno può essere composta da 0 o n locomotrici e una locomotrice deve essere associata all'entità treno. Raggruppando le specifiche che descrivono l'entità Locomotrici e Treni si ottiene il seguente schema:



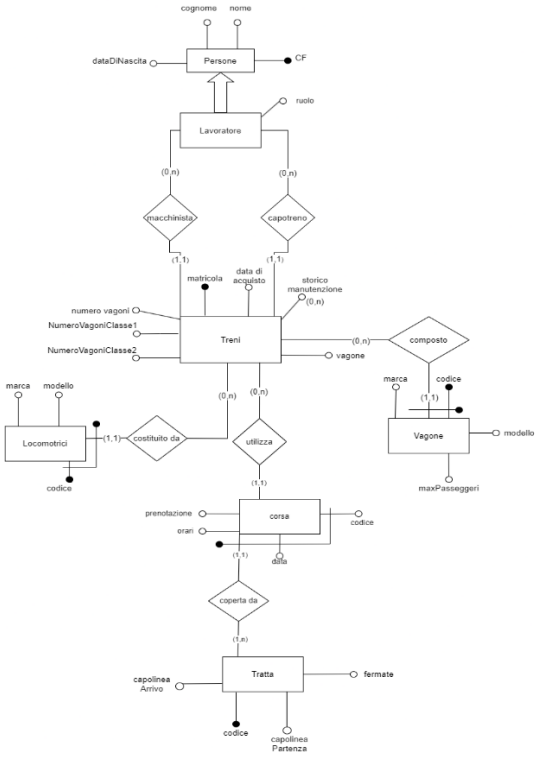
Poiché ciascun treno deve avere un macchinista e un capotreno, coi i relativi attributi. Reifico l'attributo macchinista e capotreno nell'entità lavoratore in quanto rappresenta un concetto importante per il nostro Mini-modo e lo associo tramite la relazione "macchinista" e "capotreno" a treno. Abbiamo una generalizzazione parziale, in cui esiste una relazione IS-A tra l'entità Persone, che rappresenta l'entità padre e lavoratore entità figlia. L'entità Persone include gli attributi data di nascita, cognome, nome ed è identificata tramite codice fiscale.



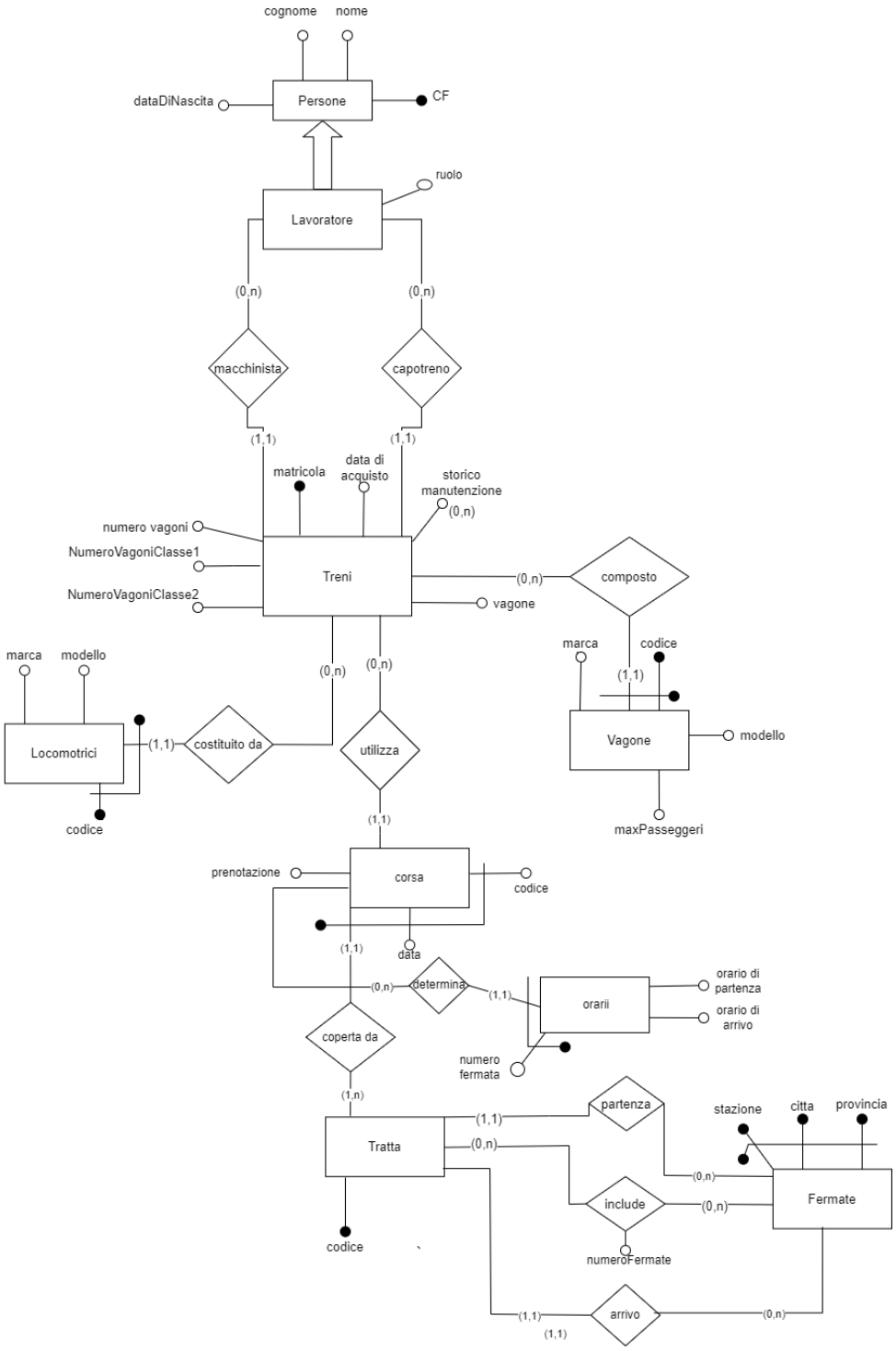
-Adesso introduciamo l'entità tratta di cui si salvano i campi capolinea di arrivo e di partenza, anche in questo caso non bastano come informazione in quanto potrebbero esserci più tratte che hanno stessi capolinea ma fanno diverse fermate, perciò, necessita di un codice. Con l'entità tratta si utilizza il pattern "instance of" con l'entità corsa. In quanto l'entità corsa sta a significare la corsa reale fatta dal treno. La corsa ha come attributi la data in cui viene fatta, l'orario e la prenotazione. Viene identificata dal codice, dalla data e dalla tratta che compie.



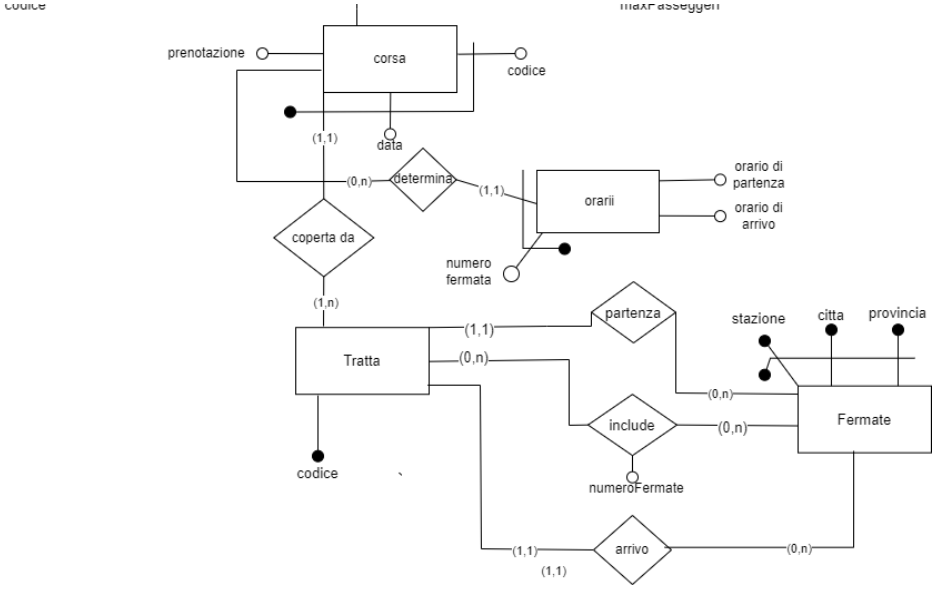
La corsa si relazione con l'entità treni tramite una associazione uno a molti in quanto un treno può coprire 0, n corse ma ogni tratta deve essere coperta da un solo treno. Raggruppando le specifiche che descrivono l'entità Tratta e Treni si ottiene il seguente schema:



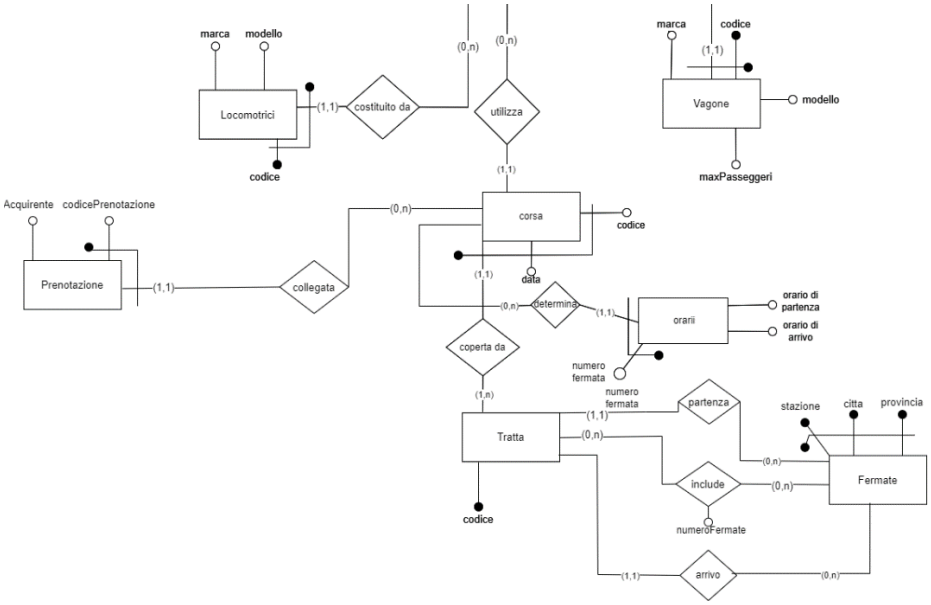
Successivamente si reifica l'attributo fermata contenuto in tratta e si crea l'entità fermata la quale viene identificata dalla tripletta di attributi fermate, città e provincia come viene descritto dal nostro Minimondo di riferimento. Tra tratte e fermate c'è una associazione molti a molti in quanto una tratta è composta da 0, n fermate e una fermata può essere compresa in 0, n tratte. Aggiungo l'associazione partenza e arrivo per tenere traccia dei capolinea di partenza e arrivo per ogni tratta, nella relazione include inserisco come attributo il numero della fermata che mi tiene traccia dell'ordinamento numerato delle fermate che vengono fatte per ogni tratta.



