Compagnie di navigazione

Progetto di Basi di dati

Autori del progetto

Group Manager: Francesco Trotti
Componenet 2: Riccardo Puggioni

UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Indice

1 Progettazione concettuale

1.1 Analisi dei requisiti

Nella prima parte del nostro progetto andremo ad analizzare i requisti e tutte le informazioni utili allo sviluppo del database del sistema delle compagnie di navigazione. In questa parte saranno individuate le entità, gli attributi e le relazioni che avvengono tra di esse.

"Il sistema si basa sulla conoscenza delle corse offerte dalle compagnie di navigazione.Ogni corsa ha cadenza giornaliera, un orario di partenza e un orario di arrivo ma può essere operata solo in alcuni giorni della settimana e solo in alcuni specifici periodi dell'anno."

Per creare il nostro sistema riconosciamo come prima entità la **corsa**. Essa ha come attributi: *l'orario di partenza* e *l'orario di arrivo*. Le corse inoltre possono essere svolte solo in alcuni periodi e in alcuni giorni della settimana, dunque andiamo ad inserire degli attributi per rappresentare il periodo. Gli attributi in questione sono:

- *Data_Inizio_Servizio*;
- *Data_Fine_Servizio*;
- *Giorni_Servizio_Attivo*(un attributo multivalore contenente un boolean per ogni giorno della settimana).

"Ogni corsa ha diversi prezzi: un prezzo per il biglietto intero, uno per il biglietto ridotto. "

Quindi definiamo gli attributi *Prezzo_Intero*, *Prezzo_Ridotto* e oltre i precedenti attributi decidiamo di inserirne un altro: *Sconto_Residente*.

"Ogni corsa è offerta da una specifica compagnia di navigazione, che indica il tipo di natante utilizzato. Tra i tipi di natante si distinguono i traghetti (che trasportano persone e automezzi), gli aliscafi e le motonavi (che trasportano entrambe solo passeggeri). Ogni natante ha un nome che lo identifica. "

La singola corsa puó essere eseguita da un **natante**, che definiamo come

entità. Tale entità possiede una specializzazione disgiunta totale,le specializzazioni sono:

1. Traghetti;
2. Aliscafi;
3. Motonavi.

I natanti tra loro si distinguono tramite:

- il loro *nome*;
- che cosa *trasportano*.

Difatti i traghetti oltre i passeggeri, possono trasportare veicoli; cosa che aliscafi e motonavi non possono fare. Il nome del natante e cosa trasportano lo inseriamo tra gli attributi.

"Ogni corsa ha diversi prezzi: un prezzo per il biglietto intero, uno per il biglietto ridotto. Inoltre, può esserci un sovrapprezzo per la prenotazione e uno per i bagagli."

Definiamo l'entità **biglietto**. Il Biglietto ha come attributi *importo_totale*, *Sovrapprezzo_totale*, *N_Bagagli*, *Veicolo* e *Prenotazione*. Il biglietto lo possiamo contraddistinguere in:

1. *biglietto_intero*;
2. *biglietto_ridotto*.

Il **Biglietto_Ridotto** si riferisce ad un passeggero che non possiede l'età minima per viaggiare da solo e ha bisogno quindi di un **Accompagnatore**.
Oltre l'entità biglietto, creiamo una nuova entità chiamata: "**sovrapprezzo**".
Il sovrapprezzo può essere aggiunto in caso di:

1. *prenotazione*;
2. aggiunta di uno o più *bagaglio*;
3. aggiunta di un *veicolo*.

Per tutti questi casi definiamo specializzazioni apposite.

"Ogni corsa è caratterizzata da un porto di arrivo e da uno di partenza: nel caso di corse che abbiano uno scalo intermedio il sistema espone tra le sue corse tutte le singole tratte. Ad esempio, se esiste una corsa tra Mu e Atlantide con scalo a Tortuga, il sistema manterrà tutte e tre le corse da Mu a Tortuga, da Tortuga ad Atlantide e da Mu ad Atlantide.

Andiamo ad aggiungere all'entità corsa gli attributi che forniscono alcune informazioni sullo scalo, ovvero:

- *Orario_Partenza_Scalo*
- *Orario_Arrivo_Scalo*

"Ogni porto è caratterizzato dal comune di appartenenza, un indirizzo e dal numero di telefono del servizio informazioni"

Creiamo l'entità **porto**, che descrive i porti collegati da corse eseguite da natanti. Di questi porti dobbiamo definire:

- *Indirizzo*;
- *Comune*;
- *Tel_Info*;

"Ogni compagnia di navigazione ha un nome e una serie di contatti (telefono, mail, sito web indirizzi su diversi social)."

Aggiungiamo alla compagnia di navigazione un attributo strutturato "**contatti**" formato dai seguenti attributi semplici e uno multivalore.
Gli attributi semplici sono:

- *telefono*;
- *mail* ;
- *Sito_web*;

L'attributo multivalore è:

- *Social*(formato da più stringhe contenenti gli indirizzi dei vari social).

"Il sistema può essere utilizzato dalle compagnie e dai passeggeri."

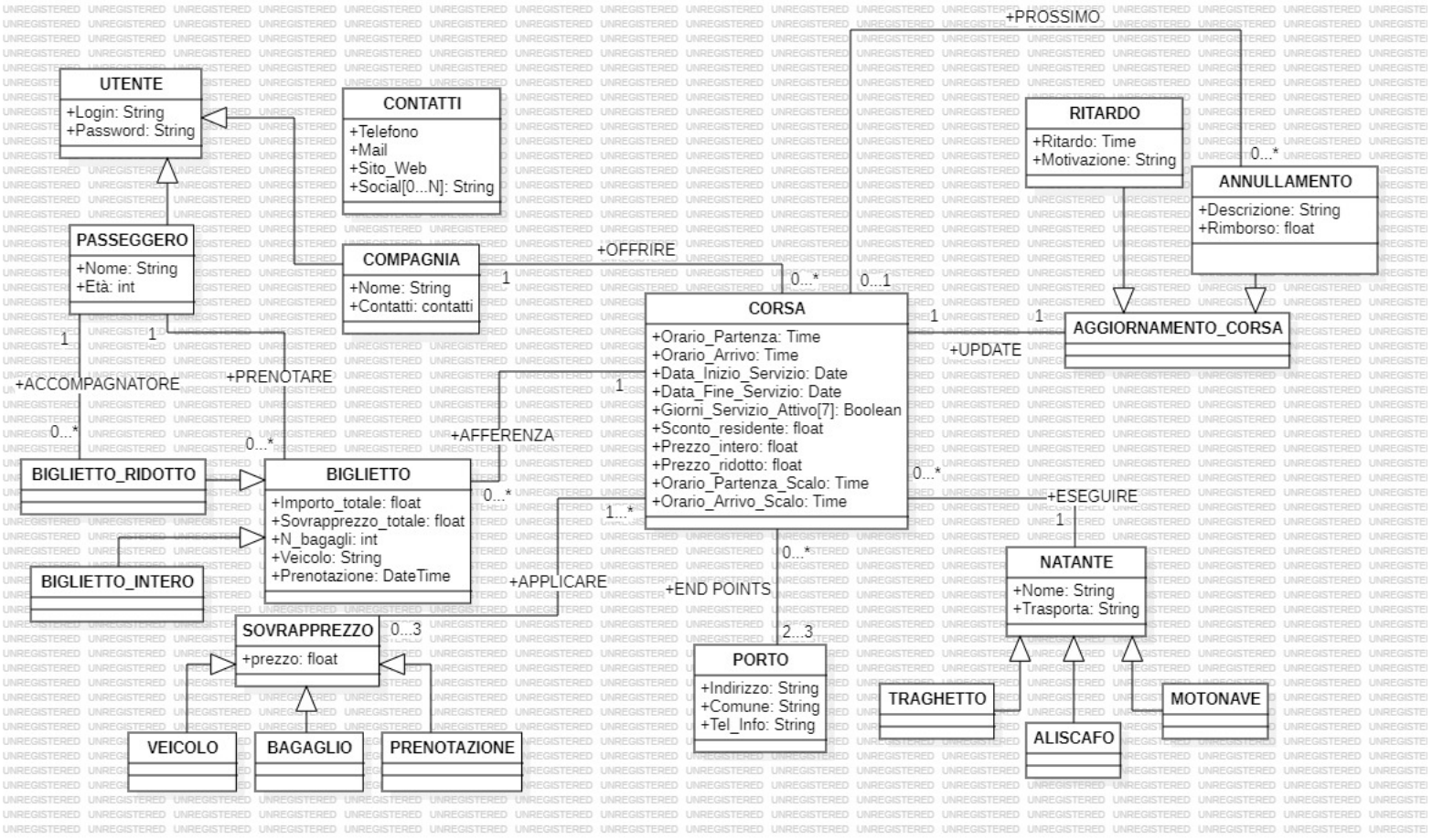
Per poter far interagire compagnie di navigazione e passeggeri con il nostro sistema, definiamo una classe **Utente**, in cui inseriamo una *login* e una *password* con cui si può accedere al sistema.

"Le compagnie possono aggiornare le proprie corse oppure segnalare l'annullamento o il ritardo di una singola corsa"

L'ultima aggiunta del nostro schema concettuale è l'entità **Aggiornamento_Corsa**. Con questa andiamo a gestire il *ritardo* della singola corsa e il suo eventuale *annullamento*, con due specializzazioni apposite. La specializzazione ritardo conterrà gli attributi *Ritardo* e *Motivazione*, la specializzazione **Annullamento** conterrà *Descrizione* e *Rimborso*. Inoltre Annullamento può indicare la **Prossima** corsa sostitutiva.

1.2 Schema concettuale

Dopo aver ultimato la fase di analisi dei requisiti lo schema concettuale che otteniamo si presenta così



1.3 Dizionario delle entità

Entità	Descrizione	Attributi
Corsa	Elemento cardine che definisce la tratta da un porto ad un altro	**Orario_Partenza**(Time): Indica l'orario in cui parte la corsa **Orario_Arrivo**(Time): Indica l'orario in cui la corsa arriva **Data_Inizio_Servizio**(Date): Indica il giorno in cui inizia il servizio di corse **Data_Fine_Servizio**(Date):Indica il giorno in cui termina il servizio di corse **Giorni_Servizio_attivo**(Boolean): Indica i giorni in cui il servizio è attivo **Prezzo_Intero**(float): costo del biglietto intero. **Prezzo_Ridotto**(float): costo del biglietto ridotto. **Sconto_Residente**:(float): percentuale di sconto per i residenti nel porto di arrivo. **Orario_Partenza_Scalo**: (time): Indica l'orario di partenza da porto di scalo **Orario_Arrivo_Scalo**: (time): indica l'orario di arrivo al porto di scalo
Natante	Mezzo con cui si svolge una corsa	**Nome**(String): Nome del natante utilizzato nella corsa **Trasporta**(String): Chi e/o cosa trasporta il natante, se solo persone o anche veicoli
Aggiornamento corsa	Entità che descrive l'aggiornamento dello stato della corsa	...
Ritardo	Indica il ritardo che può fare una corsa rispetto all'orario previsto	**Ritardo**(Time): Il tempo di ritardo effettivo della corsa. **Motivazione**(String): Motivazione del ritardo.
Annullamento	Entità che permette di notificare se una corsa è annullata	**Rimborso**(Float): Cifra che viene restituita al passeggero in caso di annullamento della corsa. **Motivazione**(String): Motivazione dell'annullamento.
Porto	Luogo dove stazionano, partono e arrivano i natanti	**Indirizzo**(String): Indica l'indirizzo del porto **Comune**(String): Indica il comune di appartenenza del porto **Telefono_info**(String): Rappresenta le informazioni telefoniche del porto

Entità	Descrizione	Attributi
Compagnia	Provider che fornisce la corsa per gli utenti	**Nome**(String): Rappresenta il nome della compagnia di navigazione **Contatti**(Contatti): Contatti della compagnia dio navigazione
Contatti	Elementi che permettono di poter contattare le compagnie di navigazione	**Telefono**(String): Indica il numero di telefono della compagnia **Mail**(String): Indica l'indirizzo di posta elettronica della compagnia **Sito_web**(String): Indica l'indirizzo del sito web della compagnia **Social**[0...N](String): indirizzi sui vari social
Utente	Account con cui il passeggero si interfaccia con la compagnia e account con cui la compagnia monitora le corse	**Login**(String): Login account dell'utente **Password**(String): Password dell'utente
Passeggero	Colui che usufruisce della corsa fornita dalla compagnia	**Nome**(String): Nome del passeggero
Biglietto	Biglietto utilizzato dal passeggere per poter acceder ad una corsa	**Importo_totale**(float): Indica quanto l'utente ha pagato per la corsa **Sovrapprezzo_totale**(float): Indica l'ammontare dei sovrapprezzi. **N_Bagagli**(Int): Indica il numero di bagagli portati dal passeggero. **Veicolo**(enum) : Indica il tipo di veicolo del passeggero **Prenotazione**(DateTime): Indica data e ora della prenotazione se c'è
Biglietto ridotto	Biglietto che viene pagato solo in parte se vengono rispettate certe condizioni	...
Biglietto intero	Biglietto che viene pagato interamente	...
Sovrapprezzo	Prezzo aggiuntivo da sommare al prezzo del biglietto della corsa in caso di aggiunte extra al biglietto standard	**Prezzo**(float): Indica il prezzo extra da sommare al prezzo del biglietto
Veicolo	Sovrapprezzo aggiunto al biglietto in caso di aggiunta di uno o più veicoli	...
Bagaglio	Sovrapprezzo aggiunto al biglietto in caso di aggiunta di uno o più bagagli	...
Prenotazione	Sovrapprezzo aggiunto al biglietto in caso di prenotazione del biglietto	...

