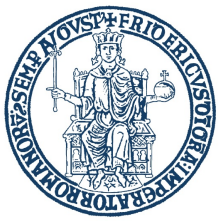


UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Corso di Laurea Triennale in Informatica

Documentazione del DataBase per Compagnia di Navigazione

Raucci Giovanni N86004635
Regina Riccardo N86004614
Rossetti Francesco N86004505

DICEMBRE 2023

Indice

1 Progettazione Concettuale

1.1 Analisi dei Requisiti

Durante la fase di analisi dei requisiti, l'obiettivo principale è identificare le informazioni chiave necessarie per la realizzazione della struttura del database della compagnia di navigazione. In particolare, si concentrerà sull'individuazione delle entità, delle associazioni tra di esse, e sui vincoli e comportamenti del database.

> *"Il sistema si basa sulla conoscenza delle corse offerte dalle compagnie di navigazione. 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 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. Ad esempio, una compagnia potrebbe operare una corsa di motonave da Mu ad Atlantide soltanto il martedì e il giovedì e nel periodo tra il 15 giugno e il 15 settembre."*

In base alle specifiche fornite, è possibile identificare diverse entità fondamentali per la struttura del database. Inizialmente, l'entità *CORSAREGOLARE* conterrà le informazioni fondamentali relative alle corse ricorrenti in determinati periodi dell'anno, compresi gli orari di partenza e di arrivo. Inoltre, *CORSASPECIFICA* rappresenterà un'istanza specifica di una corsa regolare e includerà quindi come attributo la data in cui si svolgerà la corsa.

Un'altra entità individuata è il *NATANTE* dotato di nome e suddiviso in tre tipi distinti: *TRAGHETTO*, con posti dedicati sia a passeggeri che a veicoli; *ALISCAFO* e *MOTONAVE*, entrambi progettati solo per passeggeri. Poiché le richieste specificano il vincolo di effettuare ogni corsa solo in determinati periodi dell'anno, l'entità *CORSAREGOLARE* sarà associata ad un'entità *PERIODO* che conterrà una data di inizio e una data di fine periodo in cui la corsa può essere effettuata, inoltre avrà anche un attributo multivalore per indicare i giorni della settimana in cui la corsa è disponibile. Questa scelta di implementazione è motivata dal fatto che ogni corsa può coprire un numero non precisato di periodi.

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

Per quanto riguarda la gestione dei prezzi, l'entità *CORSAREGOLARE* comprende gli attributi relativi al costo del biglietto intero, eventuali sconti per prezzi ridotti e i sovrapprezzi legati alla prenotazione, ai bagagli o al veicolo che il *CLIENTE* desidera imbarcare.

Dall'altra parte, nell'entità *BIGLIETTO* sono inclusi gli attributi associati alle scelte effettuate dal cliente durante l'acquisto del biglietto, come l'età del passeggero, la prenotazione e l'imbarco di bagagli o veicoli.

L'entità *PORTO* è caratterizzata da attributi come il comune, l'indirizzo e il numero di telefono del porto. Tra l'entità *PORTO* e *CORSAREGOLARE*, esistono diverse relazioni che andranno a mappare per ogni corsa il porto di partenza, di arrivo e quello di un possibile scalo intermedio.

Per quanto riguarda l'entità *COMPAGNIA*, gli attributi comprendono il nome e vari contatti come numeri telefonici, indirizzi email e sito web. Inoltre, considerando la diversità dei social media, i quali potrebbero avere tag distinti, verranno strutturati come un'entità separata *SOCIAL*.

"Il sistema può essere utilizzato dalle compagnie e dai passeggeri. Le compagnie possono aggiornare le proprie corse oppure segnalare l'annullamento o il ritardo di una singola corsa. Il passeggero può consultare il tabellone delle corse, che contiene tutte le corse da un determinato porto di partenza verso un determinato porto di destinazione, da un giorno e orario di partenza indicato e per le successive 24 ore, eventualmente filtrate in base al tipo di natante scelto o in base al prezzo. Nel tabellone le corse in ritardo o cancellate dovranno comunque essere mostrato con una annotazione che riporti questo evento."

Poiché il sistema prevede l'utilizzo da parte di due categorie distinte di utenti, verrà introdotta una generalizzazione delle classi *COMPAGNIA* e *CLIENTE* nella classe *UTENTE*. Quest'ultima sarà dotata di attributi di login e password per consentire l'autenticazione.

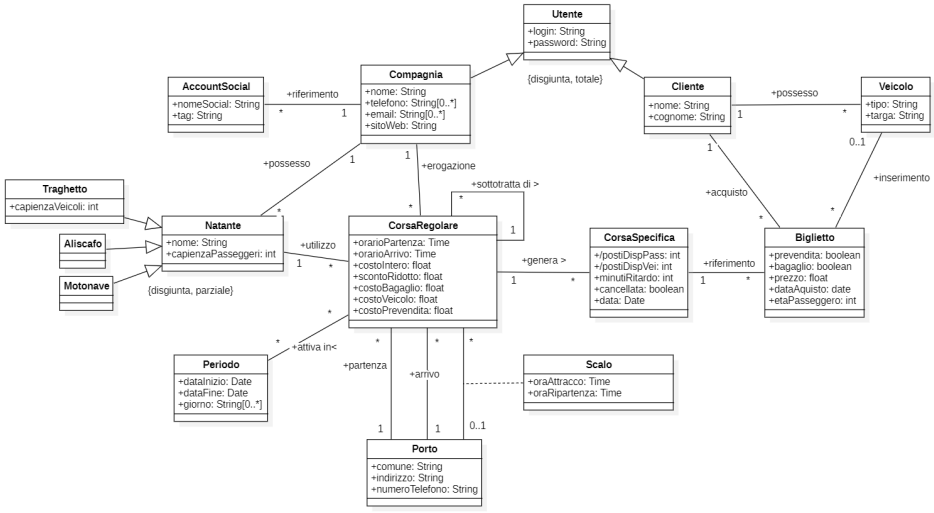
Dal momento che le compagnie hanno la facoltà di aggiornare, cancellare o segnalare ritardi per singole corse, saranno aggiunti due attributi all'entità

CORSASPECIFICA: uno per i minuti di ritardo e l'altro per indicare se la corsa è stata cancellata o meno.