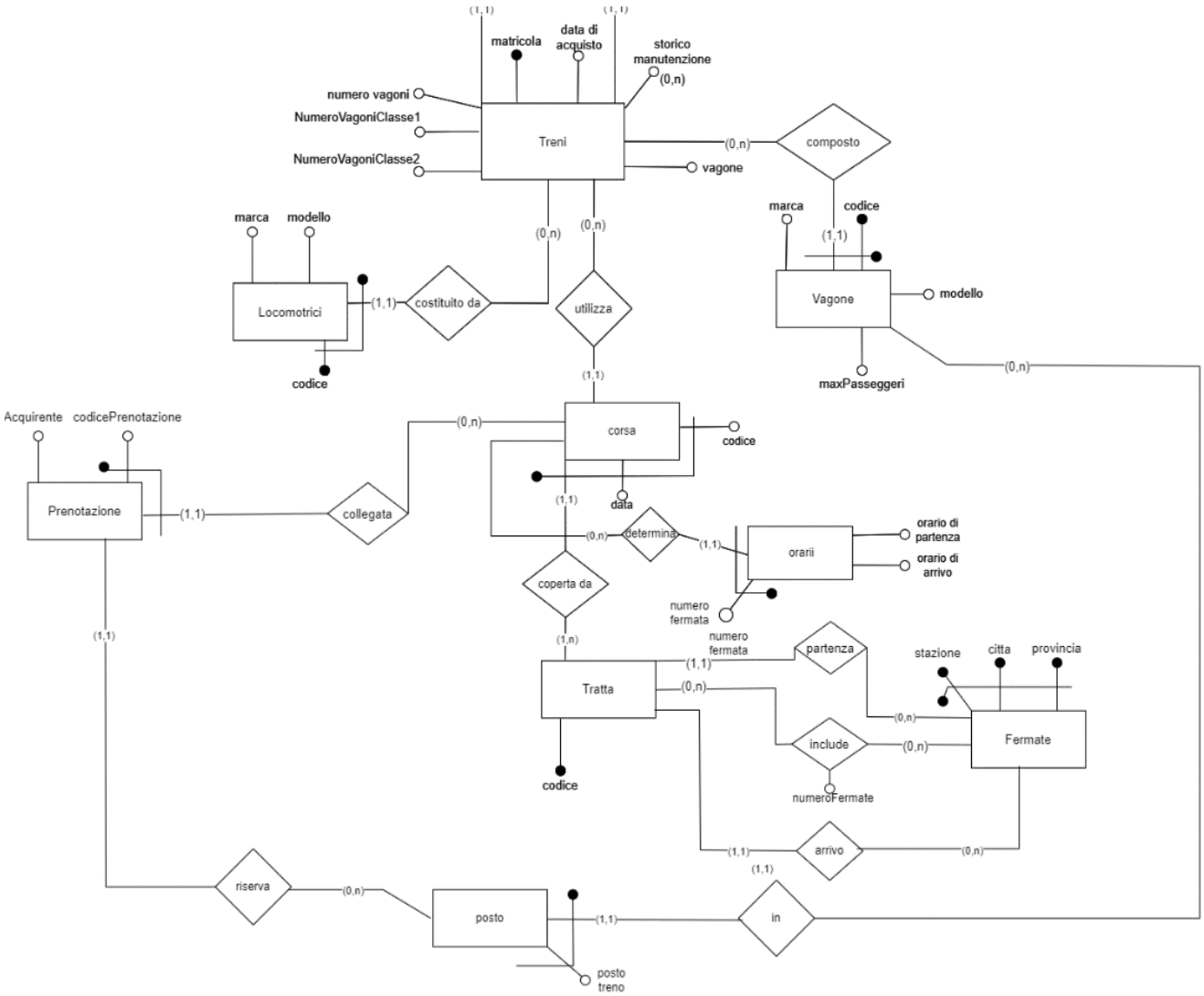
-Reifico l'attributo orari contenuto in corsa e creo una entità orari, la quale mi tiene traccia dell'orario di arrivo, partenza e del numero della fermata che viene fatto dalla corsa in base alla tratta, viene identificato tramite il numero della fermata e la relazione che si ha con l'entità corsa. Questa scelta di progettazione è dovuta al fatto che possono esserci più corse durante lo stesso giorno della stessa tratta, gli orari cambiano ogni volta a seconda della corsa. Con il vincolo che il numero fermata nella tabella orari deve essere uguali al numero fermata contenuto nella relazione "include" tra Tratta e Fermate.



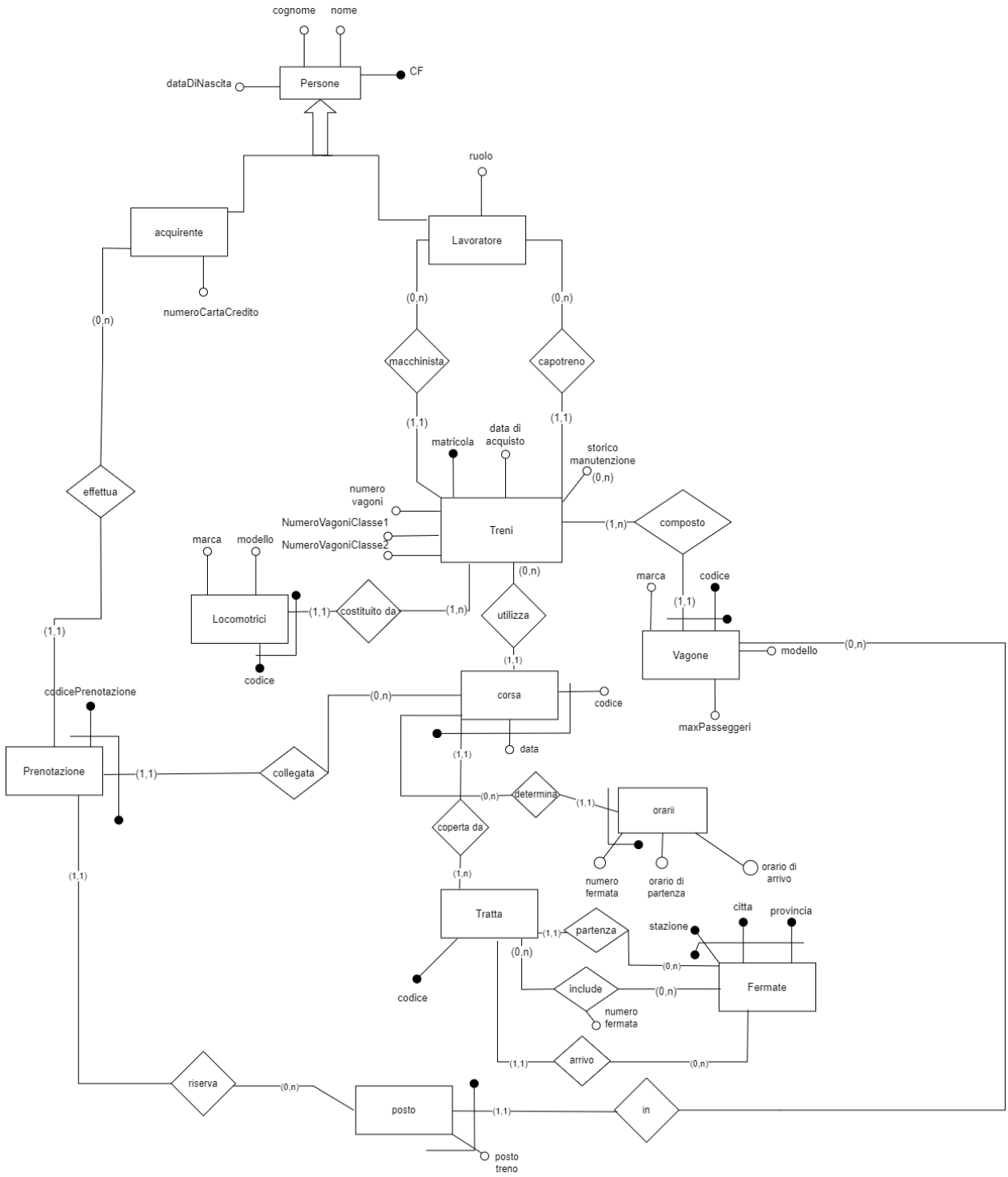
Di corsa reifico anche l'attributo prenotazione, si crea l'entità prenotazione in quanto si identifica dal codice della prenotazione e dalla relazione che ha con corsa. Si salva l'utente che effettua la prenotazione e anche il posto a sedere che gli viene assegnato. La relazione che ha con corsa è una "uno a n" in quanto una prenotazione può essere collegata a una sola corsa, mentre una corsa può avere 0, n prenotazioni.



-Successivamente reifico l'attributo posto di Prenotazione in una entità denominata Posto, la quale mi tiene traccia del posto a sedere, tra queste due entità c'è una associazione "uno a molti" in quanto una prenotazione può essere associata a un solo posto e un posto può essere associato a 0, n prenotazioni. L'entità posto viene identificata dal posto e dalla relazione con l'entità vagoni. In quanto un posto deve essere associato a un solo vagone, mentre un vagone a 0, n posti. Perveniamo così al seguente schema:

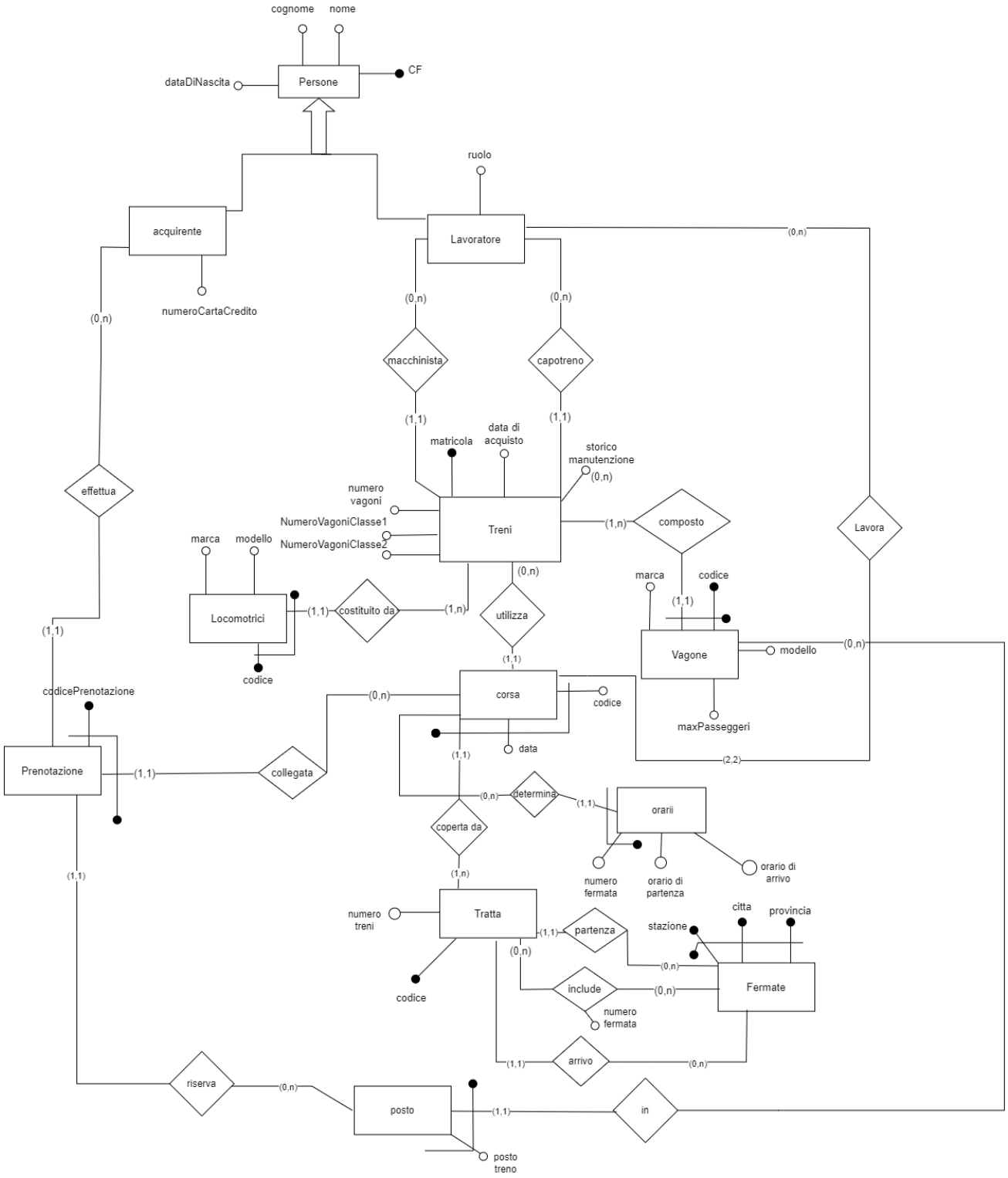


-Come ultima istanza, di prenotazione reifico l'attributo acquirente in una entità, in quanto si vuole salvare il proprio nome, cognome data di nascita, codice fiscale e numero di carta di credito. L'entità acquirente viene aggiunta come entità figlia alla classe padre Persone poiché contiene gli stessi attributi con l'unica aggiunta della carta di credito. La relazione che si ha tra prenotazione e acquirenti è una "uno a molti" in quanto una prenotazione deve essere associata a un acquirente mentre un acquirente può fare 0, n prenotazioni.