1.4 Dizionario delle associazioni

Associazione	**Descrizione**
Offrire	Associazione **uno-a-molti** tra *Corsa* e *Compagnia*. Una corsa può essere offerta da una e una sola Compagnia di navigazione, e una compagnia può offrire una o più corse.
Afferenza	Associazione **uno-a-molti** tra *Biglietto* e *Corsa*. Un Biglietto può afferire ad una singola corsa, mentre una corsa può afferire ad uno o più biglietti.
Applicare	Associazione **tre-a-molti** tra *Corsa* e *Sovrapprezzo*. Ad una Corsa possono essere applicati da 0 a massimo 3 sovrapprezzi e un sovrapprezzo può essere applicato a molteplici corse.
Prenotare	Associazione **uno-a-molti** tra *Biglietto* e *Passeggero*. Un Biglietto può essere prenotato da un singolo passeggero, un passeggero però può prenotare uno o più biglietti.
Eseguire	Associazione **uno-a-molti** tra *Corsa* e *Natante*. Una corsa può essere eseguita un solo natante. Un natante può eseguire una o molteplici corse.
END POINTS	Associazione **due/tre-a-molti** tra *Porto* e *Corsa*. Una Corsa ha due porti(arrivo e partenza) e potenzialmente un porto di scalo, però per un porto possono transitare una o più corse.
Accompagnatore	Associazione **uno-a-molti**tra *Passeggero* e *Biglietto_Ridotto* Un biglietto_Ridotto può corrispondere ad un solo passeggero accompagnatore .Un passeggero può essere accompagnatore di uno o più possessori di un biglietto ridotto.
Prossimo	Associazione **uno-a-molti** tra *Corsa* e *Annullamento*. Per un annullamento si può indicare una corsa sostitutiva(Prossimo). Una corsa può essere sostitutiva di molteplici annullamenti.
Update	Associazione **uno-a-uno** tra *Corsa* e *Aggiornamento_Corsa* Una corsa può avere un aggiornamento e un aggiornamento corrisponde ad una singola corsa.

2 Ristrutturazione del modello concettuale

2.1 Analisi delle ridondanze

Tabella 1: Tavola dei volumi

COSTRUTTO	TIPO	VOLUME
corsa	E	1000
compagnie	E	80
social	E	240
biglietto_intero	E	20000
biglietto_ridotto	E	12000
passeggero	E	4000
porto	E	800
natante	E	110
ritardo	E	90
annullamento	E	20
offrire	A	1000
possiede	A	240
acquista_ridotto	A	12000
acquista_intero	A	20000
intero_valido_per	A	20000
ridotto_valido_per	A	12000
partenza	A	1000
arrivo	A	1000
esegue	A	1000
ritardo_corsa	A	90
annullamento_corsa	A	20
prossimo	A	20

Tabella 2: Tavola delle operazioni

OPERAZIONE	TIPO	FREQUENZA
Op. 1 - dati biglietto	I	25000
Op. 2 - dati corse	I	6000
Op. 3 - registrazione	I	25
Op. 4 - inserire corse	I	8
Op. 5 - Cancellazione giornaliera annullamenti e ritardi	B	1

Tabella 3: Tavola degli accessi Op.1 (senza ridondanza sovrapprezzo)

COSTRUTTO	N.VOLTE	L/S
biglietto(ridotto o intero)	1	L
valido_per(intero o ridotto)	1	L
corsa	1	L

Cio vuol dire che avremo: **3L x 25000volte = 75000 accessi giornalieri in lettura**.

Tabella 4: Tavola degli accessi Op.1 (con ridondanza sovrapprezzo)

COSTRUTTO	N.VOLTE	L/S
biglietto(ridotto o intero)	1	L

Quindi abbiamo: **1L X 25000 volte = 25000 accessi giornalieri in lettura**
Ma anche: **32000 biglietti x 4bytes = 128 Kbytes**

Per una scelta progettuale preferiamo avere $\frac{1}{3}$ degli accessi con realtiva occupazione di memoria aggiuntiva di 128 Kbytes (che sono relativamente pochi). Per lo stesso identico motivo di sovrapprezzo teniamo anche la ridondanza Importo_tot in biglietto($\frac{1}{3}$ degli accessi e +128Kbytes).

2.2 Eliminazione degli attributi multivalore

Ci concentriamo ora sulla eliminazione degli attributi multivalore. Abbiamo due attributi multivalore nel nostro modello concettuale:

1. L'attributo social dell'entità contatti.

2. L'attributo giorni_servizio_attivo dell' entità corsa.

Per l'attributo **Social** decidiamo di creare una classe social apposita, e di collegarla alla compagnia di navigazione tramite un associazione uno-a-molti, aggiungiamo nella classe social nuovi attributi:nome_social(String), indirizzo_social(String). Per l'attributo **Giorni_Servizio_Attivo** abbiamo deciso di utilizzare un unico attributo Giorni_Servizio_Attivo(BitSet) una stringa binaria di 7 bit la cui configurazione indica l'informazione sui vari giorni desiderata.

2.3 Eliminazione degli attributi composti

In questa fase ci soffermiamo sugli attributi composti e li risolviamo. L'unico attributo composto presente nel nostro schema concettuale è **contatti**. La soluzione che decidiamo di adoperare consiste nel portare tutti gli attributi di contatti in **Compagnia**. In questo modo evitiamo accessi aggiuntivi nel nostro sistema, cosa che avremmo avuto se avessimo creato una classe apposita per contatti.

2.4 Analisi delle generalizzazioni

In questa fase di ristrutturazione verranno analizzate e convertite le generalizzazioni, in quanto non possiamo rappresentare quest' ultime direttamente nel nostro modello logico.

Osservando la specializzazione di natante ci siamo accorti che portare il padre nelle figlie non sarebbe stata la soluzione ottimale. Osservando i calcoli infatti ci siamo resi conto che procedendo nel portare il padre nei figli, nel modello logico cio verrebbe mappato in due possibili modi:

1. mappare le associazioni in corsa nel seguente modo: corsa(.....,n1*,n2*,n3*) dove n1 n2 e n3 sono chiavi esterne (nullable) delle tre diverse associazioni derivanti dall associazione originaria tra il padre e corsa. Tuttavia è da mettere in evidenza che per ogni corsa una e una soltanto di queste 3 chiavi sarà non null, mentre le altre due lo saranno: ciò implica uno spreco di memoria (ipotizzando di avere a regime 1000 corse) 1000 x 2 = 2000 valori null, in termini di bytes avremmo 2000 x 4bytes = 8Kb di memoria sprecata.

2. mappare le associazioni con tabella apposita con le coppie (corsa,natante) tuttavia questa soluzione, in primis raddoppierebbe gli accessi, mentre in secundis dovremmo tenere traccia di coppie di identificativi e non si un solo identificativo per ogni corsa, ciò vuol dire che avremmo 4bytes(dimensione ipotetica di un id) x 1000 corse = 4Kb di memoria in più di cui tenere traccia.

Portare i figli nel padre tuttavia comporterebbe la seguente mappatura logica:
corsa(....,natante) dove natante sarà sicuramente non null, ma natante avrà

ora necessariamente un attributo tipo(una stringa di al piu 9 caratteri) e quindi 9bytes x (200 natanti a regime) = 1.8 Kb di memoria in piu.

Siccome si prevede una grande frequenza di accesso giornaliera tra corsa e natante abbiamo deciso di procedere per un numero di accessi più contenuto e di optare per la soluzione più efficiente di portare i figli nel padre. Inoltre abbiamo deciso per la specializzazione di **Sovrapprezzo** di portare le figlie nel padre (in previsione di un successivo accorpamento in corsa) ottenendo:

- Una nuova entità **Natante** con al suo interno un nuovo attributo *tipo* che specifica il tipo di natante e gli attributi presenti nella precedente entità **Natante**.
- Una nuova entità **Sovrapprezzo** con al suo interno un nuovo attributo *tipo* che specifica il tipo di sovrapprezzo e gli attributi presenti nella precedente entità Sovrapprezzo.

Invece nelle altre specializzazioni decidiamo di portare il padre nei figli, evitando di inserire attributi inutili o molti valori a null. Inoltre per queste specializzazioni si è optato per queste soluzioni poiché le nostre operazioni sulla base di dati mantengono una netta distinzione fra le suddette specializzazioni. Quindi otterremo:

- Una nuova entità **Compagnia** con al suo interno due nuovi attributi *Login* e Password e gli attributi presenti nella precedente entità Compagnia.
- Una nuova entità **Passeggero** con al suo interno due nuovi attributi *Login* e Password e gli attributi presenti nella precedente entità Passeggero.
- Una nuova entità **Biglietto_Ridotto** con al suo interno gli attributi presenti nella precedente entità **Biglietto**.
- Una nuova entità **Biglietto_Intero** con al suo interno gli attributi presenti nella precedente entità **Biglietto**.

Invece, per l' entità Aggiornamento_Corsa abbiamo deciso di accorpare il padre nelle figlie poiché la specializzazione è totale ed evitiamo di avere valori a "null" inutili (cosa che avremmo avuto se avessimo portato le figlie nel padre). Quindi ciò che otteniamo è:

- Una nuova entità **Ritardo** con al suo interno gli attributi presenti nella precedente entità **Ritardo** e una relazione con Corsa.
- Una nuova entità **Annullamento** con al suo interno gli attributi presenti nella precedente entità **Annullamento** e una relazione con Corsa.

2.5 Partizione/Accorpamento delle associazioni/entità

Si è deciso di decomporre la relazione END POINTS fra CORSA , PORTO E SCALO. Ricaviamo tre diverse associazioni:

1. PARTENZA: lega corsa al porto di partenza;
2. ARRIVO: lega corsa al porto di arrivo;
3. SCALO: lega corsa al porto di scalo.

Ci siamo soffermati poi sulla questione di se accorpare le 3 entità sovrapprezzo derivanti dalle specializzazioni di sovrapprezzo nel passo precedente di eliminazione delle generalizzazioni. Ragionando sulle due possibilità risulta chiara la maniera più saggia di procedere:

1. Non accorpare le entità: Se non accorpiamo ci rendiamo conto che nel modello logico avremmo la seguente mappatura:
   corsa(.....,sovr_veicolo*,sovr_bagaglio*,sovr_prenotazione*) con tre chiavi esterne (nullable). Inoltre dobbiamo anche tenere in storage i vari valori dei sovrapprezzi; in totale abbiamo , ipotizzando 300 diversi sovrapprezzi e 1000 corse a regime, (1000 corse x 4 bytes x 3 chiavi esterne) + (300 sovrapprezzi x 4 bytes)= 12Kb + 1.2Kb + aumento degli accessi.
2. Accorpare le entità: Se accorpiamo osserviamo che abbiamo in corsa sempre 3 attributi float per i tre sovrapprezzi (nullable) e quindi abbiamo sempre (1000 corse x 4 bytes x 3 attributi float) = 12Kb e solo questi.

A questo punto decidiamo di accorpare e di risparmiare memoria e accessi.

2.6 Identificazioni chiavi primarie

- per l'entità **Corsa** è stato scelto di aggiungere un id_corsa in quanto ci sono gruppi di attributi di corsa che rispettano i requisiti di chiave parziale (e che formano con attributi di altre tabelle chiavi candidate) ma hanno troppi attributi all' interno.

- per l'entità **natante** inseriamo un id_natante in quanto il nome di un natante è si una chiave candidata ma è una stringa di max 20 caratteri alfanumerici (piu lenta da confrontare di un id numerico).

- per l'entità **Porto** è stato scelto di aggiungere un id_porto in quanto ci sarebbe un gruppo di attributi come chiave candidate (per esempio:[comune,indirizzo] o anche solo indirizzo), ma si è scelto di introdurre un id_porto numerico per velocità di confronto.

- per l'entità **Biglietto_Intero** è stato scelto di aggiungere un id_biglietto in quanto ci sono gruppi di attributi di Biglietto_Intero che rispettano i requisiti di chiave parziale (e che formano con attributi di altre tabelle chiavi candidate) ma hanno troppi attributi all' interno.

- per l'entità **Biglietto_Ridotto** è stato scelto di aggiungere un id_biglietto in quanto ci sono gruppi di attributi di Biglietto_Ridotto che rispettano i requisiti di chiave parziale (e che formano con attributi di altre tabelle chiavi candidate) ma hanno troppi attributi all' interno.

- per l'entità **Ritardo** è stata scelta la chiave candidata CORSA derivante dall associazione fra ritardo e corsa.