> *"Le compagnie indicano anche la capienza dei natanti utilizzati per quella corsa, in termini di numero di posti per i passeggeri e numero di posti per autoveicoli. I passeggeri possono prenotare una singola corsa solo se il sistema verificherà la presenza di un posto utile. All'atto della prenotazione il sistema dovrà aggiornare il numero dei posti utili in tutte le tratte interessate. Ad esempio, all'atto della prenotazione di due biglietti passeggero e un autoveicolo da Atlantide a Tortuga dovranno essere diminuite le disponibilità di biglietti passeggero ed autoveicolo anche nella tratta da Atlantide a Mu (ma non in quella da Tortuga a Mu)."*

Siccome è richiesto identificare quanti posti disponibili ci sono, saranno inseriti come attributi nell'entità *CORSASPECIFICA*.

1.2 Schema Concettuale UML



1.3 Dizionario delle Classi

Classe	Descrizione	Attributi
Utente	L'utente fa l'accesso al sistema o come Compagnia per aggiungere o aggiornare le corsa, o come Cliente per consultare le corse disponibili	**login**(String): login identificativo per accedere all'area riservata della compagnia o del cliente **password**(String): password per accedere all'area riservata della compagnia o del cliente
Compagnia	La compagnia di navigazione offre delle corse in traghetto, aliscafo, motonave o altro tra le varie isole	**nome**(String): nome della compagnia **numero**(String): numero di telefono della compagnia **indirizzo**(String): indirizzo di posta elettronica della compagnia **sitoWeb**(String): link URL al sito web della compagnia

Classe	Descrizione	Attributi
Porto	Luogo in cui le imbarcazioni salpano e attraccano	**comune**(String): nome del comune di appartenenza del porto **indirizzo**(String): indirizzo presso il quale è situato il porto **numeroTelefono**(String): numero telefonico del servizio informazioni
Scalo	Atrracco al porto di un isola intermedia per consentire ad altri passeggeri di arrivare a destinazione	**OraAttracco**(Timestamp): ora in cui il natante arriva al porto **OraRipartenza**(Timestamp): ora in cui il natante riparte **costoPrimaTratta**(float): costo della tratta intermedia che va dal porto di partenza al porto di scalo **costoSecondaTratta**(float): costo della tratta intermedia che va dal porto di scalo al porto di arrivo
CorsaRegolare	Tratta marittima messa a disposizione da una compagnia che collega due isole (o tre se c'è uno scalo) che si ripeterà per determinati periodi e determinati giorni	**OrarioPartenza**(Time): ora in cui il natante salpa dal porto di partenza e ha inizio la corsa **OrarioArrivo**(Time): ora in cui il natante attracca al porto di arrivo e la corsa ha termine **costoIntero**(float): costo base della corsa per un adulto senza bagaglio o veicolo **scontoRidotto**(float): percentuale di sconto per un biglietto ridotto **costoBagaglio**(float): sovrapprezzo per i clienti che portano un bagaglio **costoVeicolo**(float): sovrapprezzo per i clienti che imbarcano un veicolo **costoPrevendita**(float): sovrapprezzo per i clienti che acquistano il biglietto in prevendita

Classe	Descrizione	Attributi
CorsaSpecifica	Istanza specifica di una corsa regolare	**data**(Date): giorno in cui viene effettuata la corsa **postiDispPass**(int): numero di posti passeggeri ancora disponibili **postiDispVei**(int): numero di posti veicoli ancora disponibili **minutiRitardo**(int): minuti di ritardo della corsa **cancellata**(boolean): valore booleano che indica se una corsa è stata cancellata dalla compagnia
Periodo	Periodo dell'anno in cui è attiva una corsa regolare	**dataInizio**(Date): data di inizio del periodo **dataFine**(Date): data di fine del periodo **giorno**(String): attributo multivalore che indica i giorni in cui la corsa è attiva
Natante	Categoria di imbarcazione utilizzata dalle compagnie di navigazione	**nome**(String): nome identificativo del natante **capienzaPasseggeri**(int): numero massimo di passeggeri che il natante può trasportare
Traghetto	Specializzazione di Natante dotato di posti per passeggeri e per veicoli	**capienzaVeicoli**(int): numero massimo di veicoli che il natante può trasportare
Aliscafo	Specializzazione di Natante dotato di posti solo per passeggeri	
Motonave	Specializzazine di Natante dotati di posti solo per passeggeri	
Cliente	Utente registrato al sistema che decide di acquistare uno o più biglietti per una o più corse specifiche	**nome**(String): nome del cliente **cognome**(String): cognome del cliente

Classe	Descrizione	Attributi
Biglietto	Biglietto acquistato dal cliente per usufruire di una determinata corsa	**eta**(int): età del passeggero **prevendita**(boolean): valore booleano che indica se il cliente ha prenotato o meno il biglietto **bagaglio**(boolean): valore booleano che indica se il cliente porta con se o meno un bagaglio **prezzo**(int): prezzo totale del biglietto calcolato in base all'età del passeggero e alla presenza del bagaglio e dell'autoveicolo **dataAcquisto**(Date): data di acquisto del biglietto
Veicolo	Informazioni sul veicolo che un cliente vuole imbarcare	**targa**(String): targa identificativa del veicolo **tipo**(String): tipo di veicolo (Automobile, Scooter etc.)
Account Social	Informazioni sui vari account social di una compagnia	**nomeSocial**(String): nome del social al qual è associato l'account **tag**(String): tag dell'account

1.4 Dizionario delle Associazioni

Associazione	Descrizione
Erogazione	Associazione **uno-a-molti** tra *Compagnia* e *CorsaRegolare*. Una Compagnia può erogare zero o più corse regolari e una CorsaRegolare se esiste è erogata da una ed una sola Compagnia.
Partenza	Associazione **uno-a-molti** tra *Porto* e *CorsaRegolare*. Una Corsa parte da uno ed un solo Porto, mentre da un Porto possono partire zero o più Corse.
Arrivo	Associazione **uno-a-molti** tra *Porto* e *CorsaRegolare*. Una Corsa ha come destinazione finale uno ed un solo Porto, mentre un Porto può essere destinazione di zero o più Corse.