-Infine, traccio l'ultima relazione tra lavoratore e corse in quanto come richiesto da specifiche l'orario lavorativo è strettamente legato alle corse in cui viene coinvolto il lavoratore. La relazione tra Lavoratore e corse è una "due a n" in quanto un lavoratore può far parte di 0, n corse, ma una corsa devono far parte 2 lavoratori, un macchinista e un capotreno.

Integrazione finale



Regole aziendali

-L’ ora di partenza deve essere maggiore all’ora di arrivo della stessa fermata.

-La carta di credito deve avere sedici simboli.

-Un macchinista o un capotreno non possono essere coinvolti in più corse durante lo stesso orario di lavoro.

-Un treno non può compiere corse diverse negli stessi orari.

-La matricola del treno deve avere quattro cifre.

-Per i capilinea è disponibile soltanto l'orario di arrivo o di partenza previsto.

-Il numero di prenotazioni per una corsa non può superare la capienza dell'treno.

- Il lavoratore è associato al treno che effettua la corsa.

-Il numero della fermata nella entità orari deve essere uguale al numero fermata della relazione include.

-Non possono essere fatte più manutenzione nello stesso giorno per un treno.

-I vagoni possono essere di 1 o 2 classe.

Dizionario dei dati

Entità	Descrizione	Attributi	Identificatori
Treni	Mezzo di Trasporto pubblico utilizzato per coprire una determinate tratta.	Matricola,Data di acquisti, Storico manutenzioni, Numero vagoni, Numero vagoni classe1, Numero vagoni classe 2.	Matricola
Locomotrici	Utilizzata per trainare treni di vagoni o carri ferroviari.	Marca, modello.	Codice, matricolaTreno
Tratta	Percorso che viene fatto dal treno durante una corsa.	codice, Capolinea Arrivo, Capolinea Partenza,	codice
Vagone	Struttura su ruote che trasporta passeggeri.	Marca, modello, numero massimo passeggeri.	Codice, matricolaTreno
Fermate	Dove il treno si ferma.	Stazione, Città, Provincia	Stazione, città, provincia
Persone	Soggetti di partecipazione.	Codice Fiscale, cognome, nome, data Di Nascita.	Codice fiscale
Prenotazione	Attestazione che permette all'acquirente di accedere alla tratta.	Codice Prenotazione, posto Treno, Tratta.	Codice prenotazione, dataCorsa, codiceTratta, codiceCorsa

Acquirenti	Le informazioni sul cliente che effettua una prenotazione.	Numero carta di credito	Codice fiscale
Lavoratori	Tutte le persone coinvolte all'interno di una organizzazione o un contesto lavorativo.	ruolo	Codice fiscale
Corsa	La corsa reale fatta dal treno.	Codice, data	Codice, data, codiceTratta
orari	Gli orari in cui vengono fatte le fermate	Numero fermata	Numero Fermata, codiceCorsa
posto	Il posto a sedere	Posto treno	Posto treno,vagone,Treno

4. Progettazione logica

Volume dei dati

Durante la fase di progettazione, si considera che i dati conservati nel database siano relativi agli ultimi dieci anni e che, una volta trascorso questo periodo, il database possa essere ripulito eliminando tutte le prenotazioni e le corse fatte.

1. Considerando che vengono gestiti circa 80 treni che vengono revisionati su base settimanale. Perciò il volume atteso nelle storiche manutenzioni è di 41.142.
2. Si prevede che il numero totale delle tratte gestito è 50.
3. Inoltre, si ipotizza che ogni vagone abbia circa 20 posti a sedere e un treno ad alta velocità mediamente comprende 15 vagoni e 2 locomotive, di cui 10 vagoni di seconda classe e 5 di prima classe. Perciò il volume atteso dei vagoni è di 1200 mentre per le locomotive è 160.
4. Si presume che per ogni corsa il 70% della disponibilità sia prenotato, perciò 210 prenotazioni per ogni corsa.
5. Di tutte le prenotazioni il 35% viene fatto da persone ricorrenti. Perciò il numero degli acquirenti è 36855000.
6. In media, ogni tratta è composta da otto fermate e nel giro di 10 anni viene aggiunta una fermata l'anno per un volume di 1210.
7. Supponiamo che vengano organizzati i turni di 130 lavoratori e che ogni lavoratore, lavora 5 giorni a settimana per un volume totale di 385.714 turni gestiti dopo 10 anni. Mentre i lavoratori accedono per scaricare i turni una volta a settimana, perciò, con un volume totale di 66.857 accessi.
8. Consideriamo che ogni giorno tutte le tratte vengono coperte da almeno una corsa e che il 50% delle tratte anche da 2 corse. Perciò 75 corse al giorno. Perciò dopo 10 anni vengono fatte 270.000 corse
9. Treni sono 80, posti ogni treno sono 300. Posti totali 24000.
10. Poiché ogni tratta ha 8 fermate in media, le corse che vengono fatte in tutto sono 270.000, gli orari salvati sono 2.160.000.
11. In media ogni treno viene revisionato ogni mese

Concetto nello schema	Tipo[1]	Volume atteso
Treni	E	80
Locomotrici	E	160
Vagoni	E	1200
Tratta	E	50
Fermate	E	410
Prenotazioni	E	56.700.000
Persone	E	36.855.130
Acquirente	E	36.855.000
Corsa	E	270.000
posto	E	24.000
Lavoratore	E	130
Orari	E	2.160.000
utilizza	R	270.000
Coperta da	R	270.000
Composto	R	1200
determina	R	2.160.000
riserva	R	56.700.000
in	R	24.000
Effettua	R	56.700.000
Costituito da	R	160
Include	R	410
Collegata	R	56.700.000
Macchinista	R	80
Capotreno	R	80
Lavora	R	270.000

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

Tavola delle operazioni

Cod.	Descrizione	Frequenza attesa
AG1	Report turni di lavoro	130/settimana≃19/giorno
AG2	Registra prenotazione	472.500/mese≃15.750/giorno
AG3	Registra manutenzione	80/mese≃3/giorno
AG4	Registra Corse	2250/mese≃75/giorno
AG6	Registra Treno	0,66/mese
AG7	Registra locomotrice	1,3/mese
AG8	Registra Tratta	1,25/mese
AG9	Registra Fermata	3,41/mese
AG10	Registra Vagone	10/mese
AG11	Visualizza prenotazioni	472.500/mese≃15.750/giorno
L2	Registrazione	307.126/mese≃ 10.237/giorno
L1	Login	307.126/mese≃10.237/giorno

Costo delle operazioni

Si suppone, in questa fase di progettazione, che il costo di scrittura di un dato sia il doppio del costo in lettura.

Si calcolano gli accessi in lettura e scrittura che vengono fatti sulle entità ogni qualvolta si fa effettua una operazione.

Operazione L1- Login

Concetto	costrutto	Accessi	Tipo
Persone	E	1	L

Il costo dell'operazione è di 1 accesso, considerando la frequenza dell'operazione otteniamo 307.126 accessi al mese.

Operazione L2- Registrazione

Concetto	costrutto	Accessi	Tipo
Persone	E	1	S

Il costo dell'operazione è di 2 accessi, considerando la frequenza dell'operazione otteniamo 614.252 accessi al mese.

Operazione AG6- Registra Treno

Concetto	costrutto	Accessi	Tipo
Lavoratore	E	130	L
Treno	E	1	S

Il costo dell'operazione è di 132 accessi, in quanto bisogna leggere prima tuttu i lavoratori che sono in totale 130 e dopodiché registrare il treno. Considerando la frequenza dell'operazione che è di 0,66 ogni mese arrotondando a 1 otteniamo 132 accessi al mese.

Operazione AG10- Registra Vagone

Concetto	costrutto	Accessi	Tipo
Treno	E	80	L
Vagone	E	1	S

Il costo dell'operazione è di 82 accessi in quanto bisogna leggere prima tutti treni che sono in totale 80 e in seguito registrare il vagone con quale treno è associato. Considerando la frequenza dell'operazione che è di 10 ogni mese otteniamo 820 accessi al mese.

Operazione AG7- Registra Locomotrice

Concetto	costrutto	Accessi	Tipo
Treno	E	80	L
Locomotrice	E	1	S

Il costo dell'operazione è di 82 accessi in quanto bisogna leggere prima tutti treni che sono in totale 80 e in seguito registrare la locomotrice con quale treno è associata, considerando la frequenza dell'operazione che è di 1,3 ogni mese e arrotondando ad 1 per difetto otteniamo 82 accessi al mese.

Operazione AG9- Registra Fermata

Concetto	costrutto	Accessi	Tipo
Fermata	E	1	S

Il costo dell'operazione è di 2 accessi, considerando la frequenza dell'operazione che è di 3,41 ogni mese e arrotondando ad 3 per difetto otteniamo 6 accessi al mese.

Operazione AG8- Registra Tratta

Concetto	costrutto	Accessi	Tipo
Fermata	E	410	L
Tratta	E	1	S
Include	E	8	S

Il costo dell'operazione è di 428 accessi, poiché 410 sono il numero di tutte le fermate che vengono lette prima di registrare una nuova tratta, e 8 accessi in scrittura su include poiché di media una tratta ha 8 fermate e queste vengono salvate nella relazione include. Considerando la frequenza dell'operazione che è di 1,25 ogni mese e arrotondando a 1 per difetto otteniamo 428 accessi al mese.

Operazione AG3- Registra Manutenzione

Concetto	costrutto	Accessi	Tipo
Treno	E	80	L
Manutenzione	E	1	S

Operazione AG4- Registra Corsa

Concetto	costrutto	Accessi	Tipo
Treno	E	80	L
Include	E	1200	L
Lavoratore	E	130	L
Corsa	E	3.600	L
Orari	E	28.800	L
Orari	E	8	S
Corsa	E	1	S

Il costo dell'operazione è di 33.828 accessi, in quanto bisogna leggere prima tutti i treni per scegliere a chi assegnare la corsa che sono in totale 80, in seguito scegliere quale tratta fargli compiere con le

relative fermate nella relazione “include”, scegliere a quali lavoratori assegnare la corsa. Controllare se il treno e il lavoratore sono già impegnati in altre corse, registrare la nuova corsa e i relativi orari delle fermate. Considerando la frequenza dell’operazione che è di 2250 ogni mese otteniamo 76.113.000 accessi al mese.

Operazione AG2- Registra Prenotazione

Concetto	costrutto	Accessi	Tipo
Include	E	1200	L
Orari	E	14	L
Corsa	E	1	L
Corsa	E	1	L
Vagone	E	15	L
Prenotazione	E	1	S

Il costo dell’operazione è di 1.216 accessi, in quanto bisogna scegliere quale tratta fargli compiere con le relative fermate nella relazione “include”, in seguito leggere gli orari delle fermate associate alla corsa, in media 8 fermate perciò 14 orari in totale. Leggere il codice della corsa e il relativo treno associato alla corsa scelta e i vagoni del treno.
Considerando la frequenza dell’operazione che è di 472.500 ogni mese otteniamo 582.592.500 accessi al mese.

Operazione AG1- Report turni di lavoro.

Concetto	costrutto	Accessi	Tipo
Corsa	E	14	L
Orari	E	14	L

Il costo dell’operazione è di 28 accessi, in quanto si ipotizzi che ogni lavoratore faccia 5 turni a settimana e ogni volta vengono riportatati i turni della settimana passata e la prossima a venire .
Considerando la frequenza dell’operazione che è di 130 ogni settimana otteniamo 3640 accessi alla settimana, perciò, mese 14560 accessi al mese.

Operazione AG11- Visualizza Prenotazioni.

Concetto	costrutto	Accessi	Tipo
Prenotazione	E	1	L
Corsa	E	1	L
Orari	E	2	L

Il costo dell'operazione è di 4 accessi. Considerando la frequenza dell'operazione che è di 15.750 al giorno otteniamo 63.000 accessi al giorno, perciò, mese 1.890.000 accessi al mese.