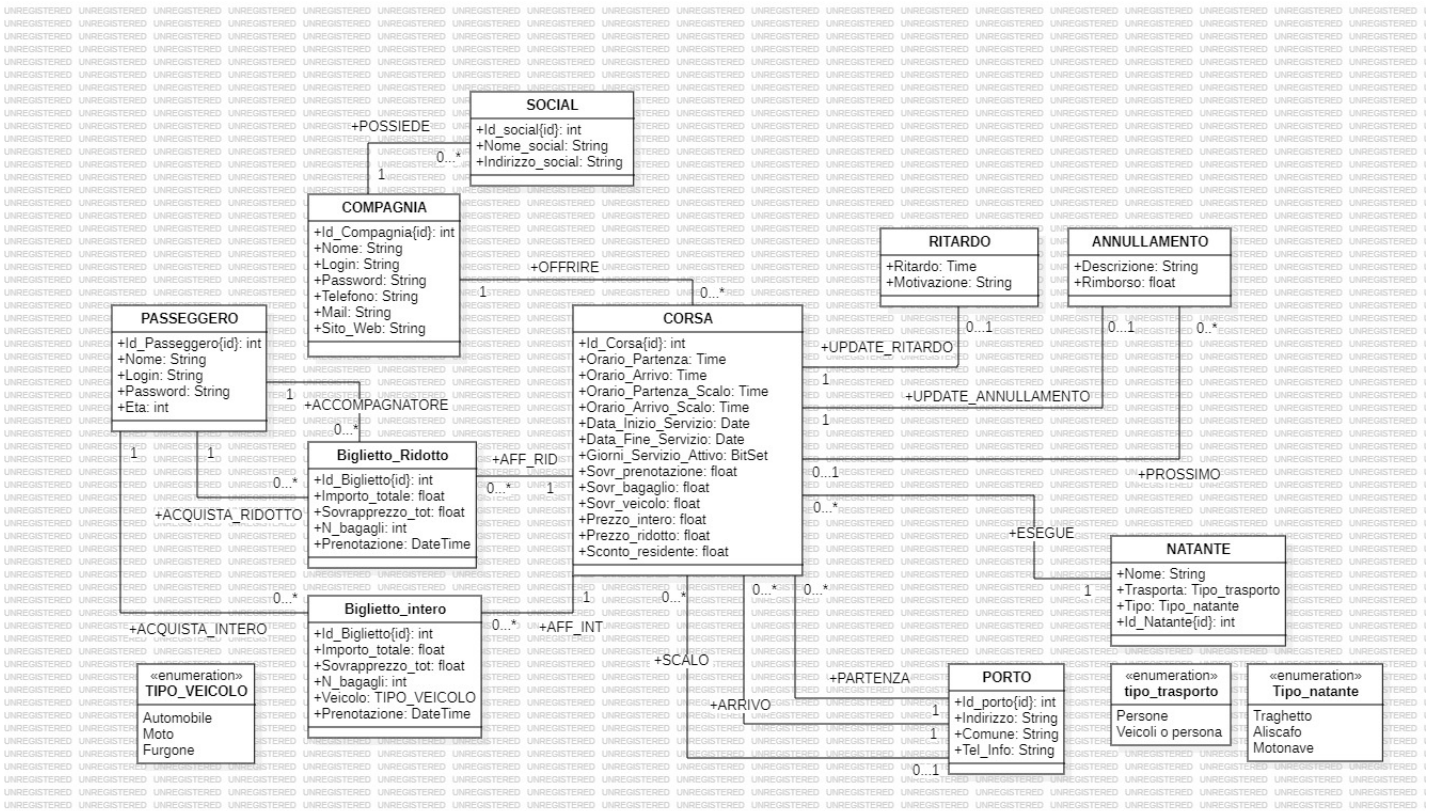
- per l'entità **Annullamento** è stata scelta la chiave candidata CORSA derivante dall associazione fra annullamento e corsa.

- per **Passeggero** inseriamo un id_passeggero in quanto la coppia (login,password) di un passeggero è si una chiave candidata ma è piu lenta da confrontare di un id numerico.

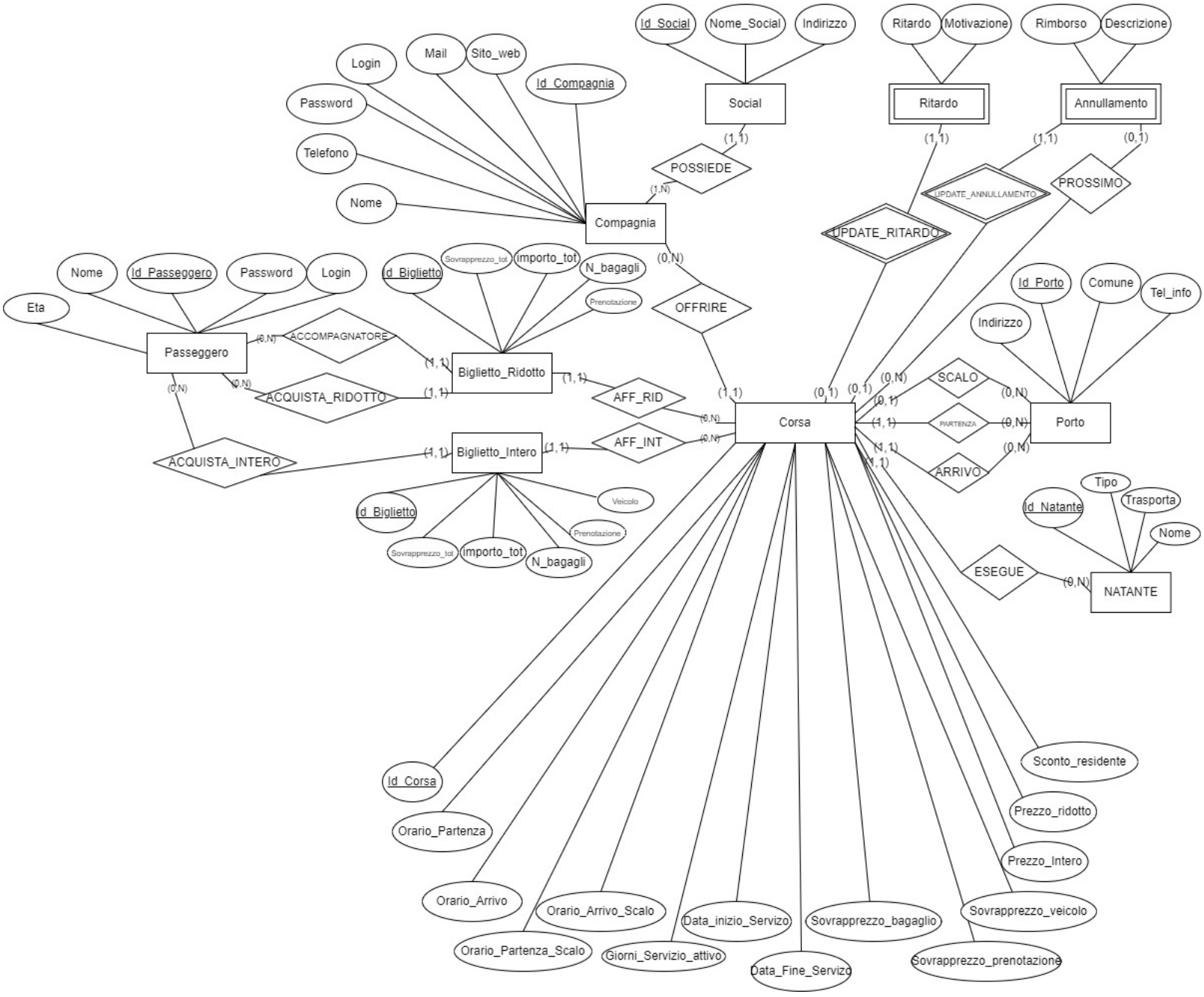
- per **Compagnia** inseriamo un id_compagnia in quanto la coppia (login,password) di una compagnia è si una chiave candidata ma è piu lenta da confrontare di un id numerico.

- per l'entità **Social** è stato scelto di inserire un id_social in quanto indirizzo_social è chiave candidata ma è una stringa di caratteri alfanumerici di lunghezza massimo 60, per cui un id numerico è preferibile per velocità di confronto degli indici.

2.7 Schema ristrutturato UML



2.8 Schema ristrutturato ER



2.9 Dizionario delle entità

Entità	Descrizione	Attributi
Corsa	Elemento cardine che definisce la tratta da un porto ad un altro	**Id_corsa**(int): è l' identificativo della corsa **Orario_Partenza**(Time): Indica l'orario in cui parte la corsa **Orario_Arrivo**(Time): Indica l'orario in cui la corsa arriva **Data_Inizio_Servizio**(Date): Indica il giorno in cui inizia il servizio di corse **Data_Fine_Servizio**(Date):Indica il giorno in cui termina il servizio di corse **Giorni_Servizio_attivo**(Boolean): Indica i giorni in cui il servizio è attivo **Prezzo_Intero**(float): costo del biglietto intero. **Prezzo_Ridotto**(float): costo del biglietto ridotto. **Sconto_Residente**(float): percentuale di sconto per i residenti nel porto di arrivo. **Orario_Partenza_Scalo**(time): Indica l'orario di partenza da porto di scalo **Orario_Arrivo_Scalo**(time): indica l'orario di arrivo al porto di scalo **Sovr_prenotazione**(float): indica il sovrapprezzo prenotazione **Sovr_bagaglio**(float): indica il sovrapprezzo bagaglio **Sovr_veicolo**(float): indica il sovrapprezzo veicolo
Natante	Mezzo con cui si svolge una corsa	**Id_natante**(int): L'identificativo del natante **Nome**(String): Nome del natante utilizzato nella corsa **Trasporta**(String): Chi e/o cosa trasporta il natante, se solo persone o anche veicoli **Tipo**(String): Specifica il tipo del natante
Ritardo	Indica il ritardo che può fare una corsa rispetto all'orario previsto	**Ritardo**(Time): Il tempo di ritardo effettivo della corsa. **Motivazione**(String): Motivazione del ritardo.
Annullamento	Entità che permette di notificare se una corsa è annullata	**Rimborso**(Float): Cifra che viene restituita al passeggero in caso di annullamento della corsa. **Descrizione**(String): Motivazione dell'annullamento.

Entità	Descrizione	Attributi
Porto	Luogo dove stazionano, partono e arrivano i natanti	**Id_porto**(int): L'identificativo del porto **Indirizzo**(String): Indica l'indirizzo del porto **Comune**(String): Indica il comune di appartenenza del porto **Telefono_info**(String): Rappresenta le informazioni telefoniche del porto
Compagnia	Provider che fornisce la corsa per gli utenti	**Id_compagnia**(int): L'identificativo della compagnia **Nome**(String): Rappresenta il nome della compagnia di navigazione **Login**(String): La login della compagnia **Password**(String): La password della compagnia **Telefono**(String): Indica il numero di telefono della compagnia **Mail**(String): Indica l'indirizzo di posta elettronica della compagnia **Sito_web**(String): Indica l'indirizzo del sito web della compagnia
Passeggero	Colui che usufruisce della corsa fornita dalla compagnia	**Id_passeggero**(int):L' identificativo del passeggero **Nome**(String): Nome del passeggero **Login**(String): La login del passeggero **Password**(String): La password del passeggero **Eta**(int): L' età del passeggero
Biglietto_intero	Biglietto utilizzato dal passeggere di maggiore eta per poter acceder ad una corsa	**Id_biglietto**(int):L'identificativo del biglietto **Importo_totale**(float): Indica quanto l'utente ha pagato per la corsa **Sovrapprezzo_totale**(float): Indica l'ammontare dei sovrapprezzi. **N_Bagagli**(Int): Indica il numero di bagagli portati dal passeggero. **Veicolo**(enum) : Indica il tipo di veicolo del passeggero **Prenotazione**(DateTime): Indica data e ora della prenotazione se c'è

Entità	Descrizione	Attributi
Biglietto_ridotto	Biglietto utilizzato dal passeggere di minore eta per poter acceder ad una corsa	**Id_biglietto**(int):L'identificativo del biglietto **Importo_totale**(float): Indica quanto l'utente ha pagato per la corsa **Sovrapprezzo_totale**(float): Indica l'ammontare dei sovrapprezzi. **N_Bagagli**(Int): Indica il numero di bagagli portati dal passeggero. **Prenotazione**(DateTime): Indica data e ora della prenotazione se c'è
Social	Social appartenente ad una compagnia	**Id_social**(int):L'identificativo del social **Nome_social**(String): Il nome del social **Indirizzo_social**(String): L'indirizzo al sito web del social.