Associazione	Descrizione
Scalo	Associazione **uno-a-molti** tra *Porto* e *CorsaRegolare*. Una Corsa può fare scalo al più in un solo Porto intermedio e in un Porto possono fare scalo zero o più Corse. Inoltre ogni Scalo è caratterizzato dall'orario in cui il natante che fa lo scalo arriva al porto (*oraAttracco*) e dall'orario in cui riparte (*oraRipartenza*).
SottotrattaDi	Associazione ricorsiva **uno-a-molti** di *CorsaRegolare*. Una corsa regolare A puó essere un segmento di una ed una sola corsa regolare B, la cui tratta di competenza contiene la tratta di competenza di A. Viceversa, B puó contenere piú sottocorse del tipo di A.
AttivaIn	Associazione **molti-a-molti** tra *CorsaRegolare* e *Periodo*. Una corsa può essere attiva in più periodi e un periodo può coprire più corse.
Utilizzo	Associazione **uno-a-molti** tra *Natante* e *CorsaRegolare*. Un Natante può essere utilizzato per compiere zero o più Corse, mentre una Corsa può usare uno ed un solo Natante.
Possesso	Associazione **uno-a-molti** tra *Compagnia* e *Natante*. Una Compagnia può possedere zero o più Natanti, mentre un Natante è intestato ad una ed una sola Compagnia.
Genera	Associazione **uno-a-molti** tra *CorsaRegolare* e *CorsaSpecifica*. Ogni CorsaRegolare genera una CorsaSpecifica per ogni giorno dei periodi in cui è disponibile, mentre una CorsaSpecifica è generata da una ed una sola CorsaRegolare.
Riferimento	Associazione **uno-a-uno** tra *CorsaSpecifica* e *Biglietto*. Per una CorsaSpecifica possono essere stati venduti zero o più Biglietti, mentre un Biglietto fa riferimento ad una ed una sola CorsaSpecifica.
Acquisto	Associazione **uno-a-molti** tra *Cliente* e *Biglietto*. Un Cliente può comprare zero o più Biglietti, mentre un Biglietto è intestato ad uno ed un solo Cliente.
Possesso	Associazione **uno-a-molti** tra *Cliente* e *Veicolo*. Un Cliente può possedere zero o più Veicoli, mentre un Veicolo è intestato ad uno ed un solo Cliente.
Inserimento	Associazione **uno-a-molti** tra *Veicolo* e *Biglietto*. Un Biglietto può essere associato al più ad un solo Veicolo, mentre un Veicolo può essere imbarcato più volte quindi può essere associato a zero o a più Biglietti.

Associazione	**Descrizione**
Riferimento	Associazione **uno-a-molti** tra *Compagnia* e *AccountSocial*. Una Compagnia può avere zero o più Account Social, mentre ogni Account Social è riferito ad una ed una sola Compagnia.

2 Ristrutturazione del modello concettuale

2.1 Analisi delle ridondanze

All'interno del diagramma iniziale, si identifica una ridondanza legata al concetto di costo, presente sia nell'entità *BIGLIETTO* con l'attributo *prezzo* che nell'entità *CORSAREGOLARE* con gli attributi *costoIntero*, *costoRidotto*, *costoBagaglio*, *costoVeicolo* e *costoPrevendita*. La scelta di mantenere la ridondanza è motivata dal fatto che il costo, sia esso intero o ridotto, rimarrà fisso per ogni corsa e non subirà variazioni, mentre il prezzo del biglietto sarà determinato in base alle opzioni selezionate dal cliente al momento dell'acquisto, ad esempio l'aggiunta di un veicolo o di un bagaglio.

Un'altra ridondanza riscontrata è relativa al numero di posti disponibili per una corsa specifica. Questo valore potrebbe essere calcolato sottraendo al numero massimo di posti del natante utilizzato per quella corsa il numero di biglietti venduti, informazione ottenibile contando il numero di tuple relative a quella corsa nella tabella *BIGLIETTO*. Tuttavia, è stata mantenuta un'esplicita registrazione del numero di posti disponibili come attributo separato nell'entità *CORSASPECIFICA* per semplificare e ottimizzare le operazioni di lettura e rendere più efficienti le query relative alla disponibilità dei posti.

2.2 Eliminazione degli attributi multivalore

Nel diagramma iniziale, sono presenti alcuni attributi multivalore, tra cui due riferiti all'entità *COMPAGNIA*, ossia *TELEFONO* e *EMAIL*. Per gestire in modo più efficiente e flessibile questi contatti di assistenza, si è optato per considerarli come entità separate, consentendo così la gestione di più contatti.

Un altro attributo multivalore è *giorno* all'interno dell'entità *PERIODO*, che indica i giorni della settimana in cui la corsa è disponibile. Al fine di semplificare le operazioni di controllo necessarie per implementare alcune richieste, si è scelto di mantenere *giorno* come una stringa unica di sette caratteri, composti esclusivamente da 0 e 1. Questa rappresentazione permette di indicare in modo chiaro e compatto la disponibilità della corsa nei vari giorni della settimana. Ad esempio, se la stringa *giorni* è "1001101", significa che la corsa sarà disponibile nei giorni domenica, mercoledì, giovedì e sabato (il bit in posizione 0 si riferisce alla domenica).

2.3 Eliminazione degli attributi composti

Non sono presenti attributi composti.

2.4 Eliminazione delle generalizzazioni

Nel diagramma iniziale progettato per la costruzione del database, sono state introdotte alcune generalizzazioni. Una di esse coinvolge l'entità *UTENTE*, la quale si specializza in *COMPAGNIA* o *CLIENTE*. Questa specializzazione è di tipo totale disgiunta. Ogni utente del sistema, sia esso un cliente o una compagnia, è caratterizzato da un *login* e da una *password*. Pertanto, abbiamo scelto di semplificare il modello eliminando l'entità *UTENTE* e includendo gli attributi di *login* e *password* sia nell'entità *COMPAGNIA* che in quella *CLIENTE*.

La seconda generalizzazione inserita è, invece, una specializzazione disgiunta parziale e coinvolge l'entità *NATANTE*, che può specializzarsi in *TRAGHETTO*, *ALISCAFO*, *MOTONAVE* o anche nessuno dei tre. Poiché solo i traghetti hanno la possibilità di trasportare veicoli e gli altri due tipi di natante condividono gli stessi attributi, abbiamo scelto di raggruppare le classi figlie all'interno della classe padre, quindi è stato aggiunto un attributo *tipo* per specificare il tipo di natante e un attributo *capienzaVeicoli*, il quale sarà NULL nel caso di aliscafi e motonavi. Questa modifica semplifica la struttura del modello, evitando la duplicazione degli attributi comuni tra aliscafi e motonavi.