Ristrutturazione dello schema E-R

Un buon schema di ristrutturazione si può suddividere in una serie di passi da effettuare in questa data sequenza:
-Analisi Ridondanze
-L'Eliminazioni delle generalizzazioni.
-Partizionamento/Accorpamento delle entità e associazioni.

Analisi Ridondanze

Analizzando lo schema E-R, emergono alcune ridondanze. Ad esempio, l'attributo "numero vagoni" risultano ridondanti poiché può essere dedotto tramite l'associazione "composto" come gli attributi "Numero Vagoni Classe 1" e "Numero Vagoni Classe 2". Inoltre, l'attributo "numero treni" nell'entità "tratta" possono essere estrapolati tramite l'entità corsa attraverso una query.

Poiché la lettura del numero di vagoni viene eseguita una volta per ogni treno, si è deciso di eliminare questo attributo, insieme a "Numero Vagoni Classe 1" e "Numero Vagoni Classe 2". Anche se mantenere questi tre attributi non comporterebbe un costo elevato (considerando, ad esempio, che sono di tipo INTEGER, quindi 4 byte * 80 treni * 3 attributi = 960 byte), l'eliminazione di tali attributi comporta l'aggiunta di un unico attributo "Classe" nell'entità "vagoni". Questo approccio è più efficiente e risolve i problemi di ridondanza dei dati. In caso di eliminazione di un vagone per un determinato treno, non sarà necessario aggiornare i campi "numero vagoni" e "numero vagoni Classe 1" o "Numero Vagoni Classe 2".

Stessa cosa analizzando l'attributo numeroTreni nell'entità tratta, lo eliminiamo per evitare problemi di ridondanza e inconsistenza dei dati poiché nel caso in cui una tratta venissi coperta da un treno in più bisogna modificare anche l'attributo numeroTreni.

In seguito, rimuovo anche l'attributo capolineaPartenza e capolineaArrivo da tratta in quanto avendo l'attributo numero fermata ordinata, tramite una semplice query posso trovarmi subito il capolinea di partenza e arrivo per ogni tratta all'interno della relazione include, cosi evitando problemi di inconsistenza.

Ridefinisco l'identificatore della corsa, utilizzando soltanto il codice corsa e la data. Questi due elementi già garantivano l'unicità necessaria, rendendo superflua l'inclusione del codice della tratta.

Un identificatore composto da tre colonne (id tratta, codice corsa, e data) richiede indici più grandi e più complessi. Gli indici più grandi possono rallentare le operazioni di ricerca e aumentare il tempo necessario per l'inserimento e l'aggiornamento dei record.
Efficienza delle Query: Le query che utilizzano un identificatore più complesso potrebbero essere meno efficienti, specialmente se la colonna aggiuntiva (id tratta) non è sempre necessaria per distinguere in modo univoco le corse.

L'attributo max passeggeri nell'entità vagoni, risulta essere ridondante in quanto posso recuperarmi il numero di passeggeri per ogni vagone direttamente dalla tabella posto.
Analizzando il costo totale vediamo:

Calcolo:

Dimensione della tupla:

- Dimensione di un attributo `int`: 4 byte

Dimensione totale per tutte le tuple:

- Numero di tuple: 1200
- Dimensione totale: 1200×4 byte=4800 bytes.

Andando ad eliminare questo attributo quando vado ad eseguire l'operazione Registra prenotazione avrei:

Concetto	costrutto	Accessi	Tipo
Include	E	1200	L
Orari	E	14	L
Corsa	E	1	L
Corsa	E	1	L
Vagone	E	15	L
Posto	E	375	L
Prenotazione	E	1	S

Il costo totale senza l'attributo ridondante aumenta a 1607 accessi, poiché l'operazione deve scansionare ogni vagone del treno per determinare quanti posti ha, leggendo tupla per tupla. Con una frequenza di 472.500 operazioni al mese, questo comporta 759.307.500 accessi mensili, ovvero 176.715.000 accessi in più rispetto a lasciare l'attributo ridondante. Pertanto, è stato deciso di mantenere l'attributo MaxPasseggeri per ottimizzare le prestazioni.

Eliminazione Generalizzazione

In seconda istanza, è necessario affrontare l'unica generalizzazione presente nel nostro sistema. La scelta migliore per eliminare questa generalizzazione è accorpare l'entità padre nelle entità figlie.

Analizziamo questa decisione: l'entità "Persone" non è direttamente coinvolta in nessuna associazione, mentre le entità "Acquirente" e "Lavoratore" lo sono. Di conseguenza, non è necessario aggiungere ulteriori associazioni se seguiamo questa strada.

L'unico svantaggio di questa scelta è che comporterebbe l'aggiunta di alcuni attributi alle entità "Acquirente" e "Lavoratore", poiché le entità figlie erediteranno tutti gli attributi dell'entità padre. Tuttavia, questo comporterebbe un costo minimo di 1 byte per ogni attributo aggiunto.
Nel caso accorpassimo i figli nel padre ci ritroveremmo una tabella persone contenete 36.855.130 di tuple. Considerando le diverse operazioni che andremo a fare come registra treni, registra corse etc.
Leggere 130 tuple da una tabella che contiene 130 tuple in totale rispetto a leggere le stesse 130 righe da una tabella di 36.855.130 di tuple può avere diverse implicazioni tecniche, specialmente in termini di prestazioni e risorse utilizzate dal database. Ecco cosa cambia:

1.Efficienza degli Indici:

In una tabella con 130 tuple, l'intero set di dati può essere facilmente gestito in memoria, e le query sono solitamente molto rapide perché il database può scansionare l'intera tabella molto velocemente.

In una tabella con 36.855.130 di tuple, l'efficienza degli indici diventa cruciale. Se le 130 righe che devi leggere sono ben indicizzate, il database può utilizzare l'indice per trovare rapidamente le righe necessarie senza dover scansionare l'intera tabella. Tuttavia, la costruzione e la manutenzione degli indici richiedono risorse aggiuntive.

2.Uso della Cache:

Una tabella con 130 tuple può essere interamente contenuta nella cache del database, rendendo le operazioni di lettura estremamente veloci.

In una tabella molto grande, come quella con 56.000.000 di tuple, è improbabile che l'intera tabella possa essere contenuta nella cache. Di conseguenza, potrebbero esserci più accessi al disco, che sono più lenti rispetto all'accesso alla memoria.

3.Tempo di Ricerca:

Per una tabella piccola, il tempo di ricerca è trascurabile, poiché la scansione dell'intera tabella è molto rapida.

Per una tabella grande, il tempo di ricerca dipende molto dall'efficienza degli indici. Senza un buon indice, la scansione di una tabella così grande per trovare 130 righe specifiche sarebbe molto inefficiente e lenta.

4.Contesa delle Risorse:

Le tabelle più piccole tendono a creare meno contesa per le risorse del sistema, come la CPU, la memoria e l'I/O del disco.

Le tabelle grandi possono causare una maggiore contesa delle risorse, specialmente se ci sono molte operazioni di lettura e scrittura simultanee. Questo può influenzare le prestazioni complessive del sistema.

In sintesi, mentre la lettura di 130 righe potrebbe sembrare la stessa operazione indipendentemente dalla dimensione della tabella, le implicazioni tecniche e le prestazioni possono variare significativamente a seconda delle dimensioni della tabella e dell'efficienza con cui il database è configurato e ottimizzato.

Partizionamento/Accorpamento delle entità e associazioni.

Come ultima modifica sono andato a disaccoppiare l’attributo multivore storico manutenzioni in una entità denominata storico manutenzioni. In quanto andando a separarla consente di eliminare le varie ridondanze che si andrebbero a creare.

Motivazione della Semplificazione della Chiave Primaria nella Tabella Prenotazione

Nella fase iniziale della modellazione concettuale della nostra base di dati, la tabella Prenotazione era definita con una chiave primaria composta da tre campi:

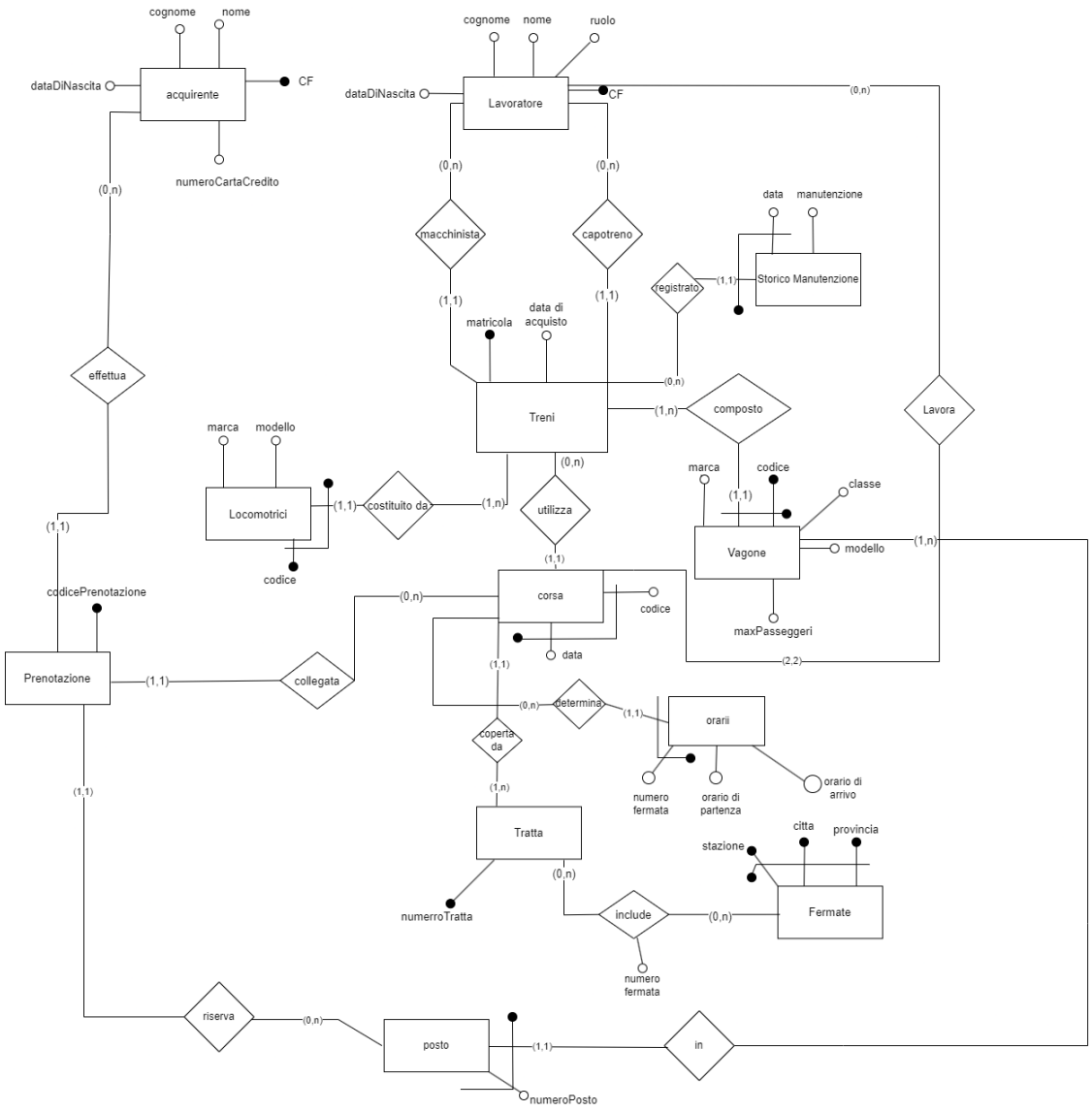
codice_prenotazione: Un identificatore unico per ogni prenotazione.
codice_corsa: Identificativo della corsa per cui è stata effettuata la prenotazione.
data_corsa: La data della corsa specifica.
Questa struttura è stata scelta per garantire l'unicità delle prenotazioni per ogni combinazione di corsa e data.