2.10 Dizionario delle associazioni

Associazione	**Descrizione**
Offrire	Associazione **uno-a-molti** tra *Corsa* e *Compagnia*. Una corsa può essere offerta da una e una sola Compagnia di navigazione, e una compagnia può offrire una o più corse.
Aff_rid	Associazione **uno-a-molti** tra *Biglietto_ridotto* e *Corsa*. Un Biglietto_ridotto può afferire ad una singola corsa, mentre una corsa può afferire ad uno o più biglietti_ridotti.
Aff_int	Associazione **uno-a-molti** tra *Biglietto_intero* e *Corsa*. Un Biglietto_intero può afferire ad una singola corsa, mentre una corsa può afferire ad uno o più biglietti_interi.
Acquista_ridotto	Associazione **uno-a-molti** tra *Bigliett_ridotto* e *Passeggero*. Un Biglietto_ridotto può essere prenotato da un singolo passeggero, un passeggero però può prenotare uno o più biglietti_ridotti.
Acquista_intero	Associazione **uno-a-molti** tra *Bigliett_intero* e *Passeggero*. Un Biglietto_intero può essere prenotato da un singolo passeggero, un passeggero però può prenotare uno o più biglietti_interi.
Eseguire	Associazione **uno-a-molti** tra *Corsa* e *Natante*. Una corsa può essere eseguita un solo natante. Un natante può eseguire zero o molteplici corse.
Partenza	Associazione **uno-a-molti** tra *Porto* e *Corsa*. Una Corsa ha un porto di partenza , da un porto possono partire zero o più corse.
Arrivo	Associazione **uno-a-molti** tra *Porto* e *Corsa*. Una Corsa ha un porto di arrivo , ad un porto possono arrivare zero o più corse.
Scalo	Associazione **uno-a-molti** tra *Porto* e *Corsa*. Una Corsa puo specificare un porto di partenza , ad un porto possono scalare zero o più corse.
Accompagnatore	Associazione **uno-a-molti**tra *Passeggero* e *Biglietto_Ridotto* Un biglietto_Ridotto può corrispondere ad un solo passeggero accompagnatore .Un passeggero può essere accompagnatore di zero o più possessori di un biglietto ridotto.
Prossimo	Associazione **uno-a-molti** tra *Corsa* e *Annullamento*. Per un annullamento si può indicare una corsa sostitutiva(Prossimo). Una corsa può essere sostitutiva di molteplici annullamenti.
Update_ritardo	Associazione **uno-a-uno** tra *Corsa* e *Ritardo* Una corsa può avere un ritardo ed un ritardo corrisponde ad una singola corsa.
Update_annullamento	Associazione **uno-a-uno** tra *Corsa* e *Annullamento* Una corsa può avere un annullamento ed un annullamento corrisponde ad una singola corsa.
Possedere	Associazione **uno-a-molti** tra *Compagnia* e *Social* Una compagnia puo avere uno o più social, mentre un social corrisponde ad una singola compagnia.

3 Traduzione al modello logico

3.1 Mapping associazioni

3.1.1 Associazioni 1-1

- *Corsa - Update_Ritardo - Ritardo* inserimento di chiave di Corsa in *Ritardo*.
- *Corsa - Update_Annullamento - Annullamento* inserimento di chiave di Corsa in *Annullamento*.

3.1.2 Associazioni 1-N

- *Corsa - Prossimo -Annullamento* inserimento di chiave di Corsa in *Annullamento*
- *Compagnia - Offrire - Corsa* inserimento di chiave di Compagnia in *Corsa*
- *Compagnia - Possiede - Social* inserimento di chiave di Compagnia in *Social*
- *Passeggero - Accompagnatore - Biglietto_Ridotto* inserimento di chiave di Passeggero in *Biglietto_Ridotto*
- *Passeggero - Acquista_Intero - Biglietto_Intero* inserimento di chiave di Passeggero in *Biglietto_Intero*
- *Passeggero - Acquista_Ridotto - Biglietto_Ridotto* inserimento di chiave di Passeggero in *Biglietto_Ridotto*
- *Corsa - Afferenza_Intero - Biglietto_Intero* inserimento di chiave di Corsa in *Biglietto_Intero*
- *Corsa - Afferenza_Ridotto - Biglietto_Ridotto* inserimento di chiave di Corsa in *Biglietto_Ridotto*
- *Corsa - Partenza - Porto* inserimento di chiave di Porto in *Corsa*
- *Corsa - Arrivo - Porto* inserimento di chiave di Porto in *Corsa*

- *Corsa* - *Scalo* - *Porto* inserimento di chiave di Porto in *Corsa*
- *Natante* - *Esegue* - *Corsa* inserimento di chiave di Natante in *Corsa*

3.1.3 Associazioni N-N

Non ci sono associazioni N - N.

3.2 Modello logico

Corsa(Id_Corsa,Orario_Partenza,Orario_Arrivo,Orario_Partenza_Scalo, Orario_Arrivo_Scalo, Data_Inizio_Servizio,Data_Fine_Servizio, Sovr_Prenotazione,Sovr_Bagaglio, Sovr_Veicolo ,Prezzo_Intero,Prezzo_Ridotto,Sconto_Residente,Porto_Partenza↑, Porto_Arrivo↑,Porto_Scalo↑,Compagnia↑,Natante↑)

Compagnia(Id_Compagnia,Nome,Login,Password,Telefono,Mail,Sito_Web)

Social(Id_Social,Compagnia↑,Nome_Social,Indirizzo_Social)

Passeggero(Id_Passeggero,Nome,Login,Password,Eta)

Biglietto_Intero(Id_Biglietto,Importo_Totale,Sovrapprezzo_Totale,N_Bagagli, Veicolo,Prenotazione, Passeggero↑)

Biglietto_Ridotto(Id_Biglietto,Importo_Totale,Sovrapprezzo_Totale,N_Bagagli, Veicolo,Prenotazione, passeggero↑,Accompagnatore↑)

Porto(Id_Porto,Indirizzo, Comune. Tel_Info)

Natante(Id_Natante,Nome,Trasporta,Tipo)

Ritardo(Corsa↑ Ritardo, Motivazione)

Annullamento(Corsa↑, Prossimo↑, Descrizione)

4 Progettazione Fisica

4.1 Vincoli individuati

In questa sottosezione della progettazione fisica andiamo ad individuare i vincoli della base di dati: partiamo dai vincoli di base appartenenti al DDL (Data Definition Language) e andiamo a poi a controllare se abbiamo bisogno di eventuali vincoli di integrità semantica.

4.1.1 Vincoli di base (DDL)

Per la tabella corsa abbiamo implementato una serie di vincoli di **NOT NULL** in modo da assicurarci di avere le informazioni per mantenere integra la nostra base di dati,inoltre abbiamo creato un vincolo di check sul dominio dello sconto_residente per assicurarci che il valore sia sempre compreso fra 0 e 1.

```
1  CREATE TABLE corsa(
2    Id_corsa SERIAL,
3    Orario_partenza time NOT NULL,
4    Orario_arrivo time NOT NULL,
5    Orario_partenza_scalo time ,
6    Orario_arrivo_scalo time,
7    Data_inizio_servizio date NOT NULL,
8    Data_fine_servizio date NOT NULL,
9    Giorni_Servizio_Attivo BIT(7) NOT NULL DEFAULT B'1111111',
10   Sovr_prenotazione float,
11   Sovr_bagaglio float,
12   Sovr_veicolo float,
13   Prezzo_intero float NOT NULL,
14   Prezzo_ridotto float NOT NULL,
15   Sconto_residente sconto,
16   Porto_partenza int NOT NULL,
17   Porto_arrivo int NOT NULL,
18   Porto_scalo int ,
19   Compagnia int NOT NULL,
20   Natante int NOT NULL,
21
22    CONSTRAINT pk_corsa PRIMARY KEY(Id_corsa),
23    CONSTRAINT prezzo_intero_ridotto_check check(
      Prezzo_ridotto<=Prezzo_intero),
```

```
24     CONSTRAINT periodo_check check((Data_inizio_servizio <=
       Data_fine_servizio) AND (Data_fine_servizio > CURRENT_DATE
       )),
25     CONSTRAINT endpoints_check check(Porto_partenza <>
       Porto_arrivo AND Porto_Scalo<>Porto_partenza AND
       Porto_Scalo<>Porto_arrivo),
26     CONSTRAINT scalo_check check((Porto_scalo IS NULL AND
       Orario_partenza_scalo IS NULL AND Orario_arrivo_scalo IS
       NULL) OR (Porto_scalo IS NOT NULL AND
       Orario_partenza_scalo IS NOT NULL AND Orario_arrivo_scalo
       IS NOT NULL)),
27     CONSTRAINT orario_check check(Orario_partenza <=
       Orario_arrivo),
28     CONSTRAINT orario_scalo_check check((Orario_partenza_scalo
        BETWEEN Orario_partenza AND Orario_arrivo) AND (
       Orario_arrivo_scalo BETWEEN Orario_partenza AND
       Orario_arrivo) AND Orario_arrivo_scalo<=
       Orario_partenza_scalo)
29
30 );
31
32 ALTER TABLE corsa ADD CONSTRAINT fk_corsa_porto_partenza
       FOREIGN KEY(Porto_partenza) REFERENCES porto ON DELETE
       CASCADE ON UPDATE CASCADE;
33 ALTER TABLE corsa ADD CONSTRAINT fk_corsa_porto_arrivo
       FOREIGN KEY(Porto_arrivo) REFERENCES porto ON DELETE
       CASCADE ON UPDATE CASCADE;
34 ALTER TABLE corsa ADD CONSTRAINT fk_corsa_porto_scalo FOREIGN
        KEY(Porto_scalo) REFERENCES porto ON DELETE CASCADE ON
       UPDATE CASCADE;
35 ALTER TABLE corsa ADD CONSTRAINT fk_corsa_natante FOREIGN KEY
       (Natante) REFERENCES natante ON DELETE CASCADE ON UPDATE
       CASCADE;
36 ALTER TABLE corsa ADD CONSTRAINT fk_corsa_compagnia FOREIGN
       KEY(Compagnia) REFERENCES compagnia ON DELETE CASCADE ON
       UPDATE CASCADE;
```

Per la tabella porto abbiamo sempre una serie di **NOT NULL** e un vincolo sul dominio di tel_num in modo da assicurarci con una regular expression che il numero di telefono sia nel formato corretto:

```
1  CREATE DOMAIN numero_telefono AS varchar(13) CONSTRAINT
       tel_check check(VALUE SIMILAR TO '\+[0-9]*');
2
3  CREATE TABLE porto(
4    Id_porto SERIAL,
5    Indirizzo varchar(30) NOT NULL,
6    Comune varchar(20) NOT NULL,
7    Tel_info numero_telefono NOT NULL,
8
9
10     CONSTRAINT pk_porto PRIMARY KEY(Id_porto)
11
12 );
```

Per la tabella natante abbiamo una serie di **NOT NULL** e due enumeration create per i tipi particolari tipo_natante e tipo_trasporto:

```
1  CREATE or REPLACE TYPE tipo_trasporto AS enum('persone','
       persone/veicoli');
2  CREATE or REPLACE TYPE tipo_natante AS enum('traghetto','
       aliscafo','motonave');
3
4
5  CREATE TABLE natante(
6    Id_natante SERIAL,
7    Nome varchar(20) UNIQUE NOT NULL,
8    Trasporta tipo_trasporto NOT NULL,
9    Tipo tipo_natante NOT NULL,
10
11
12     CONSTRAINT pk_natante PRIMARY KEY(Id_natante),
13     CONSTRAINT tipo_tipo_trasporto CHECK((tipo_natante='
       traghetto' AND tipo_trasporto='persone/veicoli')or(
       tipo_natante<>'traghetto' AND tipo_trasporto<>'persone/
       veicoli'))
14
15 );
```

Per biglietto ridotto e biglietto intero abbiamo sempre dei **NOT NULL**, e abbiamo creato un altra enumeration per creare un tipo apposito per l attributo veicolo di biglietto_intero:

```
1  CREATE TABLE biglietto_ridotto(
```

```
2   Id_biglietto SERIAL,
3   Importo_totale float NOT NULL,
4   Sovrapprezzo_tot float default 0,
5   N_bagagli int default 0,
6   Prenotazione timestamp,
7   Corsa int NOT NULL,
8   Passeggero int NOT NULL,
9   Accompagnatore int NOT NULL,
10
11
12  CONSTRAINT pk_biglietto_r PRIMARY KEY(Id_biglietto)
13
14 );
15
16 CREATE or REPLACE TYPE tipo_veicolo AS enum('automobile','
       moto','furgone');
17
18 CREATE TABLE biglietto_intero(
19   Id_biglietto SERIAL,
20   Importo_totale float NOT NULL,
21   Sovrapprezzo_tot float default 0,
22   N_bagagli int default 0,
23   Veicolo tipo_veicolo,
24   Prenotazione timestamp,
25   Corsa int NOT NULL,
26   Passeggero int NOT NULL,
27
28
29
30    CONSTRAINT pk_biglietto_i PRIMARY KEY(Id_biglietto)
31
32 );
33
34 ALTER TABLE biglietto_ridotto ADD CONSTRAINT
       fk_biglietto_ridotto_corsa FOREIGN KEY(Corsa) REFERENCES
       corsa ON DELETE CASCADE ON UPDATE CASCADE;
35 ALTER TABLE biglietto_ridotto ADD CONSTRAINT
       fk_biglietto_ridotto_passeggero FOREIGN KEY(Passeggero)
       REFERENCES passeggero ON DELETE CASCADE ON UPDATE CASCADE;
36 ALTER TABLE biglietto_ridotto ADD CONSTRAINT
       fk_biglietto_ridotto_accompagnatore FOREIGN KEY(
       Accompagnatore) REFERENCES passeggero ON DELETE CASCADE ON
        UPDATE CASCADE;
37 ALTER TABLE biglietto_intero ADD CONSTRAINT
       fk_biglietto_intero_corsa FOREIGN KEY(Corsa) REFERENCES
```

```
    corsa ON DELETE CASCADE ON UPDATE CASCADE;
38 ALTER TABLE biglietto_intero ADD CONSTRAINT
    fk_biglietto_intero_passeggero FOREIGN KEY(Passeggero)
    REFERENCES passeggero ON DELETE CASCADE ON UPDATE CASCADE;
```

per la tabella passeggero abbiamo dei **NOT NULL** e un vincolo di **UNIQUE** sulla coppia (login,password):

```
1  CREATE TABLE passeggero(
2    Id_passeggero SERIAL,
3    Nome varchar(20) NOT NULL,
4    Login varchar(30) NOT NULL,
5    Password varchar(30) NOT NULL,
6    Eta int NOT NULL,
7
8     CONSTRAINT pk_passeggero PRIMARY KEY(Id_passeggero),
9     CONSTRAINT unique_login_password_passeggero UNIQUE(Login,
     Password)
10
11 );
```

per la tabella compagnia abbiamo dei **NOT NULL** e un vincolo di **UNIQUE** sulla coppia (login,password):

```
1  CREATE TABLE compagnia(
2    Id_compagnia SERIAL,
3    Nome varchar(20) NOT NULL UNIQUE,
4    Login varchar(30) NOT NULL,
5    Password varchar(30) NOT NULL,
6    Telefono numero_telefono NOT NULL,
7    Mail varchar(30) NOT NULL,
8    Sito_web varchar(30) NOT NULL,
9
10    CONSTRAINT pk_compagnia PRIMARY KEY(Id_compagnia),
11    CONSTRAINT unique_login_password_compagnia UNIQUE(Login,
     Password)
12
13 );
```