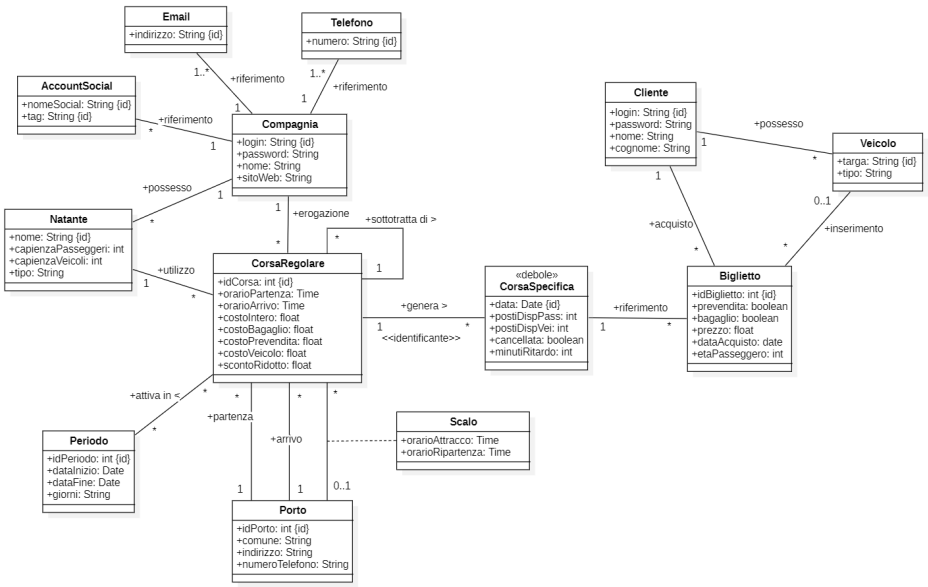
2.5 Identificazioni delle chiavi primarie

In questa fase, procederemo a selezionare uno più attributi per l'identificazione univoca delle diverse entità presenti nello schema precedente. In particolare:

- ***COMPAGNIA***: Ogni compagnia può essere identificata univocamente attraverso l'attributo **login**, utilizzato per l'accesso al sistema.
- ***CLIENTE***: Analogamente, anche per i clienti, l'identificazione univoca avviene tramite l'attributo **login**.
- ***CORSAREGOLARE***: Per l'entità *CORSAREGOLARE*, è stata introdotta una chiave surrogata, **idCorsa**, poiché le altre chiavi candidate erano composte da un insieme di più attributi, rendendo poco efficiente l'identificazione.
- ***CORSASPECIFICA***: L'entità *CORSASPECIFICA* è un'entità debole con chiave parziale **data**, poichè ogni corsa regolare ha cadenza giornaliera.

- ***PERIODO***: Anche per *PERIODO* è stata aggiunta una chiave surrogata, **idPeriodo**.
- ***PORTO***: Anche per *PORTO*, l'identificazione avviene attraverso una chiave surrogata, **idPorto**.
- ***SCALO***: L'identificazione di uno *SCALO* si basa sulla chiave esterna di CORSA, poiché ogni corsa può avere al più uno scalo.
- ***NATANTE***: L'identificazione di ogni *NATANTE* avviene tramite l'attributo **nome**.
- ***BIGLIETTO***: Per l'entità *BIGLIETTO*, è stata aggiunta la chiave surrogata **idBiglietto**, in quanto non è possibile identificarlo in altro modo.
- ***VEICOLO***: L'identificazione di ogni VEICOLO avviene attraverso l'attributo **targa**.
- ***ACCOUNTSOCIAL***: L'identificazione di *ACCOUNTSOCIAL* è basata sulla coppia di attributi **nomeSocial** e **tag** del profilo.
- ***EMAIL e TELEFONO***: In entrambi i casi, l'identificazione avviene attraverso un unico attributo, rispettivamente **indirizzo** per *EMAIL* e **numero** per *TELEFONO*, poiché ciascun valore deve essere unico all'interno del sistema.

2.6 Class Diagram UML ristrutturato



3 Traduzione al modello logico

3.1 Schema finale

Gli attributi <u>sottolineati</u> sono chiavi primarie, mentre gli attributi indicati con ↑ sono chiavi esterne.

Compagnia	(login, password, nome, sitoWeb)
CorsaRegolare	(idCorsa, PortoPartenza↑, PortoArrivo↑, orarioPartenza, orarioArrivo, costoIntero, scontoRidotto, costoBagaglio, costoPrevendita, costoVeicolo, Compagnia↑, Natante↑, CorsaSup↑) *PortoPartenza* → *Porto.idPorto* *PortoArrivo* → *Porto.idPorto* *Compagnia* → *Compagnia.login* *Natante* → *Natante.nome* *CorsaSup* → *CorsaRegolare.idCorsa*
CorsaSpecifica	(idCorsa↑, data, postiDispPass, postiDispVei, minutiRitardo, cancellata) *idCorsa* → *CorsaRegolare.idCorsa*
Periodo	(idPeriodo, datainizio, dataFine, giorni)
AttivaIn	(idCorsa↑, idPeriodo↑) *idCorsa* → *CorsaRegolare.idCorsa* *idPeriodo* → *Periodo.idPeriodo*
Porto	(idPorto, comune, indirizzo, numeroTelefono)
Scalo	(idCorsa↑, idPorto↑, orarioAttracco, orarioRipartenza) *idCorsa* → *CorsaRegolare.idCorsa* *idPorto* → *Porto.idPorto*
Natante	(nome, capienzaPasseggeri, capienzaVeicoli, tipo, Compagnia↑) *Compagnia* → *Compagnia.login*
Cliente	(login, password, nome, cognome)
Biglietto	(idBiglietto, idCorsa↑, data↑, Cliente↑, Veicolo↑, prevendita, bagaglio, prezzo, dataAcquisto, etaPasseggero) {*idCorsa,data*} → {*CorsaSpecifica.idCorsa*, *CorsaSpecifica.data*} *Cliente* → *Cliente.login* *Veicolo* → *Veicolo.targa*
Veicolo	(targa, tipo, Proprietario↑) *Proprietario* → *Cliente.login*
AccountSocial	(nomeSocial, tag, Compagnia↑) *Compagnia* → *Compagnia.login*
Email	(indirizzo, Compagnia↑) *Compagnia* → *Compagnia.login*
Telefono	(numero, Compagnia↑) *Compagnia* → *Compagnia.login*

4 Progettazione fisica

4.1 Creazione del database

```
--creazione di un database di nome Navigazione
create database Navigazione;
```

Creazione del database

4.2 Creazione dello schema

```
--creazione di uno schema di nome Navigazione
create schema Navigazione;
```

Creazione dello schema

4.3 Creazione dei tipi

Poiché un natante può essere o un traghetto, o una motonave, o un aliscafo o nessuno di questi tre è stato scelto di definire un nuovo tipo.

```
--creazione del tipo di natante come un'enumerazione dei
    ↪ 4 possibili tipi
create type tipoNatante as enum ('traghetto',
    ↪ 'motonave', 'aliscafo', 'altro');
```