Analisi delle Esigenze
Dopo un'analisi approfondita delle esigenze del sistema e delle operazioni tipiche eseguite sui dati, sono emerse alcune considerazioni chiave:

Univocità e Identificazione:

codice_prenotazione è stato determinato essere un identificatore univoco sufficiente per ogni prenotazione. Ogni codice prenotazione è unico nel sistema e garantisce l'identificazione univoca di ogni record nella tabella Prenotazione.
Integrità referenziale:

Utilizzare una chiave primaria composta da un solo campo (codice_prenotazione) semplifica la manutenzione del database e le operazioni CRUD (Create, Read, Update, Delete). Le operazioni sui dati risultano più efficienti e meno soggette a errori.



Trasformazione di attributi e identificatori

●***matricolaTreno*** viene usato come Foreign key associata alla **Matricola** della entità **Treno**

●***idTratta*** viene usato come Foreign key associata all' **id** della entità **Tratta**

● *CFmacchinista, CFcapotreno* viene usato come Foreign key associata al CF della entità **Lavoratore**

Traduzione di entità e associazioni

●Treno (**<u>Matricola</u>**, data di Acquisto, *CFmacchinista, CFcapotreno*)

○Treno (*CFmacchinista*) ⊆ Lavoratore (*CF*)

○Treno (*CFcapotreno*) ⊆ Lavoratore (*CF*)

●Storico Manutenzione (**<u>dataManutenzione, *matricolaTreno*</u>**, manutenzione)

○Storico Manutenzione (Matricola treno) ⊆ Treni (*Matricola*)

●Vagone (**codice**, ***matricolaTreno***,modello, marca, maxPasseggeri, classe)

○Vagone(*matricolaTreno*) ⊆ Treni (*Matricola*)

●Locomotrici (**codice**, **matricolaTreno**, marca, modello)

○Locomotrici(*matricolaTreno*) ⊆ Treni (*Matricola*)

●Tratta (**id**)

●Include (**idTratta, stazione, città, provincia,** numeroFermata)

○ include (*stazione*, *città*, *provincia*) ⊆ Fermate (*stazione*, *città*, *provincia*)

○ include (*idTratta*) ⊆ Tratta (*id*)

●Fermate (**stazione, città, provincia**)

●Prenotazione (**codicePrenotazione,** idCorsa, data_**,** postroTreno, id_vagone, matricolatreno, CFAcquirente)

○ Prenotazione (*CFAcquirente*) ⊆ Acquirente (*CF*)

○ Prenotazione (idCorsa) ⊆ Corsa (idCorsa)

○ Prenotazione (data_) ⊆ Corsa (dataCorsa)

○ Prenotazione (postroTreno) ⊆ posto (*posto_treno*)

○ Prenotazione (id_vagone) ⊆ posto (*id_vagone*)

○ Prenotazione (matricolatreno) ⊆ posto (*matricolaTreno*)

●Lavoratore (**CF**, nome, cognome, dataDiNascita,ruolo,)

●Acquirente (**CF**, nome, cognome, dataDiNascita, numeroCartaCredito)

●Orari (**numeroFermata**, **codiceCorsa**, **dataCorsa**, orarioPartenza, orarioArrivo)

○ Orari (codiceCorsa) ⊆ Corsa (idCorsa)

○ Orari (dataCorsa) ⊆ Corsa (dataCorsa)

●Corsa (**idCorsa**, **dataCorsa**, idTratta, matricolaTreno, CFLavoratore1, CFLavoratore2)

○ Corsa (idTratta) ⊆ Tratta (id)

○ Corsa (matricolaTreno) ⊆ Treno (Matricola)

○ Corsa (CFLavoratore1) ⊆ Lavoratore (CF)

○ Corsa (CFLavoratore2) ⊆ Lavoratore (CF)

●Posto (**postoTreno**, **id_vagone, matricolaTreno**)

○ Posto (**id_vagone**) ⊆ vagone (*codice*)

○ Posto (**matricolaTreno**) ⊆ vagone (*matricolaTreno*)

Normalizzazione del modello relazionale

1NF

Ricordando che una relazione soddisfa la 1NF se
· e presente chiave primaria
· non ci sono attributi ripetuti
· gli attributi non sono strutture complesse

Le tabelle fornite soddisfano la prima forma normale, in quanto ogni colonna contiene un singolo valore, non ci sono ripetizioni o gruppi di colonne e le chiavi esterne collegano correttamente le tabelle.

2NF

Ricordando che una relazione e in 2NF se la chiave scelta non contiene chiavi più piccole (ovvero la chiave scelta e effettivamente una chiave e non una superchiave).
In tutte le relazioni le chiavi non contengono chiavi più piccole e quindi sono minimali.

3NF

Una relazione e in terza forma normale se per ogni dipendenza funzionale $X \rightarrow A$ vale una delle seguenti:

• X e superchiave della relazione

• A appartiene ad almeno una chiave

Nel nostro caso questo vale per tutte le relazioni.

Utenti e privilegi

Valutare se inserire anche Manutentore che si occupa di AG6(manutenzione treni).

Sono previsti 5 ruoli

-Login

Grant in esecuzione su L1.

-Acquirenti

Grant in esecuzione su AG2-AG11.

-Gestori del Servizio

Grant in esecuzione su AG4-AG6-AG7-AG8-AG9-AG10.

-Lavoratori

Grant in esecuzione su AG1.

-Manutentore

Grant in esecuzione su AG3.

Strutture di memorizzazione

Tabella <GestoreDelServizio>		
Attributo	**Tipo di dato**	**Attributi[2]**
Username	VARCHAR (20)	PK, NN
Password	VARCHAR (45)	NN
Tabella <Manutentore>		
Attributo	**Tipo di dato**	**Attributi[3]**
Username	VARCHAR (20)	PK, NN
Password	VARCHAR (45)	NN
Tabella <Treno>		
Colonna	**Tipo di dato**	**Attributi[4]**
Matricola	VARCHAR (4)	PK, NN
Data di acquisto	DATE	NN
CFMacchinista	VARCHAR (16)	NN
CFCapotreno	VARCHAR (16)	NN

Tabella <Vagone>		
Colonna	**Tipo di dato**	**Attributi**
codice	VARCHAR (10)	PK, NN
marca	VARCHAR (20)	NN
modello	VARCHAR (20)	NN
classe	INT	NN
maxPasseggeri	INT	NN
matricolaTreno	VARCHAR (4)	NN

Tabella <Locomotrici>		
Colonna	**Tipo di dato**	**Attributi**
codice	VARCHAR (10)	PK, NN
marca	VARCHAR (20)	NN

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

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

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

modello	VARCHAR (20)	NN
matricolaTreno	VARCHAR (4)	NN

Tabella <Tratta>		
Colonna	**Tipo di dato**	**Attributi**
id	INT	PK, NN

Tabella <Fermate>		
Colonna	**Tipo di dato**	**Attributi**
stazione	VARCHAR (50)	PK, NN
città	VARCHAR (50)	PK, NN
provincia	VARCHAR (50)	PK, NN

Tabella <Include>		
Colonna	**Tipo di dato**	**Attributi**
idTratta	INT	PK, NN
stazione	VARCHAR (50)	PK,NN
città	VARCHAR (50)	PK,NN
provincia	VARCHAR (50)	PK,NN
numeroFermata	INT	NN

Tabella <Lavoratore>		
Colonna	**Tipo di dato**	**Attributi**
CF	VARCHAR (16)	PK, NN
nome	VARCHAR (20)	NN
cognome	VARCHAR (20)	NN
dataDiNascita	DATE	NN
username	VARCHAR (45)	NN, UNIQUE
password	VARCHAR (45)	NN
ruolo	VARCHAR (45)	NN

Tabella <Acquirente>		
Colonna	**Tipo di dato**	**Attributi**
CF	VARCHAR (16)	PK, NN
nome	VARCHAR (20)	NN
cognome	VARCHAR (20)	NN
dataDiNascita	DATE	NN
username	VARCHAR (45)	NN, UNIQUE
password	VARCHAR (45)	NN
numeroCartaCredito	BIGINT	NN

Tabella <Prenotazione>		
Colonna	**Tipo di dato**	**Attributi**
codicePrenotazione	VARCHAR (20)	PK, NN
id_corsa	INT	NN
data_	DATA	NN
postoTreno	INT	NN
CFAcquirente	VARCHAR (16)	NN
idVagone	VARCHAR (10)	NN
matricolaTreno	VARCHAR (4)	NN

Tabella <Corsa>		
Colonna	**Tipo di dato**	**Attributi**
idCorsa	INT	PK, NN
dataCorsa	DATE	PK, NN
idTratta	INT	NN
matricolaTreno	VARCHAR (4)	NN
CFLavoratore1	VARCHAR (16)	NN
CFLavoratore2	VARCHAR (16)	NN

Tabella <Orari>		
Colonna	**Tipo di dato**	**Attributi**
numeroFermata	INT	PK, NN
orarioPartenza	TIME	
orarioArrivo	TIME	
idCorsa	INT	PK, NN
dataCorsa	DATE	PK, NN

Tabella <StoricoManutenzioni>		
Colonna	**Tipo di dato**	**Attributi**
dataManutenzione	DATE	PK, NN
matricolaTreno	VARCHAR (4)	PK, NN
manutenzione	VARCHAR (100)	NN

Tabella <Posto>		
Colonna	**Tipo di dato**	**Attributi**
Id_vagone	VARCHAR (10)	PK, NN
matricolaTreno	VARCHAR (4)	PK, NN
Posto_treno	INT	PK,NN

Indici

Tabella <nome>	
Indice <nome>	**Tipo[5]:**

Trigger

I 4 trigger sono progettati per garantire che l'username inserito o aggiornato non sia già presente in nessuna delle quattro tabelle specificate. Questo controllo di unicità previene la duplicazione degli username tra diverse categorie di utenti nel sistema

```
DELIMITER //

CREATE TRIGGER before_insert_or_update_trigger_gestore
BEFORE INSERT ON gestoredelservizio
FOR EACH ROW
BEGIN
   DECLARE user_count INT;

   -- Verifica che il nome utente non sia già presente in nessuna delle tabelle
   SELECT COUNT(*) INTO user_count
   FROM (
      SELECT username FROM gestoredelservizio WHERE username = NEW.username
      UNION ALL
      SELECT username FROM lavoratore WHERE username = NEW.username
      UNION ALL
```

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

```
    SELECT username FROM manutentore WHERE username = NEW.username
    UNION ALL
    SELECT username FROM acquirente WHERE username = NEW.username
  ) AS user_union;

  IF user_count > 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Il nome utente è già in uso in una delle tabelle';
  END IF;
END //

CREATE TRIGGER before_insert_or_update_trigger_lavoratore
BEFORE INSERT ON lavoratore
FOR EACH ROW
BEGIN
  DECLARE user_count INT;

  -- Verifica che il nome utente non sia già presente in nessuna delle tabelle
  SELECT COUNT(*) INTO user_count
  FROM (
    SELECT username FROM gestoredelservizio WHERE username = NEW.username
    UNION ALL
    SELECT username FROM lavoratore WHERE username = NEW.username
    UNION ALL
    SELECT username FROM manutentore WHERE username = NEW.username
    UNION ALL
    SELECT username FROM acquirente WHERE username = NEW.username
  ) AS user_union;

  IF user_count > 0 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'Il nome utente è già in uso in una delle tabelle';
  END IF;
END //
```

```
CREATE TRIGGER before_insert_or_update_trigger_manutentore
BEFORE INSERT ON manutentore
FOR EACH ROW
BEGIN
   DECLARE user_count INT;

   -- Verifica che il nome utente non sia già presente in nessuna delle tabelle
   SELECT COUNT(*) INTO user_count
   FROM (
      SELECT username FROM gestoredelservizio WHERE username = NEW.username
      UNION ALL
      SELECT username FROM lavoratore WHERE username = NEW.username
      UNION ALL
      SELECT username FROM manutentore WHERE username = NEW.username
      UNION ALL
      SELECT username FROM acquirente WHERE username = NEW.username
   ) AS user_union;

   IF user_count > 0 THEN
      SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'Il nome utente è già in uso in una delle tabelle';
   END IF;
END //

CREATE TRIGGER before_insert_or_update_trigger_acquirente
BEFORE INSERT ON acquirente
FOR EACH ROW
BEGIN
   DECLARE user_count INT;

   -- Verifica che il nome utente non sia già presente in nessuna delle tabelle
   SELECT COUNT(*) INTO user_count
   FROM (
```

```
        SELECT username FROM gestoredelservizio WHERE username = NEW.username
        UNION ALL
        SELECT username FROM lavoratore WHERE username = NEW.username
        UNION ALL
        SELECT username FROM manutentore WHERE username = NEW.username
        UNION ALL
        SELECT username FROM acquirente WHERE username = NEW.username
    ) AS user_union;

    IF user_count > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Il nome utente è già in uso in una delle tabelle';
    END IF;
END //

DELIMITER ;
```