Per la tabella social abbiamo dei **NOT NULL**

```
1  CREATE TABLE social(
2    Id_social SERIAL,
3    Compagnia int NOT NULL,
4    Nome_social varchar(10) NOT NULL,
5    Indirizzo_social varchar(60) NOT NULL,
6
7
```

```
8     CONSTRAINT pk_social PRIMARY KEY(Id_social)
9
10 );
11 ALTER TABLE social ADD CONSTRAINT fk_social_compagnia FOREIGN
       KEY(Compagnia) REFERENCES compagnia ON DELETE CASCADE ON
     UPDATE CASCADE;
```

Per la tabella ritardo e annullamento abbiamo una serie di **NOT NULL**:

```
1  CREATE TABLE ritardo(
2    Corsa SERIAL,
3    Ritardo time NOT NULL,
4    Motivazione text,
5
6
7     CONSTRAINT pk_ritardo PRIMARY KEY(Corsa)
8
9  );
10
11
12 CREATE TABLE annullamento(
13   Corsa SERIAL,
14   Prossimo int,
15   Rimborso float,
16   Descrizione text,
17
18
19    CONSTRAINT pk_annullamento PRIMARY KEY(Corsa)
20
21 );
22 ALTER TABLE ritardo ADD CONSTRAINT fk_ritardo_corsa FOREIGN
       KEY(Corsa) REFERENCES corsa ON DELETE CASCADE ON UPDATE
       CASCADE;
23
24
25 ALTER TABLE annullamento ADD CONSTRAINT fk_annullamento_corsa
       FOREIGN KEY(Corsa) REFERENCES corsa ON DELETE CASCADE ON
     UPDATE CASCADE;
26 ALTER TABLE annullamento ADD CONSTRAINT
     fk_annullamento_prossimo FOREIGN KEY(Prossimo) REFERENCES
     corsa ON DELETE SET NULL ON UPDATE CASCADE;
```

4.1.2 Vincoli di integrità semantica

Non vogliamo che nella nostra base di dati un natante possa essere scelto per due corse diverse nello stesso giorno ,nello stesso periodo e negli stessi orari; per cui andiamo ad implementare un trigger che controlla prima di ogni inserimento che non si verifichino le suddette condizioni:

```
1  CREATE or REPLACE FUNCTION check_corsa() RETURNS trigger AS
       $check_corsa$
2
3       BEGIN
4
5          /*il trigger verifica se esiste gia un altra corsa per
        il natante tale che le corse hanno periodi e orari in
       overlapping, se esiste lancia un arrore per notificarlo*/
6
7           IF(EXISTS(SELECT * FROM corsa WHERE natante=new.
       natante AND (CAST(new.giorni_servizio_attivo &
       giorni_servizio_attivo AS VARCHAR(7)) LIKE '%1%')
8           AND (Data_inizio_Servizio<=new.Data_fine_servizio AND
        Data_fine_servizio>=new.Data_inizio_Servizio) AND
9           (Orario_partenza<=new.Orario_arrivo AND Orario_arrivo
       >=new.Orario_partenza))) THEN
10
11              RAISE EXCEPTION 'il natante specificato per
       questa corsa esegue gia un altra corsa nelle date, nei
       giorni e negli orari specificati';
12
13          END IF;
14
15
16
17          RETURN NEW;
18      END;
19 $check_corsa$ LANGUAGE plpgsql;
20
21 CREATE or REPLACE TRIGGER check_corsa
22 BEFORE INSERT OR UPDATE ON corsa
23 FOR EACH ROW
24 EXECUTE FUNCTION check_corsa();
```

Inoltre non vogliamo che quando si inserisce un annullamento manualmente specificando la prossima corsa questa corsa non sia realmente valida come sostituta:

```
1  CREATE or REPLACE FUNCTION check_prossimo() RETURNS trigger
       AS $check_prossimo$
2
3       DECLARE
4
5           ann_partenza corsa.porto_partenza%TYPE;
6           ann_arrivo corsa.porto_arrivo%TYPE;
7       ann_orario_partenza corsa.orario_partenza%TYPE;
8           day_of_week INTEGER;
9
10
11      BEGIN
12
13          /*seleziona il giorno della settimana corrente,
        abbiamo bisogno del modulo perche extract restituisce i
        giorni da 0 a 6 partendo da domenica*/
14
15          day_of_week = mod(extract( dow from CURRENT_DATE)
        +6,7) +1;
16
17
18
19          /*seleziono la corsa annullata*/
20
21          SELECT porto_partenza,porto_arrivo,orario_partenza
        INTO ann_partenza,ann_arrivo,ann_orario_partenza FROM
        corsa WHERE id_corsa=new.corsa;
22
23          /*se la prossima corsa impostata non e' nelle corse
        che partono dallo stesso punto , arrivano nello stesso
        punto, disponibili nel giorno corrente , nella data
        corrente ad un orario piu tardo allora lancia un errore
        per notificarlo*/
24
25          IF(new.prossimo NOT IN (SELECT id_corsa FROM corsa
        WHERE SUBSTRING(Giorni_Servizio_Attivo,day_of_week+1,1) =
        '1' AND porto_partenza=ann_partenza AND porto_arrivo=
        ann_arrivo AND orario_partenza>ann_orario_partenza AND(
        CURRENT_DATE BETWEEN Data_inizio_servizio AND
        Data_fine_servizio))) THEN
26
27              RAISE EXCEPTION 'la corsa sostitituiva
        selezionata non e'' valida';
28          END IF;
29
```

```
30
31          RETURN new;
32      END;
33 $check_prossimo$ LANGUAGE plpgsql;
34
35 CREATE or REPLACE TRIGGER check_prossimo
36 BEFORE INSERT OR UPDATE OF corsa ON annullamento
37 FOR EACH ROW
38 WHEN (new.prossimo IS NOT NULL)
39 EXECUTE FUNCTION check_prossimo();
```

Per le tabelle biglietto_intero e biglietto_ridotto non vogliamo che ci siano casi di inserimenti in cui un biglietto_ridotto è acquistato da una persona di maggiore età e un biglietto_intero è acquistato da una persona di minore età. Per cui inseriamo due trigger:

```
1  CREATE or REPLACE FUNCTION check_intero() RETURNS trigger AS
       $check_intero$
2
3       DECLARE
4
5          eta_passeggero integer;
6
7       BEGIN
8
9           /*il trigger controlla che il passeggero abbia
       effettivamente piu o almeno 16 anni, se non vero allora lo
        notifica con un errore*/
10
11          SELECT eta INTO eta_passeggero FROM passeggero WHERE
       Id_passeggero=new.passeggero;
12
13          IF(eta_passeggero<16) THEN
14              RAISE EXCEPTION 'Un passeggero di eta minore non
       puo acquistare un biglietto intero';
15          END IF;
16
17
18          RETURN new;
19      END;
20 $check_intero$ LANGUAGE plpgsql;
21
22 CREATE or REPLACE TRIGGER check_intero
23 BEFORE INSERT OR UPDATE OF corsa ON biglietto_intero
24 FOR EACH ROW
25 EXECUTE FUNCTION check_intero();
```

E il secondo (controlla anche che l accompagnatore abbia piu o almeno 16 anni e controlla che l accompagnatore abbia prima acquisato un biglietto per la stessa corsa):

```
1  CREATE or REPLACE FUNCTION check_ridotto() RETURNS trigger AS
       $check_ridotto$
2
3      DECLARE
4
5         eta_passeggero integer;
6         eta_accompagnatore integer;
7
8      BEGIN
9
10         /*il trigger controlla che il passeggero abbia
       effettivamente meno di 16 anni e che l 'accompagnatore
       abbia piu o almeno 16 anni, se non vero allora lo notifica
        con un errore*/
11
12         SELECT eta INTO eta_passeggero FROM passeggero WHERE
       Id_passeggero=new.passeggero;
13         SELECT eta INTO eta_accompagnatore FROM passeggero
       WHERE Id_passeggero=new.Accompagnatore;
14
15         /*controlla inoltre prima che l'accompagnatore abbia
       comprato un biglietto per la stessa corsa*/
16
17         IF (new.accompagnatore NOT IN (SELECT passeggero FROM
        biglietto_intero WHERE corsa=new.corsa)) THEN
18             RAISE EXCEPTION 'l'' accompagnatore deve prima
       acquistare il biglietto per la stessa corsa';
19         END IF;
20
21         IF(eta_accompagnatore<16) THEN
22             RAISE EXCEPTION 'l'' accompagnatore non puo avere
        meno di 16 anni';
23         END IF;
24
25
26         IF(eta_passeggero>=16) THEN
27             RAISE EXCEPTION 'Un passeggero di eta maggiore
       non puo acquistare un biglietto ridotto';
28         END IF;
29
30
```

```
31          RETURN new;
32      END;
33 $check_ridotto$ LANGUAGE plpgsql;
34
35 CREATE or REPLACE TRIGGER check_ridotto
36 BEFORE INSERT OR UPDATE OF corsa ON biglietto_ridotto
37 FOR EACH ROW
38 EXECUTE FUNCTION check_ridotto();
```

4.2 Creazione Viste

Nella nostra progettazione abbiamo ritenuto utile creare le seguenti viste:

- VISTA_CORSA_PORTI_COMPAGNIA_AGGIORNAMENTO: è la vista da cui prende una delle funzioni cardine del progetto (la funzione filtra_corse).
- VISTA_CORSA_PORTI_NATANTE: è la vista da cui prendono i dati alcune funzioni come la funzione retrieve_corse_compagnia.

Di seguito il codice:

```
1 CREATE or REPLACE VIEW
    VISTA_CORSA_PORTI_COMPAGNIA_AGGIORNAMENTO(Id_corsa,
    Orario_partenza,
2   Orario_arrivo,Orario_partenza_scalo,Orario_arrivo_scalo,
    Data_inizio_servizio,Data_fine_servizio,
    Giorni_Servizio_Attivo,Sovr_prenotazione,Sovr_bagaglio,
    Sovr_veicolo, Prezzo_intero,Prezzo_ridotto,
    Sconto_residente,Porto_partenza, Porto_arrivo,Porto_scalo,
    Compagnia,Natante,P1_id_porto,P1_indirizzo,P1_comune,
    P1_tel_info,P2_id_porto,P2_indirizzo,P2_comune,P2_tel_info
    ,PS_id_porto,PS_indirizzo,PS_comune,PS_tel_info,
    C_compagnia,C_nome,C_login,C_password,C_telefono,C_mail,
    C_sito_web,
3 R_corsa,R_ritardo,R_motivazione,A_corsa,A_prossimo,A_rimborso
    ,A_descrizione)AS
4
5 SELECT * FROM corsa JOIN porto AS p ON p.id_porto=
    porto_partenza JOIN porto AS a ON a.id_porto=porto_arrivo
    LEFT JOIN porto AS s ON s.id_porto=porto_scalo JOIN
    compagnia ON compagnia=id_compagnia LEFT JOIN ritardo AS r
     ON r.corsa=id_corsa LEFT JOIN annullamento AS an ON an.
    corsa=id_corsa JOIN natante ON natante=id_natante
```

```
6
7
8  CREATE or REPLACE VIEW VISTA_CORSA_PORTI_NATANTE(Id_corsa,
       Orario_partenza,
9    Orario_arrivo,Orario_partenza_scalo,Orario_arrivo_scalo,
       Data_inizio_servizio,Data_fine_servizio,
       Giorni_Servizio_Attivo,Sovr_prenotazione,Sovr_bagaglio,
10   Sovr_veicolo, Prezzo_intero,Prezzo_ridotto,Sconto_residente
       ,porto_partenza,porto_arrivo,porto_scalo,compagnia,natante
       ,P1_id_porto,P1_indirizzo,P1_comune,P1_tel_info,
11   P2_id_porto,P2_indirizzo,P2_comune,P2_tel_info,ps_id_porto,
       Ps_indirizzo,Ps_comune,Ps_tel_info,id_natante,nome_natante
       ,trasporta,tipo_natante) AS
12
13   SELECT * FROM corsa JOIN porto as p ON p.id_porto=
       porto_partenza JOIN porto AS a ON a.id_porto=porto_arrivo
       LEFT JOIN porto AS s ON s.id_porto=porto_scalo JOIN
       natante as n ON n.id_natante=natante;
```