Creazione del tipo tipoNatante

Lo stesso ragionamento è stato fatto per i tipi di veicolo che un cliente può scegliere di imbarcare.

```
--creazione del tipo di veicolo come un'enumerazione dei
    ↪ 4 possibili tipi
create type tipoVeicolo as enum ('automobile',
    ↪ 'motociclo', 'mezzo pesante', 'altro');
```

Creazione del tipo tipoVeicolo

4.4 Creazione delle tabelle

La maggior parte dei vincoli di chiave esterna verranno aggiunti in un secondo momento.

```
create table Navigazione.Compagnia(
    login text primary key,
    password text not null,
    nome text not null,
```

```
        sitoWeb text not null
    );
```

Creazione della tabella Compagnia

```
    create table Navigazione.CorsaRegolare(
        idCorsa serial primary key,
        PortoPartenza integer not null,
        PortoArrivo integer not null,
        orarioPartenza time not null,
        orarioArrivo time not null,
        costoIntero numeric not null check(costoIntero >= 0),
        scontoRidotto numeric not null check(scontoRidotto
        ↪ >= 0 AND scontoRidotto <=100),
        costoBagaglio numeric default 0 check(costoBagaglio
        ↪ >= 0),
        costoPrevendita numeric default 0
        ↪ check(costoPrevendita >= 0),
        costoVeicolo numeric default 0 check(costoVeicolo >=
        ↪ 0),
        Compagnia text not null,
        Natante text not null,
        CorsaSup integer not null,

        check (PortoArrivo <> PortoPartenza)
    );
```

Creazione della tabella CorsaRegolare

```
    create table Navigazione.CorsaSpecifica(
        idCorsa integer,
        data date,
        postiDispPass integer not null check(postiDispPass
        ↪ >= 0),
        postiDispVei integer check(postiDispVei >= 0 or
        ↪ postiDispVei is null),
        minutiRitardo integer not null default 0,
        cancellata boolean not null default 'false',

        primary key(idCorsa, data),
        foreign key(idCorsa) references
        ↪ Navigazione.CorsaRegolare(idCorsa)
            on delete cascade       on update cascade
    );
```

Creazione della tabella CorsaSpecifica

```
    create table Navigazione.Periodo(
        idPeriodo serial primary key,
        dataInizio date not null,
```

```
        dataFine date not null,
        giorni bit(7) not null,

        check (dataInizio < dataFine)
    );
```

Creazione della tabella Periodo

```
    create table Navigazione.AttivaIn (
        idCorsa integer,
        idPeriodo integer,

        primary key(idCorsa, idPeriodo),
        foreign key(idCorsa) references
        ↪ Navigazione.CorsaRegolare(idCorsa)
            on delete cascade       on update cascade,

        foreign key(idPeriodo) references
        ↪ Navigazione.Periodo(idPeriodo)
            on delete cascade       on update cascade
    );
```

Creazione della tabella AttivaIn

```
    create table Navigazione.Porto(
        idPorto serial primary key,
        comune text not null,
        indirizzo text not null,
        numeroTelefono text not null
    );
```

Creazione della tabella Porto

```
    create table Navigazione.Scalo(
        idCorsa integer primary key,
        Porto integer not null,
        orarioAttracco time not null,
        orarioRipartenza time not null,

        check(orarioAttracco < orarioRipartenza)
    );
```

Creazione della tabella Scalo

```
    create table Navigazione.Natante(
        nome text primary key,
        Compagnia text not null,
        capienzaPasseggeri integer not null,
        capienzaVeicoli integer,
        tipo tipoNatante not null default 'altro'
```

```
    );
```

Creazione della tabella Natante

```
    create table Navigazione.Cliente(
        login text primary key,
        password text not null,
        nome text not null,
        cognome text not null
    );
```

Creazione della tabella Cliente

```
    create table Navigazione.Biglietto(
        idBiglietto serial primary key,
        idCorsa integer not null,
        data Date not null,
        Cliente text not null,
        Veicolo text,
        prevendita boolean not null default 'false',
        bagaglio boolean not null default 'false',
        prezzo numeric not null check(prezzo >= 0),
        dataAcquisto date not null,
        etaPasseggero integer not null check (etaPasseggero
        ↪ >= 0),

        foreign key(idCorsa, data) references
        ↪ Navigazione.CorsaSpecifica(idCorsa, data)
            on delete cascade       on update cascade
    );
```

Creazione della tabella Biglietto

```
    create table Navigazione.Veicolo(
        targa text primary key,
        tipo tipoVeicolo not null default 'altro',
        Proprietario text not null
    );
```

Creazione della tabella Veicolo

```
    create table Navigazione.AccountSocial(
        nomeSocial text,
        tag text,
        Compagnia text not null,

        primary key(nomeSocial, tag)
    );
```

Creazione della tabella AccountSocial

```
create table Navigazione.Email(
    indirizzo text primary key,
    Compagnia text not null
);
```

Creazione della tabella Email

```
create table Navigazione.Telefono(
    numero text primary key,
    Compagnia text not null
);
```

Creazione della tabella Telefono

4.5 Creazione dei vincoli di chiave esterna

```
alter table Navigazione.CorsaRegolare
    add constraint corsaFKcompagnia
        foreign key (Compagnia) references
    ↪ Navigazione.Compagnia(login)
            on delete cascade       on update cascade;

alter table Navigazione.CorsaRegolare
    add constraint corsaFKnatante
        foreign key (Natante) references
    ↪ Navigazione.Natante(nome)
            on delete cascade   on update cascade;

alter table Navigazione.CorsaRegolare
    add constraint corsaFKportoPartenza
        foreign key (PortoPartenza) references
    ↪ Navigazione.Porto(idPorto)
            on delete cascade       on update cascade;

alter table Navigazione.CorsaRegolare
    add constraint corsaFKportoArrivo
        foreign key (PortoArrivo) references
    ↪ Navigazione.Porto(idPorto)
            on delete cascade       on update cascade;

alter table Navigazione.CorsaRegolare
    add constraint corsaFKcorsaSup
        foreign key (CorsaSup) references
    ↪ Navigazione.CorsaRegolare(idCorsa)
            on delete cascade       on update cascade;
```

Aggiunti vincoli di chiave esterna per la tabella CorsaRegolare

```
1    alter table Navigazione.Scalo
2        add constraint scaloFKcorsa
3            foreign key (idCorsa) references
         ↪ Navigazione.CorsaRegolare(idCorsa)
4                on delete cascade       on update cascade;
5
6    alter table Navigazione.Scalo
7        add constraint scaloFKporto
8            foreign key (Porto) references
         ↪ Navigazione.Porto(idPorto)
9                on delete cascade       on update cascade;
```