Eventi

Viene aggiunto l’evento relativo alla cancellazione dei dati più vecchi di 10 anni, che risultano trascurabili ai fini dell’applicazione. In questo modo si evita che il volume dei dati mantenuti all’interno del sistema crescano in eccesso mantenendo dati irrilevanti ai fini del funzionamento del sistema. L’evento verrà eseguito per la prima volta dopo 10 anni e verrà ripetuto ogni anno, in modo da tenere il volume dei dati della base di dati stabile.

```
CREATE EVENT IF NOT EXISTS `trasporto_ferroviario`.`eliminazione_dati_vecchi`

ON SCHEDULE EVERY 1 YEAR

STARTS CURRENT_TIMESTAMP

ON COMPLETION PRESERVE

COMMENT 'Rimozione dati più vecchi di 10 anni - Test'

DO

BEGIN

   DELETE FROM orari

   WHERE dataCorsa <= DATE_SUB(NOW(), INTERVAL 10 YEAR);

   DELETE FROM prenotazione

   WHERE data_ <= DATE_SUB(NOW(), INTERVAL 10 YEAR);

   DELETE FROM corsa

   WHERE dataCorsa <= DATE_SUB(NOW(), INTERVAL 10 YEAR);
END//
```

Viste

Non sono state utilizzate delle viste.

Funzioni

Funzione controllo_cf

Funzione che si occupa di controllare la correttezza del codice Fiscale in fase di registrazione da parte dell'acquirente.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `controllo_cf`(codice_fiscale varchar(16))
RETURNS tinyint(1)
   DETERMINISTIC
```

```
BEGIN
        if codice_fiscale regexp '[A-Z]{6}[0-9]{2}[A-Z][0-9]{2}[A-Z][0-9]{3}[A-Z]$' then
                return true;
        else
                return false;
        end if;
END
```

Funzione controllo_carta_di_credito

Funzione che si occupa di controllare la correttezza del numero della carta in fase di registrazione da parte dell'acquirente.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `controllo_carta_di_credito`(carta_di_credito varchar(16)) RETURNS tinyint(1)
   DETERMINISTIC
BEGIN
        if carta_di_credito regexp '[0-9]{16}$' then
                return true;
        else
                return false;
        end if;
END
```

Funzione controllo_matricola

Funzione che si occupa di controllare che la matricola del treno sia di lunghezza 4 caratteri.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `controllo_matricola`(matricola VARCHAR(4)) RETURNS tinyint(1)
   DETERMINISTIC
BEGIN
   IF LENGTH(matricola) = 4 THEN
      RETURN 1;
```

```
   ELSE
      RETURN 0;
   END IF;
END
```

Funzione controlla_lavoratori_treno

Funzione che si occupa di controllare che gli stessi lavoratori che lavorano su un determinato treno sono gli stessi lavoratori che compiono la corsa.

```
CREATE  DEFINER=`root`@`localhost`  FUNCTION  `controlla_lavoratori_treno`(var_macchinista varchar(16),var_capotreno varchar(16),matricolaTreno varchar(4)) RETURNS tinyint(1)
   DETERMINISTIC
BEGIN

        DECLARE macchinista VARCHAR(16);
   DECLARE capotreno VARCHAR(16);

        SELECT CFMacchinista,CFCapotreno
   INTO macchinista,capotreno
   from treno
   WHERE matricola=matricolaTreno;



        if var_macchinista=macchinista AND var_capotreno=capotreno  then
                return true;
        else
                return false;
        end if;
END
```

Funzione controllo_lavoratore

Funzione che si occupa di controllare che durante la registrazione di una nuova corsa il lavoratore selezionato sia libero(non è già impegnato in altre corse) durante quell'intervallo di tempo.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `controllo_lavoratore`(
   CFLavoratore1_ VARCHAR(16),
   CFLavoratore2_ VARCHAR(16),
   var_data DATE,
   orario_partenza TIME,
   orario_arrivo TIME
) RETURNS tinyint(1)
   DETERMINISTIC
BEGIN
   DECLARE sovrapposizione BOOLEAN;

   SELECT
      EXISTS (
         SELECT 1
         FROM (
            SELECT
               orarioPartenza,
               orarioArrivo
            FROM
               (SELECT
                  corsa.idCorsa,
                  corsa.dataCorsa,
                  corsa.matricolaTreno,
                  CASE WHEN orari.orarioArrivo = '00:00:00' THEN orari.orarioPartenza ELSE orari.orarioArrivo END AS orarioPartenza,
                  CASE WHEN orari.orarioPartenza = '00:00:00' THEN orari.orarioArrivo ELSE orari.orarioPartenza END AS orarioArrivo
               FROM
                  orari
               JOIN
                  corsa ON corsa.dataCorsa = orari.dataCorsa AND corsa.idCorsa = orari.idCorsa
```

```
            WHERE
                (corsa.CFLavoratore1 = CFLavoratore1_ OR corsa.CFLavoratore2 =
CFLavoratore2_)
                AND corsa.dataCorsa = var_data
            ) AS combined_results
        ) AS subquery
        WHERE
            (orario_partenza BETWEEN orarioPartenza AND orarioArrivo
            OR orario_arrivo BETWEEN orarioPartenza AND orarioArrivo
            OR orarioPartenza BETWEEN orario_partenza AND orario_arrivo
            OR orarioArrivo BETWEEN orario_partenza AND orario_arrivo)
      ) INTO sovrapposizione;

   RETURN sovrapposizione;
END
```

Funzione max_passeggeri

Funzione che si occupa di controllare che durante la registrazione di un nuovo vagone il numero massimo di passeggeri sia massimo 25.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `max_passeggeri`(max_passeggeri INT)
RETURNS tinyint(1)
   DETERMINISTIC
BEGIN
        if max_passeggeri <= 25 then
                return true;
        else
                return false;
        end if;
END
```

Funzione controllo_treno

Funzione che si occupa di controllare che durante la registrazione di un nuovo corsa il treno selezionato sia libero(non è già impegnato in altre corse) durante quella fascia oraria.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `controllo_treno`(
   treno VARCHAR(16),
   var_data DATE,
   orario_partenza TIME,
   orario_arrivo TIME
) RETURNS tinyint(1)
   DETERMINISTIC
BEGIN
   DECLARE sovrapposizione BOOLEAN;

   SELECT
      EXISTS (
         SELECT 1
         FROM (
            SELECT
               orarioPartenza,
               orarioArrivo
            FROM
               (SELECT
                  corsa.idCorsa,
                  corsa.dataCorsa,
                  corsa.matricolaTreno,
                  CASE WHEN orari.orarioArrivo = '00:00:00' THEN orari.orarioPartenza ELSE orari.orarioArrivo END AS orarioPartenza,
                  CASE WHEN orari.orarioPartenza = '00:00:00' THEN orari.orarioArrivo ELSE orari.orarioPartenza END AS orarioArrivo
               FROM
                  orari
               JOIN
                  corsa ON corsa.dataCorsa = orari.dataCorsa AND corsa.idCorsa = orari.idCorsa
               WHERE
```

```
            corsa.matricolaTreno = treno AND corsa.dataCorsa = var_data
        ) AS combined_results
    ) AS subquery
    WHERE
      (orario_partenza BETWEEN orarioPartenza AND orarioArrivo
      OR orario_arrivo BETWEEN orarioPartenza AND orarioArrivo
      OR orarioPartenza BETWEEN orario_partenza AND orario_arrivo
      OR orarioArrivo BETWEEN orario_partenza AND orario_arrivo)
  ) INTO sovrapposizione;

  RETURN sovrapposizione;
END
```

Funzione controllo_orari

Funzione che si occupa di controllare che l'orario di partenza sia maggiore dell'orario di arrivo rispetto alla fermata considerata.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `controllo_orari`(orario_partenza Time, oraraio_arrivo time) RETURNS tinyint(1)
  DETERMINISTIC
BEGIN
    if orario_partenza > oraraio_arrivo THEN
        return true;
    else
        return false;
    end if;
END
```

Funzione controlla_classe

Funzione che si occupa di controllare che la classe del vagone sia di tipo 1 o 2.

```
CREATE DEFINER=`root`@`localhost` FUNCTION `controlla_classe`(classe int) RETURNS tinyint(1)
  DETERMINISTIC
```

```
BEGIN
	if classe =1 OR classe=2 then
		return true;
	else
		return false;
	end if;
END
```

Stored Procedures e transazioni

Procedura registra_treno

Procedura introdotta per registrare un nuovo treno. Poiché non vengono fatte letture nella transazione, la procedura non e affetta da anomalie perciò scelgo come livello di isolamento read uncommitted.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_treno`(
	in var_matricola varchar(4),
   in var_dataDiAcquisto varchar(45),
   in var_macchinista varchar(45),
   in var_capotreno varchar(45)
)
BEGIN
	declare exit handler for sqlexception
   begin
		rollback;
      resignal;
	end;

	set transaction isolation level READ uncommitted;
	start transaction;

	if(`trasporto_ferroviario`.controllo_matricola(var_matricola) is false) then
		signal sqlstate '45011' set message_text = 'Inserire il codice del treno di 4 caratteri.';
	end if;
	insert into treno values (var_matricola, var_dataDiAcquisto,var_macchinista,var_capotreno);
```

```
    commit;

END
```

Procedura registra_vagone

Procedura introdotta per registrare un nuovo vagone. Poiché non vengono fatte letture nella transazione, la procedura non e affetta da anomalie perciò scelgo come livello di isolamento read uncommitted.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_vagone`(
        IN var_id int,
    IN var_marca VARCHAR(45),
    IN var_classe INT,
    IN var_modello VARCHAR(45),
    IN var_numMaxPasseggeri INT,
    IN var_matricolaTreno INT
)
BEGIN
        declare exit handler for sqlexception
    begin
                rollback;
       resignal;
        end;

    set transaction isolation level read uncommitted;
        start transaction;

   if(`trasporto_ferroviario`.max_passeggeri(var_numMaxPasseggeri) is false) then
                signal sqlstate '45008' set message_text = 'Il numero di passeggeri è maggiore di 25.';
        end if;

    if(`trasporto_ferroviario`.controlla_classe(var_classe) is false) then
                signal sqlstate '45013' set message_text = 'Il vagone puo esssere di 1 o 2 classe.';
        end if;
```

```
   INSERT INTO vagone
   VALUES(var_id, var_marca,var_classe, var_modello,var_numMaxPasseggeri, var_matricolaTreno);

   COMMIT;
END
```

Procedura registra_Fermate

Procedura introdotta per registrare una nuova fermata. Poiché non vengono fatte letture nella transazione, la procedura non e affetta da anomalie, perciò, scelgo come livello di isolamento read uncommitted.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_Fermate`(
   IN stazione VARCHAR(50),
   IN citta VARCHAR(50),
   IN provincia VARCHAR(50)
)
BEGIN


   DECLARE exit handler for sqlexception
   BEGIN
      ROLLBACK;
      RESIGNAL;
   END;


   SET TRANSACTION ISOLATION LEVEL READ uncommitted;
   START TRANSACTION;

   INSERT INTO fermate VALUES(stazione, citta, provincia);
```

```
    COMMIT;
END
```