4.3 Funzioni SQL individuate

Sono state individuate per la gestione della base di dati le seguenti funzioni scritte con il linguaggio procedurale di postgresql (plpgsql):

4.3.1 filtra_corse(attributi di filtraggio)

La funzione filtra_corse serve al filtraggio e al recupero delle corse cercate dal passeggero.

```
1  CREATE OR REPLACE FUNCTION filtra_corse(id_corsa int ,
       portoPartenza VARCHAR(100) , portoArrivo VARCHAR(100) ,
       dataPartenza DATE , orarioPartenza TIME ,prezzo float ,
       tipo_natante VARCHAR(100) ) RETURNS refcursor AS $$
2  DECLARE
3    corse_cursor refcursor;
4    comando VARCHAR(1000)='SELECT * FROM
       vista_corsa_porti_compagnia_aggiornamento';
5    day_of_week INTEGER;
6  BEGIN
7
```

```
8   /*Se qualcuno degli attributi non e' null allora vuol dire
     che devo filtrare per uno o piu parametri quindi aggiungo
     un WHERE*/
9
10  IF portoPartenza IS NOT NULL or portoArrivo IS NOT NULL or
     dataPartenza IS NOT NULL or orarioPartenza IS NOT NULL or
     prezzo IS NOT NULL or tipo_natante IS NOT NULL THEN
11
12    comando= comando || ' WHERE ';
13
14    /*A questo punto capisco per quali attributi devo
     filtrare e modifico la query di conseguenza*/
15
16    IF id_corsa IS NOT NULL THEN
17      comando=comando || 'id_corsa=' || id_corsa || ' AND ';
18    END IF;
19
20    IF portoPartenza IS NOT NULL THEN
21      comando = comando || 'Porto_partenza IN (SELECT
     id_porto FROM porto WHERE Comune=''' ||portoPartenza ||'''
     ) AND ';
22
23    END IF;
24
25
26    IF portoArrivo IS NOT NULL THEN
27      comando = comando || 'Porto_arrivo IN (SELECT id_porto
     FROM porto WHERE Comune=''' || portoArrivo || ''') AND ';
28
29    END IF;
30
31    IF dataPartenza IS NOT NULL THEN
32      day_of_week = select mod(extract( dow from DATE
     dataPartenza) +6,7) +1;
33      comando = comando || '('''||dataPartenza||''' BETWEEN
     Data_inizio_servizio AND Data_fine_servizio) AND SUBSTRING
     (Giorni_Servizio_Attivo,'||day_of_week||'+1,1)= ''1'' AND'
     ;
34
35        END IF;
36
37    IF orarioPartenza IS NOT NULL THEN
38      comando = comando || 'Orario_partenza >= ''' ||
     orarioPartenza || ''' AND ';
39
```

```
40      END IF;
41
42      IF prezzo IS NOT NULL THEN
43        comando = comando || 'Prezzo_intero <= ' || prezzo || '
        AND ';
44
45          END IF;
46
47      IF tipo_natante IS NOT NULL THEN
48        comando = comando || '''' || tipo_natante ||''' =(
       SELECT tipo FROM natante WHERE Id_natante = Natante) AND '
       ;
49
50      END IF;
51
52      /*ora mi sara' rimasto un AND ballerino finale + uno
       spazio ,quindi li elimino operando un comando di substring
       */
53
54      comando=SUBSTRING(comando,1,LENGTH(comando)-4) || ';';
55
56     END IF;
57
58    OPEN corse_cursor FOR EXECUTE comando;
59    RETURN corse_cursor;
60
61
62 END;
63 $$ LANGUAGE plpgsql;
```

4.3.2 login_passeggero(login,password)

La funzione sottostante è la funzione che implementa la funzionalità di login per il passeggero:

```
1 CREATE OR REPLACE FUNCTION loginPasseggero(login VARCHAR(10),
     password VARCHAR(10)) RETURNS refcursor AS $$
2
3 DECLARE
4   command VARCHAR(100);
5
6   res refcursor;
7
8 BEGIN
```

```
9     /*Seleziona il passeggero con login e password
      corrispondenti e restituisce il cursore (potenzialmente
      vuoto)*/
10    command='SELECT * FROM passeggero WHERE login=''' || login
      || ''' AND ' || 'password=''' || password || '''';
11    OPEN res FOR EXECUTE command;
12    RETURN res;
13
14 END;
15 $$ LANGUAGE plpgsql;
```

4.3.3 login_compagnia(login,password)

La funzione sottostante è la funzione che implementa la funzionalità di login per la compagnia:

```
1  CREATE OR REPLACE FUNCTION loginCompagnia(login VARCHAR(10),
      password VARCHAR(10)) RETURNS refcursor AS $$
2
3  DECLARE
4    command VARCHAR(100);
5
6    res refcursor;
7
8  BEGIN
9     /*Seleziona la compagnia con login e password
      corrispondenti e restituisce il cursore (potenzialmente
      vuoto)*/
10   command='SELECT * FROM compagnia WHERE login=''' || login
      || ''' AND ' || 'password=''' || password || '''';
11   OPEN res FOR EXECUTE command;
12   RETURN res;
13
14 END;
15 $$ LANGUAGE plpgsql;
```

4.3.4 retrieve_corsa(id_compagnia)

Questa funzione serve alla compagnia per recuperare tutte le sue corse:

```
1  CREATE or REPLACE FUNCTION retrieve_corse(id_compagnia
      INTEGER) RETURNS refcursor AS
```

```
2 $$
3
4     DECLARE
5         cursore_corse refcursor;
6
7     BEGIN
8         /*se la compagnia con l id fornito non esiste allora
      lancia un errore per notificarlo*/
9         IF(NOT EXISTS(SELECT * FROM compagnia as c WHERE p.
      Id_compagnia=id_compagnia)) THEN
10
11              RAISE EXCEPTION 'la compagnia con questo id non
      esiste';
12
13        END IF;
14
15        /*altrimenti procedi a recuperare tutte le corse
      della compagnia*/
16
17        OPEN cursore_corse FOR SELECT * FROM
      vista_corsa_porti_natante WHERE compagnia=id_compagnia;
18
19        RETURN cursore_corse;
20
21    END;
22
23 $$LANGUAGE plpgsql;
```

4.3.5 retrieve_accompagnatori()

Questa funzione serve al passeggero di minore eta per scegliere un accompagnatore per il viaggio, in particolare recupera tutti i passeggeri di maggiore eta:

```
1 CREATE or REPLACE FUNCTION retrieve_accompagnatori() RETURNS
      refcursor AS $$
2
3     DECLARE
4         cursore_passeggeri refcursor;
5     BEGIN
6         OPEN cursore_passeggeri FOR SELECT * FROM passeggero
      WHERE Eta >= 16;
7
```

```
8         RETURN cursore_passeggeri;
9     END;
10
11
12 $$ LANGUAGE plpgsql;
```

4.3.6 retrieve_biglietti_interi(id_passeggero)

Questa funzione serve per recuperare tutti i biglietti di un passeggero di maggiore eta (quindi biglietti interi):

```
1 CREATE or REPLACE FUNCTION retrieve_biglietti_interi(
    id_passeggero INTEGER) RETURNS refcursor AS
2 $$
3
4     DECLARE
5         cursore_biglietti refcursor;
6
7     BEGIN
8         IF(NOT EXISTS(SELECT * FROM passeggero as p WHERE p.
    Id_passeggero=id_passeggero)) THEN
9
10            RAISE EXCEPTION 'l'' utente con questo id non
    esiste';
11
12        END IF;
13
14
15        OPEN cursore_biglietti FOR SELECT * FROM
    biglietto_intero WHERE Passeggero=id_passeggero;
16
17        RETURN cursore_biglietti;
18
19    END;
20
21 $$LANGUAGE plpgsql;
```

4.3.7 retrieve_biglietti_ridotti(id_passeggero)

Questa funzione serve per recuperare tutti i biglietti di un passeggero di minore eta (quindi biglietti ridotti):

```
1  CREATE or REPLACE FUNCTION retrieve_biglietti_ridotti(
       id_passeggero INTEGER) RETURNS refcursor AS
2  $$
3
4      DECLARE
5          cursore_biglietti refcursor;
6
7      BEGIN
8          /*se non esiste un utente con l id fornito allora
       lancia un errore per notificarlo,altrimenti procedi al
       recupero dei biglietti*/
9          IF(NOT EXISTS(SELECT * FROM passeggero as p WHERE p.
       Id_passeggero=id_passeggero)) THEN
10
11             RAISE EXCEPTION 'l'' utente con questo id non
       esiste';
12
13         END IF;
14
15
16         OPEN cursore_biglietti FOR SELECT * FROM
       biglietto_ridotto WHERE Passeggero=id_passeggero;
17
18         RETURN cursore_biglietti;
19
20     END;
21
22 $$LANGUAGE plpgsql;
```

4.3.8 retrieve_natanti()

Questa funzione serve per recuperare tutti i natanti in modo che una compagnia possa poi selezionare un natante per creare una nuova corsa :

```
1  CREATE or REPLACE FUNCTION retrieve_natanti() RETURNS
       refcursor AS $$
2      DECLARE
3          cursore_natanti refcursor;
4      BEGIN
5          OPEN cursore_natanti FOR SELECT * FROM natante;
6          RETURN cursore_natanti;
7      END;
8
9  $$LANGUAGE plpgsql;
```

4.3.9 retrieve_passeggero(id_passeggero)

Questa funzione serve per recuperare i dati di un passeggero specifico :

```
1  CREATE or REPLACE FUNCTION retrieve_passeggero(id_passeggero
       INTEGER) RETURNS refcursor AS $$
2
3      DECLARE
4          cursore_passeggeri refcursor;
5      BEGIN
6          OPEN cursore_passeggeri FOR SELECT * FROM passeggero
       AS p WHERE p.id_passeggero=id_passeggero;
7          RETURN cursore_passeggeri;
8      END;
9
10 $$LANGUAGE plpgsql;
```

4.3.10 retrieve_porti()

Questa funzione serve per recuperare i dati di tutti i porti in modo che la compagnia possa scegliere determinati porti per partenza,arrivo e scalo di una corsa :

```
1  CREATE or REPLACE FUNCTION retrieve_porti() RETURNS refcursor
       AS $$
2      DECLARE
3          cursore_porti refcursor;
4      BEGIN
5          OPEN cursore_porti FOR SELECT * FROM porto;
6          RETURN cursore_porti;
7      END;
8
9  $$LANGUAGE plpgsql;
```

4.3.11 retrieve_social(id_compagnia)

Questa funzione serve per recuperare i dati di tutti i social di una determinata compagnia:

```
1  CREATE or REPLACE FUNCTION retrieve_social(id_compagnia
       INTEGER) RETURNS refcursor AS $$
```

```
2     DECLARE
3         cursore_social refcursor;
4     BEGIN
5         OPEN cursore_social FOR SELECT * FROM social WHERE
      compagnia=id_compagnia;
6         RETURN cursore_social;
7     END;
8
9 $$LANGUAGE plpgsql;
```

4.4 Procedure SQL individuate

Sono state individuate per la gestione della base di dati le seguenti procedure scritte con il linguaggio procedurale di postgresql (plpgsql):

4.4.1 register_passeggero(nome,login,password,eta)

La procedura sottostante è la funzione che implementa la funzionalità di registrazione per il passeggero:

```
1 CREATE or REPLACE  PROCEDURE register_passeggero(nome VARCHAR
      (100),login VARCHAR(100), password VARCHAR(100),eta
      INTEGER)
2 AS $$
3     DECLARE
4
5     BEGIN
6         /*Se esiste gia un utente con le credenziali fornite
      allora viene lanciato un errore che lo notifica*/
7         IF(EXISTS(SELECT * FROM passeggero AS p WHERE p.Login
      =login AND p.Password=login)) THEN
8             RAISE EXCEPTION 'l'' utente con queste
      credenziali esiste gia';
9         END IF;
10
11        INSERT INTO passeggero VALUES (nome,login,password,
      eta);
12    END;
13
14 $$ LANGUAGE plpgsql;
```

4.4.2 change_login(old_login,new_login,password)

Questa procedura serve al passeggero per cambiare la propria login:

```
1  CREATE or REPLACE PROCEDURE change_login(old_login VARCHAR
       (1000),new_login VARCHAR(1000),password_in VARCHAR(1000))
       AS $$
2
3       DECLARE
4
5       BEGIN
6           /*Se esiste l utente corrispondente con le
       credenziali fornite allora si puo procedere ad aggiornarle
       ,altrimenti lancia un errore*/
7           IF(EXISTS(SELECT * FROM passeggero WHERE  password=
       password_in AND login=old_login)) THEN
8               UPDATE passeggero SET login=new_login WHERE
       password=password_in AND login=old_login;
9           ELSE
10              RAISE EXCEPTION 'il passeggero con questa
       password e questa login non esiste';
11          END IF;
12
13
14      END;
15
16 $$ LANGUAGE plpgsql;
```

4.4.3 change_password(login,password,new_password)