Aggiunti vincoli di chiave esterna per la tabella Scalo

```
1    alter table Navigazione.Natante
2        add constraint natanteFKcompagnia
3            foreign key (Compagnia) references
         ↪ Navigazione.Compagnia(login)
4                on delete cascade       on update cascade;
```

Aggiunto vincolo di chiave esterna per la tabella Natante

```
1    alter table Navigazione.Biglietto
2        add constraint bigliettoFKcliente
3            foreign key (Cliente) references
         ↪ Navigazione.Cliente(login)
4                on delete cascade       on update cascade;
5
6    alter table Navigazione.Biglietto
7        add constraint bigliettoFKveicolo
8            foreign key (Veicolo) references
         ↪ Navigazione.Veicolo(targa)
9                on delete set null      on update cascade;
```

Aggiunti vincoli di chiave esterna per la tabella Biglietto

```
1    alter table Navigazione.Veicolo
2        add constraint veicoloFKproprietario
3            foreign key (Proprietario) references
         ↪ Navigazione.Cliente(login)
4                on delete cascade       on update cascade;
```

Aggiunto vincolo di chiave esterna per la tabella Veicolo

```
1    alter table Navigazione.AccountSocial
2        add constraint accountFKcompagnia
3            foreign key (Compagnia) references
         ↪ Navigazione.Compagnia(login)
4                on delete cascade       on update cascade;
```

Aggiunto vincolo di chiave esterna per la tabella AccountSocial

```
    alter table Navigazione.Email
        add constraint emailFKcompagnia
            foreign key (Compagnia) references
        ↪ Navigazione.Compagnia(login)
                on delete cascade       on update cascade;
```

Aggiunto vincolo di chiave esterna per la tabella Email

```
    alter table Navigazione.Telefono
        add constraint telefonoFKcompagnia
            foreign key (Compagnia) references
        ↪ Navigazione.Compagnia(login)
                on delete cascade       on update cascade;
```