Procedura registra_locomotrici

Procedura introdotta per registrare una nuova locomotrice. Poiché non vengono fatte letture nella transazione, la procedura non e affetta da anomalie, perciò, scelgo come livello di isolamento read uncommitted.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_locomotrici`(
        IN vard_id VARCHAR(10),
    IN var_marca VARCHAR(20),
    IN var_modello VARCHAR(20),
    IN var_matricolaTreno INT)
BEGIN
        declare exit handler for sqlexception
        begin
                rollback;
      resignal;
        end;

    set transaction isolation level Read Uncommitted ;
        start transaction;


                insert into `locomotrici` (`codice`, `marca`, `modello`, `matricolaTreno`)
                        values (vard_id, var_marca, var_modello, var_matricolaTreno);
        commit;

END
```

Procedura registra_Tratta

Procedura introdotta per registrare una nuova tratta. Si prende tramite tipo di dato text tutte le fermate che compongono la nuova tratta, ogni fermata è separata da un carattere “,”. Il loop ‘cursor_loop’ estrae e processa ogni fermata dalla stringa ‘fermate’. Considerando il basso volume di

esecuzione della procedura assegno come livello di isolamento repeatable read che è un buon compromesso tra consistenza e performance e garantisce che, durante una transazione, se leggi un dato, non cambierà fino alla fine della transazione.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_Tratta`(
   IN fermate TEXT
)
BEGIN
   DECLARE citta_ VARCHAR(20);
   DECLARE provincia_ VARCHAR(20);
   DECLARE current_fermata VARCHAR(20);
   DECLARE start INT DEFAULT 1;
   DECLARE length INT DEFAULT 0;
   DECLARE done INT DEFAULT 0;
   Declare max_id INT DEFAULT 1;
        Declare counter INT DEFAULT 1;

   begin
                rollback;
      resignal;
        end;

   set transaction isolation level repeatable read ;
        start transaction;


   Select MAX(id)
      into max_id
      From tratta;

      IF max_id=NULL THEN
                        set max_id=1;
                else
                        set max_id=max_id+1;
                END if;
```

```
    INSERT INTO tratta(id) VALUES(max_id);

  cursor_loop: LOOP

    SET length = LOCATE(',', fermate, start) - start;


    IF length <= 0 THEN
       SET current_fermata = TRIM(SUBSTRING(fermate, start));
       SET done = 1;
    ELSE

       SET current_fermata = TRIM(SUBSTRING(fermate, start, length));
    END IF;


    SELECT citta, provincia
    INTO citta_, provincia_
    FROM fermate
    WHERE stazione = current_fermata;


    SELECT current_fermata AS debug_fermata, citta_ AS debug_citta, provincia_ AS
debug_provincia;


    INSERT INTO include (idTratta,stazione, citta, provincia,numeroFermata) VALUES
(max_id,current_fermata, citta_, provincia_,counter);


    SET start = start + length + 1;
    Set counter=counter +1;
```

```
        IF done THEN
            LEAVE cursor_loop;
        END IF;
    END LOOP;
END
```

Procedura new_registra_corsa

La procedura è progettata per registrare una nuova corsa ferroviaria, inserendo informazioni sulle fermate, gli orari, e i lavoratori coinvolti. Vediamo in dettaglio come funziona questa procedura e la logica della transazione.

- Calcola e pulisce gli orari forniti.
- Verifica la disponibilità dei lavoratori e del treno.
- Genera un nuovo ID per la corsa.
- Inserisce la nuova corsa nella tabella corsa.
- Elabora ogni orario dalla stringa var_orari e li inserisce nella tabella orari.

La scelta del livello di isolamento repeatable read è dovuta al fatto che prima di inserire la nuova corsa leggo l'ultimo codice delle corse precedenti in tabella, dello ulteriori dati tramite le funzioni controlla_lavoratori_treno, controllo treno, così facendo mi evito letture sporche inconsistenti e aggiornamenti fantasma. La frequenza con cui viene eseguita la procedura è circa 75 volte al giorno

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `new_registra_corsa`(
    IN var_macchinista VARCHAR(16),
    IN var_capotreno VARCHAR(16),
    IN var_treno VARCHAR(4),
    IN var_data DATE,
    IN var_idTratta INT,
    IN var_orari TEXT
)
BEGIN
    DECLARE var_idCorsa INT;
    DECLARE str_orario TEXT DEFAULT '';
    DECLARE orarioPartenza TIME;
```

```
DECLARE orarioArrivo TIME;
DECLARE numeroFermata INT DEFAULT 1;
DECLARE primo_orario TIME;
DECLARE ultimo_orario TIME;
DECLARE a INT DEFAULT 1;
DECLARE total_orari INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
BEGIN
  ROLLBACK;
  RESIGNAL;
END;

SET TRANSACTION ISOLATION LEVEL repeatable read;
START TRANSACTION;

-- Calcolare il numero totale di orari
SET total_orari = LENGTH(var_orari) - LENGTH(REPLACE(var_orari, ',', '')) + 1;

-- Rimuovere eventuali caratteri non numerici dagli orari
SET var_orari = REPLACE(REPLACE(REPLACE(REPLACE(var_orari, '[', ''), ']', ''), ' ', ''), '(', '');

SET primo_orario = SUBSTRING_INDEX(var_orari, ',', 1);
SET ultimo_orario = SUBSTRING_INDEX(REVERSE(SUBSTRING_INDEX(REVERSE(var_orari), ',', 1)), ',', 1);

IF (`trasporto_ferroviario`.controlla_lavoratori_treno(var_macchinista,var_capotreno,var_treno) IS FALSE) THEN
    SIGNAL SQLSTATE '45025' SET MESSAGE_TEXT = 'Questo treno ha assegnati diversi lavoratori';
END IF;

IF (`trasporto_ferroviario`.controllo_treno(var_treno,var_data,primo_orario,ultimo_orario) IS TRUE) THEN
```

```
    SIGNAL SQLSTATE '45020' SET MESSAGE_TEXT = 'Il treno è già impegnato in un altra corsa.';
  END IF;

  IF (`trasporto_ferroviario`.controllo_lavoratore(var_macchinista,var_capotreno,var_data,primo_orario,ultimo_orario) IS TRUE) THEN
    SIGNAL SQLSTATE '45021' SET MESSAGE_TEXT = 'Il lavoratore è già impegnato in un altra corsa.';
  END IF;

  -- Genera un nuovo idCorsa
  SELECT IFNULL(MAX(idCorsa), 0) + 1 INTO var_idCorsa FROM corsa;

  -- Inserire la nuova corsa nella tabella corsa
  INSERT INTO corsa (idCorsa, dataCorsa, idTratta, matricolaTreno, CFLavoratore1, CFLavoratore2)
  VALUES (var_idCorsa, var_data, var_idTratta, var_treno, var_macchinista, var_capotreno);

  -- Etichetta per il loop WHILE
  WHILE_LOOP: WHILE LENGTH(var_orari) > 0 DO
    -- Estrarre l'orario corrente dalla stringa
    SET str_orario = TRIM(SUBSTRING_INDEX(var_orari, ',', 1));

    -- Aggiungi la validazione degli orari
    IF INSTR(str_orario, '-') > 0 THEN
      -- Orario nel formato HH:MM-HH:MM (partenza e arrivo)
      SET orarioPartenza = STR_TO_DATE(SUBSTRING_INDEX(str_orario, '-', 1), '%H:%i');
      SET orarioArrivo = STR_TO_DATE(SUBSTRING_INDEX(str_orario, '-', -1), '%H:%i');

      IF (`trasporto_ferroviario`.controllo_orari(orarioPartenza, orarioArrivo) IS FALSE) THEN
        SIGNAL SQLSTATE '45003' SET MESSAGE_TEXT = 'Orario di partenza è minore di quello di arrivo';
      END IF;
```

```
        ELSE
            -- Gestione del primo orario singolo e dell'ultimo orario singolo
            IF a = 1 THEN
                -- Primo orario singolo nel formato HH:MM
                SET orarioPartenza = STR_TO_DATE(str_orario, '%H:%i');
                SET orarioArrivo = STR_TO_DATE('00:00', '%H:%i');
            ELSEIF a = total_orari THEN
                -- Ultimo orario singolo nel formato HH:MM
                SET orarioPartenza = STR_TO_DATE('00:00', '%H:%i');
                SET orarioArrivo = STR_TO_DATE(str_orario, '%H:%i');
            ELSE
                -- Gestione intermedia, ma dovrebbe essere inesistente secondo l'input fornito
                SET orarioPartenza = STR_TO_DATE(str_orario, '%H:%i');
                SET orarioArrivo = STR_TO_DATE('00:00', '%H:%i');
            END IF;
        END IF;

        -- Inserire l'orario nella tabella orari
        INSERT INTO orari (numeroFermata, orarioPartenza, orarioArrivo, idCorsa, dataCorsa)
        VALUES (numeroFermata, orarioPartenza, orarioArrivo, var_idCorsa, var_data);

        -- Aggiornare la stringa di orari rimanente
        SET var_orari = TRIM(SUBSTRING(var_orari, LENGTH(str_orario) + 2));

        -- Incrementare il numero della fermata e la posizione
        SET numeroFermata = numeroFermata + 1;
        SET a = a + 1;
    END WHILE WHILE_LOOP;

    COMMIT;
END
```

Procedura registra_prenotazione2

L’obiettivo di questa procedura è quello di trovare il primo vagone con un posto libero per la corsa selezionata, una volta assegnato il posto disponibile registra la prenotazione.
Il livello di isolamento utilizzato è repeatable read, così da evitare le letture sporche e inconsistenti. Essendo la procedura con la frequenza più alta, il livello di isolamento SERIALIZABLE sarebbe stato eccessivo e avrebbe aumentato di troppo la probabilità di deadlock.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_prenotazione2`(
   IN tratta INT,
   IN var_partenza TIME,
   IN var_data DATE,
   IN CF VARCHAR(16)
)
BEGIN
   DECLARE codicePrenotazione VARCHAR(20);
   DECLARE conta INT;
   DECLARE vagoneC VARCHAR(10);
   DECLARE maxP INT;
   DECLARE cur_corsa INT;
   DECLARE cur_treno VARCHAR(4);
   DECLARE vagone_candidato VARCHAR(10);
   DECLARE vagone_trovato BOOLEAN DEFAULT FALSE;
   DECLARE posto_trovato INT;
   DECLARE ultimo_posto INT;
   DECLARE vagoni_cursor CURSOR FOR
      SELECT codice, maxPasseggeri
      FROM vagone
      WHERE matricolaTreno = cur_treno;

   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN

      ROLLBACK;
      RESIGNAL;
   END;
```

```
    SET TRANSACTION ISOLATION LEVEL repeatable read;
    START TRANSACTION;

-- mi trovo l'id della corsa
    SELECT corsa.idCorsa
    INTO cur_corsa
    FROM orari, corsa
    WHERE corsa.dataCorsa = orari.dataCorsa
      AND corsa.idCorsa = orari.idCorsa
      AND orarioPartenza = var_partenza
      AND corsa.dataCorsa = var_data
      AND corsa.idTratta = tratta;



-- mi trovo la matricola del treno
    SELECT matricolaTreno
    INTO cur_treno
    FROM corsa
    WHERE idCorsa = cur_corsa;

    OPEN vagoni_cursor;

    vagoni_loop: LOOP
        FETCH vagoni_cursor INTO vagoneC, maxP;

        IF vagone_trovato THEN
            LEAVE vagoni_loop;
        END IF;

        SELECT COUNT(*)
        INTO conta
        FROM prenotazione
```

```
        WHERE id_corsa = cur_corsa AND id_vagone = vagoneC;

        IF conta < maxP THEN
            SET vagone_candidato = vagoneC;
            LEAVE vagoni_loop;
        END IF;
    END LOOP;

    CLOSE vagoni_cursor;



    IF vagone_candidato IS NULL THEN
        ROLLBACK;
        SIGNAL SQLSTATE '45030' SET MESSAGE_TEXT = 'La corsa ha raggiunto la capienza massima';
    END IF;

    SELECT MAX(postoTreno)
    INTO ultimo_posto
    FROM prenotazione
    WHERE id_vagone = vagone_candidato AND id_corsa = cur_corsa;

    IF ultimo_posto THEN
        SET posto_trovato = ultimo_posto + 1;
    ELSE
        SET ultimo_posto = 0;
        SET posto_trovato = 1;
    END IF;

   -- INSERT INTO posto (id_vagone, posto_treno, matricolaTreno)
   -- VALUES (vagone_candidato, posto_trovato, cur_treno);

    SET codicePrenotazione = SUBSTRING(MD5(CONCAT(UUID(), RAND())), 1, 20);
```

```
    INSERT INTO prenotazione (codicePrenotazione, postoTreno, CFAcquirente, matricolaTreno, id_vagone, id_corsa, data_)
    VALUES (codicePrenotazione, posto_trovato, CF, cur_treno, vagone_candidato, cur_corsa, var_data);

    COMMIT;
END
```