Questa procedura serve al passeggero per cambiare la propria password:

```
1  CREATE or REPLACE PROCEDURE change_password(login_in VARCHAR
       (1000),new_password VARCHAR(1000),old_password VARCHAR
       (1000)) AS $$
2
3       DECLARE
4
5       BEGIN
6           /*Se esiste l utente corrispondente con le
       credenziali fornite allora si puo procedere ad aggiornarle
       ,altrimenti lancia un errore*/
7           IF(EXISTS(SELECT * FROM passeggero WHERE login=
       login_in AND password=old_password)) THEN
```

```
8             UPDATE passeggero SET password=new_password WHERE
      login=login_in AND password=old_password;
9
10        ELSE
11            RAISE EXCEPTION 'il passeggero con questa login e
      questa password non esiste';
12        END IF;
13
14
15    END;
16
17 $$ LANGUAGE plpgsql;
```

4.4.4 create_update_corsa(attributi di corsa)

Questa procedura serve alla compagnia per creare una nuova corsa o aggiornare i dati di una corsa esistente:

```
1 CREATE or REPLACE PROCEDURE create_update_corsa(Id_corsa int,
2   Orario_partenza time ,
3   Orario_arrivo time ,
4   Data_inizio_servizio date ,
5   Data_fine_servizio date ,
6   Giorni_Servizio_Attivo BIT(7) ,
7   Sovr_prenotazione float,
8   Sovr_bagaglio float,
9   Sovr_veicolo float,
10  Prezzo_intero float ,
11  Prezzo_ridotto float ,
12  Sconto_residente float,
13  Porto_partenza int ,
14  Porto_arrivo int ,
15  Compagnia int ,
16  Natante int ) AS $$
17
18    DECLARE
19
20    BEGIN
21        /*Se l id corsa non e' null ed esiste la corsa con
      quell id allora procedi con un UDPATE dei dati*/
22        IF (Id_corsa IS NOT NULL AND EXISTS(SELECT * FROM
      corsa AS c WHERE c.id_corsa=Id_corsa)) THEN
23            UPDATE corsa AS c SET c.Orario_partenza=
      Orario_partenza, c.Orario_arrivo=Orario_arrivo, c.
```

```
    Data_inizio_Servizio=Data_inizio_Servizio, c.
    Data_fine_servizio=Data_fine_servizio, c.
    Giorni_Servizio_Attivo=Giorni_Servizio_Attivo, c.
    Sovr_prenotazione=Sovr_prenotazione, c.Sovr_bagaglio=
    Sovr_bagaglio, c.Sovr_veicolo=Sovr_veicolo, c.
    Prezzo_intero=Prezzo_intero, c.Prezzo_ridotto=c.
    Prezzo_ridotto, c.Sconto_residente=Sconto_residente, c.
    Porto_partenza=Porto_partenza, c.Porto_arrivo=Porto_arrivo
    , c.Compagnia=Compagnia,c.Natante=Natante
24              WHERE c.id_corsa=Id_corsa;
25
26          /*altrimenti crea una nuova corsa con i dati forniti
    */
27          ELSE
28              INSERT INTO corsa(Orario_partenza, Orario_arrivo,
     Data_inizio_Servizio, Data_fine_servizio,
    Giorni_Servizio_Attivo,Sovr_prenotazione,Sovr_bagaglio,
    Sovr_veicolo,Prezzo_intero,Prezzo_ridotto,Sconto_residente
    ,Porto_partenza,Porto_arrivo,Compagnia,Natante)
29              VALUES(Orario_partenza, Orario_arrivo,
    Data_inizio_Servizio, Data_fine_servizio,
    Giorni_Servizio_Attivo,Sovr_prenotazione,Sovr_bagaglio,
    Sovr_veicolo,Prezzo_intero,Prezzo_ridotto,Sconto_residente
    ,Porto_partenza,Porto_arrivo,Compagnia,Natante);
30          END IF;
31
32
33      END;
34
35 $$ LANGUAGE plpgsql;
```

4.4.5 delete_corsa(id_corsa)

Questa procedura serve alla compagnia per eliminare una corsa esistente:

```
1 CREATE or REPLACE PROCEDURE delete_corsa(id_corsa INTEGER) AS
      $$
2     DECLARE
3
4     BEGIN
5         DELETE FROM corsa AS c WHERE c.id_corsa=id_corsa;
6     END;
7 $$LANGUAGE plpgsql;
```

4.4.6 add_annullamento(attributi di annullamento)

Questa procedura serve per aggiungere un annullamento ad una corsa (annullarla):

```
1  CREATE or REPLACE PROCEDURE add_annullamento(descrizione_in
       VARCHAR(1000),rimborso_in FLOAT,id_corsa INTEGER,
       id_prossimo INTEGER) AS $$
2
3      DECLARE
4
5      BEGIN
6          /*se la corsa indicata non esiste lancia un errore
       per notificarlo*/
7          IF (NOT EXISTS(SELECT * FROM corsa AS c WHERE c.
       id_corsa=id_corsa)) THEN
8              RAISE EXCEPTION 'La corsa indicata non esiste';
9          END IF;
10
11         /*se esiste gia un annullamento per una corsa allora
       lancia un errore per notificarlo*/
12
13         IF (EXISTS(SELECT * FROM annullamento WHERE corsa=
       id_corsa)) THEN
14             RAISE EXCEPTION 'La corsa e gia stata annullata';
15         ELSE
16             INSERT INTO annullamento VALUES (id_corsa,
       id_prossimo, rimborso_in, descrizione_in);
17         END IF;
18     END;
19
20 $$ LANGUAGE plpgsql;
```

4.4.7 add_ritardo(attributi di ritardo)

Questa procedura serve per aggiungere un ritardo ad una corsa:

```
1  CREATE or REPLACE PROCEDURE add_ritardo(motivazione_in
       VARCHAR(1000),ritardo_in TIME,id_corsa INTEGER) AS $$
2
3      DECLARE
4
5      BEGIN
```

```
6         /*se la corsa indicata non esiste lancia un errore
    per notificarlo*/
7         IF (NOT EXISTS(SELECT * FROM corsa AS c WHERE c.
    id_corsa=id_corsa)) THEN
8             RAISE EXCEPTION 'La corsa indicata non esiste';
9         END IF;
10
11        /*se esiste gia un ritardo per la corsa allora
    aggiorna i dati del ritardo sommando i tempi di tutti i
    ritardi e concatenando le motivazioni fra loro*/
12        IF (EXISTS(SELECT * FROM ritardo WHERE corsa=id_corsa
    )) THEN
13            UPDATE ritardo SET motivazione= motivazione || '
    --' ||  motivazione_in, ritardo= ritardo + ritardo_in
    WHERE corsa=id_corsa;
14        /*altrimenti procedi a creare il ritardo*/
15        ELSE
16            INSERT INTO ritardo VALUES (id_corsa, ritardo_in,
     motivazione_in);
17        END IF;
18    END;
19
20 $$ LANGUAGE plpgsql;
```

4.4.8 add_biglietto_intero(attributi di biglietto intero)

Questa procedura serve per aggiungere un biglietto_intero quando un passeggero di maggiore eta acquista una biglietto per una determinata corsa:

```
1 CREATE OR REPLACE PROCEDURE add_biglietto_intero(
2
3   Importo_totale float NOT NULL,
4   Sovrapprezzo_tot float default 0,
5   N_bagagli int default 0,
6   Veicolo varchar(10),
7   Prenotazione timestamp,
8   Corsa int NOT NULL,
9   Passeggero int NOT NULL,
10
11 ) AS $$
12
13    DECLARE
14
15    BEGIN
```

```
16          INSERT INTO biglietto_intero (Importo_totale,
    Sovrapprezzo_tot,N_bagagli,Veicolo,Prenotazione,Corsa,
    Passeggero)
17          VALUES (Importo_totale,Sovrapprezzo_tot,N_bagagli,
    Veicolo,Prenotazione,Corsa,Passeggero);
18      END;
19
20 $$LANGUAGE plpgsql;
```

4.4.9 add_biglietto_ridotto(attributi di biglietto ridotto)

Questa procedura serve per aggiungere un biglietto_intero quando un passeggero di minore eta acquista una biglietto per una determinata corsa:

```
1  CREATE OR REPLACE PROCEDURE add_biglietto_ridotto(
2
3    Id_biglietto int,
4    Importo_totale float NOT NULL,
5    Sovrapprezzo_tot float default 0,
6    N_bagagli int default 0,
7    Veicolo varchar(10),
8    Prenotazione timestamp,
9    Corsa int NOT NULL,
10   Passeggero int NOT NULL,
11   Accompagnatore int NOT NULL,
12
13 ) AS $$
14
15      DECLARE
16
17      BEGIN
18          INSERT INTO biglietto_intero (Importo_totale,
     Sovrapprezzo_tot,N_bagagli,Veicolo,Prenotazione,Corsa,
     Passeggero,Accompagnatore)
19          VALUES (Importo_totale,Sovrapprezzo_tot,N_bagagli,
     Veicolo,Prenotazione,Corsa,Passeggero,Accompagnatore);
20      END;
21
22 $$LANGUAGE plpgsql;
```

4.5 Trigger individuati

Sono stati individuati i seguenti trigger di utility:

4.5.1 trigger_scali

Il trigger per gli scali controlla ad ogni inserimento di una nuova corsa (con porto_scalo non null) o ad ogni aggiornamento di porto_scalo (sempre quando non null) se le corse intermedie fra lo scalo esistono gia. in caso contrario le aggiunge con degli insert:

```
1  CREATE FUNCTION add_scalo() RETURNS trigger AS $add_scalo$
2      BEGIN
3
4          boolean condizione = porto_partenza=new.porto_scalo
       AND porto_arrivo=new.porto_arrivo AND
       giorni_servizio_attivo=new.giorni_servizio_attivo
5                              AND Orario_partenza=new.
       Orario_partenza_scalo AND Orario_arrivo=new.Orario_arrivo
       AND Data_inizio_servizio=new.Data_inizio_servizio
6                              AND Data_fine_servizio=new.
       Data_fine_servizio AND Compagnia=new.Compagnia AND Natante
       =new.Natante;
7
8          boolean condizione2 = porto_partenza=new.
       porto_partenza AND porto_arrivo=new.porto_scalo AND
       giorni_servizio_attivo=new.giorni_servizio_attivo
9                               AND Orario_partenza=new.
       Orario_partenza AND Orario_arrivo=new.Orario_arrivo_scalo
       AND Data_inizio_servizio=new.Data_inizio_servizio
10                              AND Data_fine_servizio=new.
       Data_fine_servizio AND Compagnia=new.Compagnia AND Natante
       =new.Natante
11
12         /*partiamo dal presupposto che cio che il trigger fa
       e' scomporre una corsa A->B(scalo)->C aggiungendo alla
       base di dati le due corse A->B E B->C*/
13
14         /*il primo not exists controlla tramite condizione 1
       che non esista gia la corsa B->C , nel caso in cui non
       esiste la aggiunge*/
15
16         IF(NOT EXISTS(SELECT * FROM corsa WHERE condizione))
       THEN
17             INSERT INTO corsa (Orario_partenza,Orario_arrivo,
       Orario_partenza_scalo,Orario_arrivo_scalo,
       Data_inizio_servizio,Data_fine_servizio,
18                                 Giorni_Servizio_Attivo,
       Sovr_prenotazione,Sovr_bagaglio,Sovr_veicolo,Prezzo_intero
```

```
    ,Prezzo_ridotto,Sconto_residente,
19                                  Porto_partenza,Porto_arrivo,
    porto_scalo,Compagnia,Natante)
20
21                  VALUES(new.Orario_partenza_scalo,new.
    Orario_arrivo,null,null,new.Data_inizio_servizio,new.
    Data_fine_servizio,
22                  new.Giorni_Servizio_Attivo,new.
    Sovr_prenotazione,new.Sovr_bagaglio,new.Sovr_veicolo,new.
    Prezzo_intero,new.Prezzo_ridotto,new.Sconto_residente,
23                  new.Porto_scalo,new.Porto_arrivo,null,new.
    Compagnia,new.Natante);
24          END IF;
25
26          /*il secondo not exists controlla tramite condizione
    2 che non esista gia la corsa A->B , nel caso in cui non
    esiste la aggiunge*/
27
28          IF(NOT EXISTS(SELECT * FROM corsa WHERE condizione2))
     THEN
29              INSERT INTO corsa (Orario_partenza,Orario_arrivo,
    Orario_partenza_scalo,Orario_arrivo_scalo,
    Data_inizio_servizio,Data_fine_servizio,
30                                  Giorni_Servizio_Attivo,
    Sovr_prenotazione,Sovr_bagaglio,Sovr_veicolo,Prezzo_intero
    ,Prezzo_ridotto,Sconto_residente,
31                                  Porto_partenza,Porto_arrivo,
    porto_scalo,Compagnia,Natante)
32
33              VALUES(new.Orario_partenza,new.
    Orario_arrivo_scalo,null,null,new.Data_inizio_servizio,new
    .Data_fine_servizio,
34                  new.Giorni_Servizio_Attivo,new.
    Sovr_prenotazione,new.Sovr_bagaglio,new.Sovr_veicolo,new.
    Prezzo_intero,new.Prezzo_ridotto,new.Sconto_residente,
35                  new.Porto_partenza,new.Porto_scalo,null,new.
    Compagnia,new.Natante);
36          END IF;
37
38          RETURN NULL;
39      END;
40 $add_scalo$ LANGUAGE plpgsql;
41
42 CREATE TRIGGER add_scalo
43 AFTER INSERT OR UPDATE OF porto_scalo ON corsa
```

```
44 FOR EACH ROW
45 WHEN (new.porto_scalo IS NOT NULL)
46 EXECUTE FUNCTION add_scalo();
```