Aggiunto vincolo di chiave esterna per la tabella Telefono

4.6 Trigger e trigger function

La descrizione dei trigger è riportata nella sezione Vincoli Inter-Relazionali.

```
-- trigger per aggiornare i posti (per passeggeri)
        ↪ disponibili per una corsa specifica
create function aggiornapostipasseggero() returns trigger
    language plpgsql
as
$$
declare
    v_corsa_sup navigazione.corsaregolare.corsasup%type;
begin
    select corsasup into v_corsa_sup
    from navigazione.corsaregolare
    where idcorsa = new.idcorsa;

    raise notice 'corsa sup %', v_corsa_sup;
    --se il biglietto acquistato e' per una corsa principale
    if v_corsa_sup is null then
        --aggiorna i posti disponibili anche per le
        ↪ sottocorse
        update navigazione.corsaspecifica
        set postidisppass = postidisppass - 1
        where data = new.data and idcorsa in (select
        ↪ CR.idcorsa
                                              from
        ↪ navigazione.corsaregolare as CR
                                              where
        ↪ CR.corsasup = new.idcorsa or CR.idcorsa =
        ↪ new.idcorsa);
    else --se invece il biglietto acquistato e' per una
        ↪ sottotratta
```

```
        --aggiorna i posti per la sottotratta in questione
        update navigazione.corsaspecifica
        set postidisppass = postidisppass - 1
        where idcorsa = new.idCorsa and data = new.data;

        --aggiorna i posti per la corsa principale
        update navigazione.corsaspecifica
        set postidisppass = postidisppass - 1
        where idcorsa = v_corsa_sup and data = new.data;
    end if;

    return null;
end;
$$;
----------
create trigger triggeraggiornapostipasseggero
    after insert
    on biglietto
    for each row
execute procedure aggiornapostipasseggero();
```

```
-- trigger per aggiornare i posti (per veicoli) disponibili per una corsa specifica
create function aggiornapostiveicolo() returns trigger
    language plpgsql
as
$$
declare
    v_corsa_sup navigazione.corsaregolare.corsasup%type;
begin
    select corsasup into v_corsa_sup
    from navigazione.corsaregolare
    where idcorsa = new.idcorsa;

    --se il biglietto acquistato e' per una corsa principale
    if v_corsa_sup is null then
        --aggiorna i posti disponibili anche per le sottocorse
        update navigazione.corsaspecifica
        set postidispvei = postidispvei - 1
        where data = new.data and idcorsa in (select CR.idcorsa
                                              from navigazione.corsaregolare as CR
                                              where CR.corsasup = new.idcorsa or CR.idcorsa = new.idcorsa);
    else --se invece il biglietto acquistato e' per una sottotratta
```

```
        --aggiorna i posti per la sottotratta in questione
        update navigazione.corsaspecifica
        set postidispvei = postidispvei - 1
        where idcorsa = new.idCorsa and data = new.data;

        --aggiorna i posti per la corsa principale
        update navigazione.corsaspecifica
        set postidispvei = postidispvei - 1
        where idcorsa = v_corsa_sup and data = new.data;
    end if;

    return null;
end;
$$;
----------
create trigger triggeraggiornapostiveicolo
    after insert
    on biglietto
    for each row
    when (new.veicolo IS NOT NULL)
execute procedure aggiornapostiveicolo();
```

```
-- all'inserimento di uno scalo per una corsaregolare,
        ↪ questo trigger si occupa di generare le
        ↪ sottocorse e di attivarle nei periodi
--in cui e' attiva la corsa principale.
create function aggiungicorsescalo() returns trigger
    language plpgsql
as
$$
declare
    v_record navigazione.corsaregolare%rowtype;
    v_ultimo_idcorsa navigazione.corsaregolare.idcorsa%type;
    --seleziona tutti i periodi in cui e' attiva la corsa
        ↪ principale
    cur_periodi cursor for
        select idperiodo
        from navigazione.attivain
        where idcorsa = new.idcorsa;
begin
    --seleziona in corsaRegolare l'intera tupla
        ↪ corrispondente alla corsa principale
    select * into v_record
    from navigazione.corsaregolare
    where idcorsa = new.idcorsa;

    --inserisce in corsaRegolare la prima sottotratta che
        ↪ parte dal porto di partenza della corsa principale
    --e arriva al porto di scalo
```

```
23      insert into navigazione.corsaregolare(portopartenza,
            ↪ portoarrivo, orariopartenza, orarioarrivo,
24                                              costointero,
            ↪ scontoridotto, costobagaglio, costoprevendita,
25                                              costoveicolo,
            ↪ compagnia, natante, corsasup)
26      values
27          (v_record.portopartenza, new.porto,
            ↪ v_record.orariopartenza, new.orarioAttracco,
            ↪ v_record.costointero,
28           v_record.scontoridotto, v_record.costobagaglio,
            ↪ v_record.costoprevendita, v_record.costoveicolo,
29           v_record.compagnia, v_record.natante, new.idcorsa);
30
31      --seleziona quest'ultima corsa inserita
32      select idcorsa into v_ultimo_idcorsa
33      from navigazione.corsaregolare
34      order by idcorsa desc
35      limit 1;
36
37      --e l'attiva in tutti i periodi in cui era attiva la
            ↪ corsa principale
38      for p in cur_periodi loop
39          insert into navigazione.attivain
40          values (v_ultimo_idcorsa, p.idperiodo);
41      end loop;
42
43      --inserisce in corsaRegolare la seconda sottotratta che
            ↪ parte dal porto di scalo
44      --e arriva al porto di arrivo della corsa principale
45      insert into navigazione.corsaregolare(portopartenza,
            ↪ portoarrivo, orariopartenza, orarioarrivo,
46                                              costointero,
            ↪ scontoridotto, costobagaglio, costoprevendita,
47                                              costoveicolo,
            ↪ compagnia, natante, corsasup)
48      values
49          (new.porto, v_record.portoarrivo,
            ↪ new.orarioRipartenza, v_record.orarioarrivo,
            ↪ v_record.costointero,
50           v_record.scontoridotto, v_record.costobagaglio,
            ↪ v_record.costoprevendita, v_record.costoveicolo,
51           v_record.compagnia, v_record.natante, new.idcorsa);
52
53      --seleziona quest'ultima corsa inserita
54      select idcorsa into v_ultimo_idcorsa
55      from navigazione.corsaregolare
56      order by idcorsa desc
57      limit 1;
```

```
58
59      --e l'attiva in tutti i periodi in cui era attiva la
            ↪ corsa principale
60      for p in cur_periodi loop
61              insert into navigazione.attivain
62              values (v_ultimo_idcorsa, p.idperiodo);
63      end loop;
64
65      return new;
66  end;
67  $$;
68  ----------
69  create trigger generatrattescalotrigger
70      after insert
71      on scalo
72      for each row
73  execute procedure aggiungicorsescalo();
```

```
1   -- trigger per attivare nei periodi della corsa principale
        ↪ anche le sue sottocorse
2   create function attivasottocorse() returns trigger
3       language plpgsql
4   as
5   $$
6   declare
7       --seleziona le due sottocorse della corsa in questione
8       cur_sottocorse cursor for
9           select idcorsa
10          from navigazione.corsaregolare
11          where corsasup = new.idcorsa;
12  begin
13      --e attiva ognuna nel nuovo periodo inserito
14      for t in cur_sottocorse loop
15          insert into navigazione.attivain
16          values (t.idcorsa, new.idperiodo);
17      end loop;
18
19      return new;
20  end;
21  $$;
22  ----------
23  create trigger attivasottocorsetrigger
24      after insert
25      on attivain
26      for each row
27  execute procedure attivasottocorse();
```

```
1   --trigger che quando viene cambiato l'orario di arrivo della
        ↪ corsa principale
```

```
--aggiorna l'orario di arrivo anche della sottocorsa che parte dal porto di scalo
create function cambiaorarioarrivoinsottocorsa() returns trigger
    language plpgsql
as
$$
declare
    sottocorsa navigazione.corsaregolare%rowtype;
begin
    --seleziona la sottocorsa che arriva allo stesso orario della corsa principale
    select * into sottocorsa
    from navigazione.corsaregolare
    where corsasup = old.idcorsa and orarioarrivo = old.orarioarrivo;

    --aggiorna l'orario di arrivo
    update navigazione.corsaregolare
    set orarioarrivo = new.orarioarrivo
    where idcorsa = sottocorsa.idcorsa;

    return new;
end;
$$;
----------
create trigger cambiaorarioarrivoinsottocorsa
    after update
        of orarioarrivo
    on corsaregolare
    for each row
    --quando e' una corsa principale
    when (new.corsasup IS NULL AND new.orarioarrivo IS NOT NULL)
execute procedure cambiaorarioarrivoinsottocorsa();
```

```
--trigger che quando viene cambiato l'orario di partenza della corsa principale
--aggiorna l'orario di partenza anche della sottocorsa che parte dallo stesso porto
create function cambiaorariopartenzainsottocorsa() returns trigger
    language plpgsql
as
$$
declare
    sottocorsa navigazione.corsaregolare%rowtype;

begin
```

```
    --seleziona la sottocorsa che parte allo stesso orario della corsa principale
    select * into sottocorsa
    from navigazione.corsaregolare
    where corsasup = old.idcorsa and orariopartenza = old.orariopartenza;

    --aggiorna l'orario di partenza
    update navigazione.corsaregolare
    set orariopartenza = new.orariopartenza
    where idcorsa = sottocorsa.idcorsa;

    return new;
end;
$$;
----------
create trigger cambiaorariopartenzainsottocorsa
    after update
        of orariopartenza
    on corsaregolare
    for each row
    when (new.corsasup IS NULL AND new.orariopartenza IS NOT NULL)
execute procedure cambiaorariopartenzainsottocorsa();
```

```
--trigger per eliminare le corse specifiche di una corsa regolare non piu' attiva in un periodo
--ed eliminare i periodi che non sono attaccati a nessuna corsa
create function cancellacorseinperiodo() returns trigger
    language plpgsql
as
$$
declare
    v_data_inizio navigazione.periodo.datainizio%type;
    v_data_fine navigazione.periodo.dataFine%type;
    v_data_corrente date;
    v_giorni bit(7);
    v_day integer;
    v_offset integer;
    --trovo i periodi che non sono attaccati a nessuna corsa
    cur_periodi cursor for
        select P.idperiodo
        from navigazione.periodo as P
        where P.idperiodo not in (select idperiodo
                                  from navigazione.attivain);
begin
    select datainizio, datafine, giorni into v_data_inizio, v_data_fine, v_giorni
```

```
22      from navigazione.periodo
23      where idperiodo = old.idperiodo;
24
25      --ricava un numero da 0 a 6 corrispondente al giorno
            ↪ della settimana in cui inizia il periodo
26      -- 0 se domenica, 1 se lunedi' ... fino a 6 se sabato
27      v_day := extract(dow from v_data_inizio::timestamp);
28      --per ogni bit della stringa di bit che indica i giorni
            ↪ di attivita'
29      for i in 0..6 loop
30          --ogni iterazione del for-loop parte con
            ↪ v_data_corrente uguale alla data di inizio
31          v_data_corrente := v_data_inizio;
32          --se l'i-esimo bit e' 1
33          if get_bit(v_giorni, i) = 1 then
34              v_offset := (i - v_day);
35              if v_offset < 0 then
36                  while v_offset < 0 loop
37                          v_offset := v_offset + 7;
38                  end loop;
39              else
40                  v_offset := v_offset % 7;
41              end if;
42              --aggiunge l'offset alla data corrente affinche'
            ↪ la data corrente
43              --si trovi nella prima data maggiore o uguale la
            ↪ data di inizio,
44              --che abbia come giorno della settimana
            ↪ l'i-esimo giorno
45              v_data_corrente := v_data_corrente + v_offset;
46              --per tutte le date con quel giorno incluse nel
            ↪ periodo
47              while v_data_corrente <= v_data_fine loop
48                  --elimina la corsa specifica per quella data
49                  delete from navigazione.corsaspecifica
50                  where idcorsa = old.idcorsa AND data =
            ↪ v_data_corrente;
51                  --incrementa la data alla settimana
            ↪ successiva
52                  v_data_corrente := v_data_corrente + 7;
53              end loop;
54          end if;
55      end loop;
56
57      --elimina i periodi che non sono attaccati a nessuna
            ↪ corsa
58      for p in cur_periodi loop
59          delete from navigazione.periodo
60          where idperiodo = p.idperiodo;
```

```
    end loop;

    return null;
end;
$$;
----------
create trigger cancellacorse
    after delete
    on attivain
    for each row
execute procedure cancellacorseinperiodo();
```

```
--Trigger per eliminare le sottocorse dopo che e' stato
        ↪ eliminato uno scalo
create function eliminacorsescalo() returns trigger
    language plpgsql
as
$$
begin
    delete from navigazione.corsaregolare
    where corsasup = old.idcorsa AND (portopartenza =
        ↪ old.porto OR portoarrivo = old.porto);

    return null;
end;
$$;
----------
create trigger eliminatrattescalotrigger
    after delete
    on scalo
    for each row
execute procedure eliminacorsescalo();
```

```
--Trigger per eliminare l'altra sottocorsa dopo che ne e'
        ↪ stata eliminata una, poiche' non puo' esistere
        ↪ una senza l'altra.
create function eliminasottocorse() returns trigger
    language plpgsql
as
$$
declare
    v_altra_corsa navigazione.corsaregolare.idcorsa%type;
begin
    if old.corsasup is not null then
        --seleziona la sottocorsa rimanente
        select idcorsa into v_altra_corsa
        from navigazione.corsaregolare
        where corsasup = old.corsasup;