Procedura report_turni

La procedura è progettata per generare un report richiesto da un lavoratore. Il livello di isolamento scelto è READ UNCOMMITTED. Questa scelta è dovuta al fatto che i dati estratti dalla procedura sono utilizzati esclusivamente per scopi di lettura e non hanno impatti critici se sono temporaneamente inconsistenti. Pertanto, la procedura può essere vulnerabile a varie anomalie transazionali, ma queste non comportano conseguenze catastrofiche dato il contesto in cui i dati sono utilizzati.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `report_turni`(
        In CF VARCHAR(16)
)
BEGIN

        DECLARE exit handler for sqlexception
    BEGIN
      ROLLBACK;
      RESIGNAL;
    END;

    SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
    START TRANSACTION;

SELECT
    idCorsa,
    dataCorsa,
    matricolaTreno,
    MAX(orarioPartenza) AS orarioPartenza,
```

```
   MAX(orarioArrivo) AS orarioArrivo
FROM
   (SELECT
      corsa.idCorsa,
      corsa.dataCorsa,
      corsa.matricolaTreno,
      CASE WHEN orari.orarioArrivo = '00:00:00' THEN orari.orarioPartenza ELSE NULL END AS orarioPartenza,
      CASE WHEN orari.orarioPartenza = '00:00:00' THEN orari.orarioArrivo ELSE NULL END AS orarioArrivo
   FROM
      orari
   JOIN
      corsa ON corsa.dataCorsa = orari.dataCorsa AND corsa.idCorsa = orari.idCorsa
   WHERE
      (corsa.CFLavoratore1 = CF OR corsa.CFLavoratore2 = CF)
   ) AS combined_results
GROUP BY
   idCorsa, dataCorsa, matricolaTreno
ORDER By(dataCorsa);
END
```

Procedura registrazione

La procedura è progettata per permettere ai vari utenti di registrarsi.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registrazione`(
   IN p_codiceFiscale VARCHAR(16),
   IN p_nome VARCHAR(50),
   IN p_cognome VARCHAR(50),
   IN p_dataDiNascita DATE,
   IN p_numeroCartaDiCredito bigint,
   IN p_username VARCHAR(20),
   IN p_password VARCHAR(20)
)
```

```
BEGIN
declare exit handler for sqlexception
   begin
                rollback;
      resignal;
         end;

         set transaction isolation level read uncommitted;
         start transaction;

         if(`trasporto_ferroviario`.controllo_cf(p_codiceFiscale) is false) then
                signal sqlstate '45001' set message_text = 'Il codice fiscale inserito non è valido.';
         end if;

   if(`trasporto_ferroviario`.controllo_carta_di_credito(p_numeroCartaDiCredito) is false) then
                signal sqlstate '45002' set message_text = 'la carta di credito non è valida.';
         end if;
   -- Inserimento dell'utente nel database
   INSERT INTO acquirente (CF, cognome, nome, dataDiNascita,
numeroCartaCredito,username,password)
   VALUES (p_codiceFiscale, p_cognome ,p_nome, p_dataDiNascita,
p_numeroCartaDiCredito,p_username,md5(p_password));
END
```

Procedura login

La procedura è progettata per permettere ai vari utenti di loggarsi.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `login`(in var_username
                                VARCHAR(45), in var_password VARCHAR(45), out var_role INT)
BEGIN

-- var_role = 0 --> login
-- var_role = 1 --> acquirente
-- var_role = 2 --> gestore
-- var_role = 3 --> lavoratore
```

```
-- var_role = 4 --> manutentore

            set var_role = 0;
    if exists(
                    select * from acquirente
                    WHERE username = var_username AND
                    password = md5(var_password))
            then set var_role = 1;
    end if;

    if exists(
                    select * from gestoredelservizio
                    WHERE username = var_username AND
                    password =  md5(var_password))
            then set var_role = 2;
    end if;

    if exists(
                    select * from lavoratore
                    WHERE username = var_username AND
                    password =  md5(var_password))
            then set var_role = 3;
    end if;

    if exists(
                    select * from manutentore
                    WHERE username = var_username AND
                    password =  md5(var_password))
            then set var_role = 4;
    end if;

END
```

Procedura registra_manutenzione

La procedura è progettata per permettere al manutentore di inserire una nuova manutenzione. La scelta del livello di isolamento READ uncommitted è dovuto al fatto che la procedura inserisce soltanto nuovi dati e non legge dati da altre tabelle.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `registra_manutenzione`(
        in var_matricola varchar(45),
   in var_data varchar(45),
   in descrizione varchar(200)
)
BEGIN

declare exit handler for sqlexception
   begin
                rollback;
      resignal;
        end;

        set transaction isolation level read uncommitted;
        start transaction;



        insert into storico_manutenzione values (var_matricola, var_data,descrizione);

   commit;

END
```

Procedura visualizza_lavoratori

La procedura è progettata per leggere i diversi lavoratori. Mi assicuro il livello di isolamento REPETABLE READ in modo tale da non rischiale letture sporche e inconsistenti, poiché utilizzo molto spesso questi dati letti per registrare le nuove corse.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_lavoratori`(
```

```
    IN var_matricola VARCHAR(4),
    IN a INT
)
BEGIN
    DECLARE exit handler for sqlexception
    BEGIN
        ROLLBACK;
        RESIGNAL;
    END;

    SET TRANSACTION ISOLATION LEVEL repeatable read;
    START TRANSACTION;

    IF a = 0 THEN
        SELECT CF
        FROM lavoratore
        WHERE ruolo = 'capotreno';
    ELSEIF a = 1 THEN
        SELECT CF
        FROM lavoratore
        WHERE ruolo = 'macchinista';
    ELSEIF a = 2 THEN
        SELECT CFMacchinista
        FROM treno
        WHERE matricola = var_matricola;
    ELSEIF a = 3 THEN
        SELECT CFCapotreno
        FROM treno
        WHERE matricola = var_matricola;
    END IF;

    COMMIT;
END
```

Procedura CF_Acquirenti

La procedura è progettata per leggere il CF dell'acquirente loggato all'applicazione. Mi assicuro il livello di isolamento Read Commited in modo tale da non rischiale letture sporche.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CF_Acquirenti`(
        IN var_username VARCHAR(20)
)
BEGIN

declare exit handler for sqlexception
   begin
                rollback;
      resignal;
        end;

   set transaction isolation level read committed;
        start transaction;

   select CF
   from acquirente
   WHERE username=var_username;

   commit;

END
```

Procedura CF_Lavoratori

La procedura è progettata per leggere il CF del lavoratore loggato all'applicazione. Mi assicuro il livello di isolamento READ committed in modo tale da non rischiale letture sporche.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `CF_Lavoratori`(
        IN var_username VARCHAR(20)
)
BEGIN
```

```
declare exit handler for sqlexception
   begin
                  rollback;
      resignal;
         end;

   set transaction isolation level read committed;
         start transaction;

   select CF
   from lavoratore
   WHERE username=var_username;

   commit;

END
```

Procedura visualizza_fermate

Procedura che permetti di visualizzare le fermate per essere utilizzate in seguito per registrare una nuova tratta. Utilizzo il livello repeatable read in modo tale da avere il dato privo da tutte le anomalie tranne inserimento fantasma.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_fermate`()
BEGIN
   DECLARE EXIT HANDLER FOR SQLEXCEPTION
   BEGIN
      ROLLBACK;
      RESIGNAL;
   END;

   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

   SELECT stazione, citta, provincia
   FROM fermate;
```

```
END
```

Procedura visualizza_orari

Procedura che permetti di visualizzare gli orari delle varie corse. Utilizzo il livello repeatable read per evitare tutte le varie anomali tranne inserimento fantasma in quanto il dato letto in seguito servirà per essere registrato in altre tabelle.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_orari`(
   IN deta DATE,
   IN id_Tratta INT
   )
BEGIN

DECLARE exit handler for sqlexception
   BEGIN
      ROLLBACK;
      RESIGNAL;
   END;

   SET TRANSACTION ISOLATION LEVEL repeatable read;
   START TRANSACTION;



         SELECT
   idCorsa,
   dataCorsa,
   matricolaTreno,
   MAX(orarioPartenza) AS orarioPartenza,
   MAX(orarioArrivo) AS orarioArrivo
FROM
   (SELECT
```

```
        corsa.idCorsa,
        corsa.dataCorsa,
        corsa.matricolaTreno,
        CASE WHEN orari.orarioArrivo = '00:00:00' THEN orari.orarioPartenza ELSE NULL END AS orarioPartenza,
        CASE WHEN orari.orarioPartenza = '00:00:00' THEN orari.orarioArrivo ELSE NULL END AS orarioArrivo
    FROM
        orari
    JOIN
        corsa ON corsa.dataCorsa = orari.dataCorsa AND corsa.idCorsa = orari.idCorsa
    WHERE
        ( orari.dataCorsa=deta AND corsa.idTratta=id_Tratta)
    ) AS combined_results
GROUP BY
    idCorsa, dataCorsa, matricolaTreno
ORDER BY
    dataCorsa;

    commit;
END
```

Procedura viusalizza_prenotazione

Procedura che permetti di visualizzare le prenotazioni fatte dagli utenti. Utilizzo il livello READ COMMITTED per evitare lettura sporca.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_prenotazione`(
        IN Cf VARCHAR(16)
)
BEGIN

declare exit handler for sqlexception
    begin
```

```
                rollback;
    resignal;
        end;

  set transaction isolation level read committed;
        start transaction;

  SELECT prenotazione.matricolaTreno, prenotazione.id_vagone, prenotazione.data_,
prenotazione.postoTreno, MIN(orari.orarioPartenza) AS minOrarioPartenza
FROM prenotazione
JOIN corsa ON prenotazione.id_corsa = corsa.idCorsa AND prenotazione.data_ = corsa.dataCorsa
JOIN orari ON corsa.idCorsa = orari.idCorsa AND corsa.dataCorsa = orari.dataCorsa
WHERE prenotazione.CFAcquirente = Cf AND orari.orarioPartenza != 0
GROUP BY corsa.idCorsa, corsa.dataCorsa, prenotazione.matricolaTreno, prenotazione.id_vagone,
prenotazione.data_, prenotazione.postoTreno;


  commit;

END
```

Procedura visualizza_tratte

Procedura che permetti di visualizzare le tratta nel momento dell'acquisto di una nuova prenotazione. Utilizzo il livello repeatable read per evitare tutte le varie anomali tranne inserimento fantasma in quanto il dato letto in seguito servirà per essere registrato in altre tabelle.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_tratte`()
BEGIN


        DECLARE exit handler for sqlexception
  BEGIN
```

```
    ROLLBACK;
    RESIGNAL;
  END;

  SET TRANSACTION ISOLATION LEVEL repeatable read;
  START TRANSACTION;



  SELECT numeroFermata, idTratta, stazione
  FROM include
  ORDER BY idTratta,numeroFermata;

  COMMIT;
END
```

Procedura visualizza_treni

Procedura che permetti di visualizzare i treni presenti nella base di dati. Utilizzo il livello repeatable read per evitare tutte le varie anomali tranne inserimento fantasma in quanto il dato letto in seguito servirà per essere registrato in altre tabelle.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `visualizza_treni`()
BEGIN

      DECLARE exit handler for sqlexception
  BEGIN
    ROLLBACK;
    RESIGNAL;
  END;

  SET TRANSACTION ISOLATION LEVEL repeatable read;
  START TRANSACTION;

  select matricola
  from treno
```

```
    commit;

END
```