4.5.2 trigger_prossimo

Il trigger prossimo si attiva quando si prova ad inserire un annullamento; il suo scopo è trovare un corsa di sostituzione per la giornata(se esiste) da inserire nell attributo prossimo di annullamento.

```
1
2 CREATE or replace FUNCTION add_prossimo() RETURNS trigger AS
    $add_prossimo$
3
4     DECLARE
5
6         ann_partenza corsa.porto_partenza%TYPE;
7         ann_arrivo corsa.porto_arrivo%TYPE;
8     ann_orario_partenza corsa.orario_partenza%TYPE;
9         day_of_week INTEGER;
10
11
12    BEGIN
13
14        /*seleziona il giorno della settimana corrente,
    abbiamo bisogno del modulo perche extract restituisce i
    giorni da 0 a 6 partendo da domenica*/
15
16        day_of_week = mod(extract( dow from CURRENT_DATE)
    +6,7) +1;
17
18
19
20        /*seleziono la corsa annullata*/
21
22        SELECT porto_partenza,porto_arrivo,orario_partenza
    INTO ann_partenza,ann_arrivo,ann_orario_partenza FROM
    corsa WHERE id_corsa=new.corsa;
23
24        /*se esistono corse che partono dallo stesso punto ,
    arrivano nello stesso punto, disponibili nel giorno
    corrente , nella data corrente ad un orario piu tardo
    allora seleziona quella tra queste corse che parte prima*/
25
```

```
26          IF(EXISTS(SELECT * FROM corsa WHERE SUBSTRING(
    Giorni_Servizio_Attivo,day_of_week+1,1) = '1' AND
    porto_partenza=ann_partenza AND porto_arrivo=ann_arrivo
    AND orario_partenza>ann_orario_partenza AND(CURRENT_DATE
    BETWEEN Data_inizio_servizio AND Data_fine_servizio)) )
    THEN
27
28              new.prossimo:= (SELECT id_corsa
29                              FROM corsa
30                              WHERE SUBSTRING(
    Giorni_Servizio_Attivo,day_of_week+1,1) = '1' AND
    porto_partenza=ann_partenza
31                                      AND porto_arrivo=
    ann_arrivo AND orario_partenza>ann_orario_partenza
32                                      AND(CURRENT_DATE BETWEEN
    Data_inizio_servizio AND Data_fine_servizio)
33                              ORDER BY Orario_partenza ASC
    LIMIT 1);
34          END IF;
35
36
37          RETURN new;
38      END;
39 $add_prossimo$ LANGUAGE plpgsql;
40
41 CREATE or replace TRIGGER add_prossimo
42 BEFORE INSERT ON annullamento
43 FOR EACH ROW
44 WHEN (new.prossimo IS NULL)
45 EXECUTE FUNCTION add_prossimo();
```

4.6 Operazioni Batched

La nostra base di dati per funzionare correttamente inoltre deve avere le seguenti operazioni batched:

1. **Operazione di delete ritardi:** questa operazione deve eliminare all inizio di ogni nuovo giorno, i ritardi di una corsa.

2. **Operazione di delete annullamenti:** questa operazione deve eliminare all inizio di ogni nuovo giorno, gli annullamenti di una corsa.

3. **Operazione di delete corsa (data_fine_servizio):** questa operazione deve eliminare all inizio di ogni nuovo giorno, le corse per cui è arrivata la data di fine servizio.

Le seguenti operazioni sono state implementate tramite l ausilio dell estensione di postgresql (pgAgent Jobs), le funzioni di delete sono state programmate per essere eseguite tutti i giorni dell anno all orario 00:00 (mezzanotte):

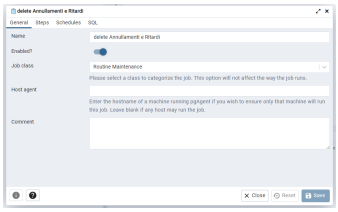


Figura 1: Creazione Job

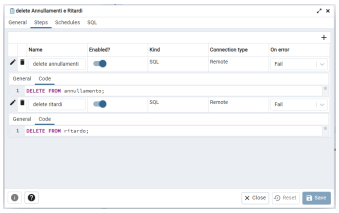


Figura 2: Inserimento steps

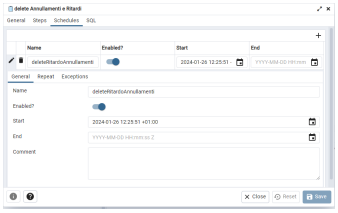


Figura 3: Inserimento scheduler

Codice dell inserimento delle operazioni:

```
1
2 DO $$
3 DECLARE
4     jid integer;
5     scid integer;
6 BEGIN
7 -- Creiamo un nuovo job
8 INSERT INTO pgagent.pga_job(
9     jobjclid, jobname, jobdesc, jobhostagent, jobenabled
10 ) VALUES (
11     1::integer, 'delete Annullamenti e Ritardi'::text, ''::
    text, ''::text, true
12 ) RETURNING jobid INTO jid;
13
14 -- Steps
15 -- Inseriamo uno step (delete annullamenti)(jobid: NULL)
16 INSERT INTO pgagent.pga_jobstep (
17     jstjobid, jstname, jstenabled, jstkind,
18     jstconnstr, jstdbname, jstonerror,
19     jstcode, jstdesc
20 ) VALUES (
21     jid, 'delete annullamenti'::text, true, 's'::character(1)
    ,
```

```
22      'host=''ep-holy-pine-64797515.eu-central-1.aws.neon.tech'
        ' dbname=''Arturo%27s%20Sea%20Travels'' user=''riccpuggio'
        ' password=''owYA6Z4KklUN''''::text, ''::name, 'f'::
        character(1),
23      'DELETE FROM annullamento;'::text, ''::text
24  );
25  -- Inseriamo uno step (delete ritardi)(jobid: NULL)
26  INSERT INTO pgagent.pga_jobstep (
27      jstjobid, jstname, jstenabled, jstkind,
28      jstconnstr, jstdbname, jstonerror,
29      jstcode, jstdesc
30  ) VALUES (
31      jid, 'delete ritardi'::text, true, 's'::character(1),
32      'host=''ep-holy-pine-64797515.eu-central-1.aws.neon.tech'
        ' dbname=''Arturo%27s%20Sea%20Travels'' user=''riccpuggio'
        ' password=''owYA6Z4KklUN''''::text, ''::name, 'f'::
        character(1),
33      'DELETE FROM ritardo;'::text, ''::text
34  );
35
36  -- Schedules
37  -- Inseriamo uno schedule per l esecuzione automatica
38  INSERT INTO pgagent.pga_schedule(
39      jscjobid, jscname, jscdesc, jscenabled,
40      jscstart,     jscminutes, jschours, jscweekdays,
        jscmonthdays, jscmonths
41  ) VALUES (
42      jid, 'deleteRitardoAnnullamenti'::text, ''::text, true,
43      '2024-01-26 12:25:51 +01:00'::timestamp with time zone,
44      -- Minutes
45      '{t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f
        ,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f
        ,f,f,f}'::bool[]::boolean[],
46      -- Hours
47      '{t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f}'::bool
        []::boolean[],
48      -- Week days
49      '{f,f,f,f,f,f,f}'::bool[]::boolean[],
50      -- Month days
51      '{f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f
        ,f,f,f,f}'::bool[]::boolean[],
52      -- Months
53      '{f,f,f,f,f,f,f,f,f,f,f,f}'::bool[]::boolean[]
54  ) RETURNING jscid INTO scid;
55  END
```

```
$$;
```

Mentre invece questo è l'inserimento dell operazione di delete corsa:

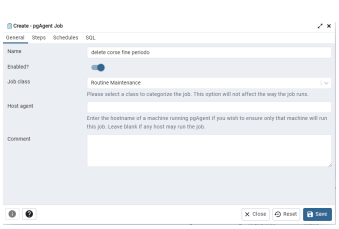


Figura 4: Creazione Job

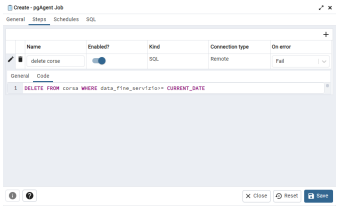


Figura 5: Inserimento steps

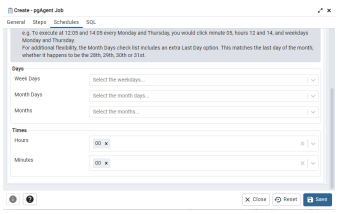


Figura 6: Inserimento scheduler

Di seguito il codice:

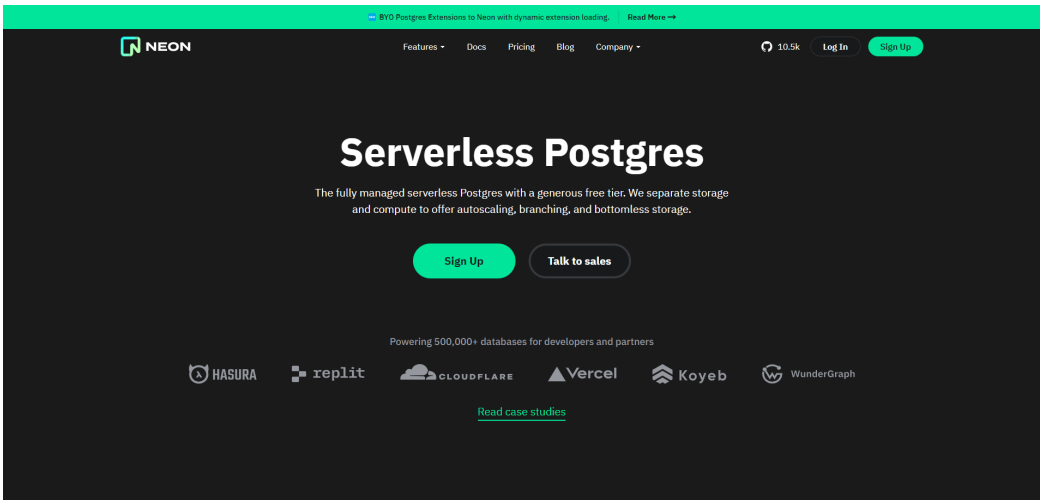
```
1
2  DO $$
3  DECLARE
4      jid integer;
5      scid integer;
6  BEGIN
7  -- crea un nuovo job
8  INSERT INTO pgagent.pga_job(
9      jobjclid, jobname, jobdesc, jobhostagent, jobenabled
10 ) VALUES (
11     1::integer, 'delete corse fine periodo'::text, ''::text,
       ''::text, true
12 ) RETURNING jobid INTO jid;
13
14 -- Steps
15 -- inseriamo gli steps
16 INSERT INTO pgagent.pga_jobstep (
17     jstjobid, jstname, jstenabled, jstkind,
18     jstconnstr, jstdbname, jstonerror,
19     jstcode, jstdesc
20 ) VALUES (
21     jid, 'delete corse'::text, true, 's'::character(1),
22     'host=''ep-holy-pine-64797515.eu-central-1.aws.neon.tech'
       ' dbname=''Arturo%27s%20Sea%20Travels'' user=''riccpuggio'
       ' password=''owYA6Z4KklUN'''::text, ''::name, 'f'::
       character(1),
23     'DELETE FROM corsa WHERE data_fine_servizio >=
       CURRENT_DATE'::text, ''::text
24 ) ;
```

```
25
26 -- Schedules
27 -- inseriamo uno schedule
28 INSERT INTO pgagent.pga_schedule(
29     jscjobid, jscname, jscdesc, jscenabled,
30     jscstart,     jscminutes, jschours, jscweekdays,
    jscmonthdays, jscmonths
31 ) VALUES (
32     jid, 'scheduler cancella data_fine_servizio'::text, ''::
    text, true,
33     '2024-02-04 19:17:00 +01:00'::timestamp with time zone,
34     -- Minutes
35     '{t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f
    ,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f
    ,f,f,f}'::bool[]::boolean[],
36     -- Hours
37     '{t,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f}'::bool
    []::boolean[],
38     -- Week days
39     '{f,f,f,f,f,f,f}'::bool[]::boolean[],
40     -- Month days
41     '{f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f,f
    ,f,f,f,f}'::bool[]::boolean[],
42     -- Months
43     '{f,f,f,f,f,f,f,f,f,f,f,f}'::bool[]::boolean[]
44 ) RETURNING jscid INTO scid;
45 END
46 $$;
```

5 Deploy della base di dati e note finali

5.1 Deploy

Per il Deploy della base di dati ci si è affidati ad una soluzione popolare e che prevede un piano gratuito molto generoso: neon db.



5.2 Note finali

Durante la creazione di una base di dati per una piattaforma di ricerca traghetti si è quindi andati a sviscerare i requisiti per la rappresentazione del nostro mini-world; abbiamo poi usato questi requisiti per costruire uno schema concettuale della base di dati,successivamente si è andati a operare una ristrutturazione di esso in modo da procedere in seguito con una traduzione al modello logico; infine abbiamo implementato il database fisicamente seguendo il modello logico ottenuto.