```

```
        if v_altra_corsa is not null then
            --e la cancella
            delete from navigazione.corsaregolare
            where idcorsa = v_altra_corsa;
        end if;

        --elimina la tupla in scalo che aveva generato le
        ↪ due sottocorse
        delete from navigazione.scalo
        where idcorsa = old.corsasup;
    end if;
    return null;
end;
$$;
----------
create trigger eliminaaltrasottocorsa
    after delete
    on corsaregolare
    for each row
    --quando la corsa eliminata e' una sottocorsa
    when (old.corsasup IS NOT NULL)
execute procedure eliminasottocorse();
```

```
-- trigger per la generazione delle corse specifiche in
        ↪ tutte le date presenti nel periodo di attivazione
create function generacorsespecifiche() returns trigger
    language plpgsql
as
$$
declare
    v_data_inizio date;
    v_data_corrente date;
    v_data_fine date;
    v_giorni bit(7);
    v_natante navigazione.natante.nome%type;
    v_cap_pass integer;
    v_cap_veic integer;
    v_day integer;
    v_offset integer;
begin
    --seleziona le date di inizio e fine e i giorni del
        ↪ periodo
    select dataInizio, dataFine, giorni into v_data_inizio,
        ↪ v_data_fine, v_giorni
    from navigazione.periodo
    where idperiodo = new.idperiodo;

    --seleziona il natante che verra' utilizzato
    select natante into v_natante
```

```
24      from navigazione.corsaRegolare
25      where idcorsa = new.idcorsa;
26
27      --seleziona la capienza passeggeri e veicoli del natante
28      select capienzaPasseggeri, capienzaVeicoli into
            ↪ v_cap_pass, v_cap_veic
29      from navigazione.natante
30      where nome = v_natante;
31
32      --ricava un numero da 0 a 6 corrispondente al giorno
            ↪ della settimana in cui inizia il periodo
33      -- 0 se domenica, 1 se lunedi' ... fino a 6 se sabato
34      v_day := extract(dow from v_data_inizio::timestamp);
35      --per ogni bit della stringa di bit che indica i giorni
            ↪ di attivita'
36      for i in 0..6 loop
37          --ogni iterazione del for-loop parte con
            ↪ v_data_corrente uguale alla data di inizio
38          v_data_corrente := v_data_inizio;
39          --se l'i-esimo bit e' 1
40          if get_bit(v_giorni, i) = 1 then
41              v_offset := (i - v_day);
42              if v_offset < 0 then
43                  while v_offset < 0 loop
44                      v_offset := v_offset + 7;
45                  end loop;
46              else
47                  v_offset := v_offset % 7;
48              end if;
49              --aggiunge l'offset alla data corrente affinche'
            ↪ la data corrente
50              --si trovi nella prima data maggiore o uguale la
            ↪ data di inizio,
51              --che abbia come giorno della settimana
            ↪ l'i-esimo giorno
52              v_data_corrente := v_data_corrente + v_offset;
53              --per tutte le date con quel giorno incluse nel
            ↪ periodo
54              while v_data_corrente <= v_data_fine loop
55                  --aggiunge la corsa specifica per quella data
56                  insert into navigazione.corsaSpecifica
57                  values(new.idcorsa, v_data_corrente,
            ↪ v_cap_pass, v_cap_veic, 0, false);
58                  --incrementa la data alla settimana
            ↪ successiva
59                  v_data_corrente := v_data_corrente + 7;
60              end loop;
61          end if;
62      end loop;
```

```

    return new;
end;
$$;
----------
create trigger generacorse
    after insert
    on attivain
    for each row
execute procedure generacorsespecifiche();
```

```
-- trigger per cancellare le sottocorse figlie (quando viene
    ↪ cancellata una corsa principale), e la
-- sottocorsa sorella (quando viene cancellata una
    ↪ sottocorsa).
create function propagacancellazione() returns trigger
    language plpgsql
as
$$
declare
    thisData date := old.data;
    CorsaRegolareRiferita navigazione.corsaregolare%rowtype;
    isCancellata boolean;
    it1 cursor for select *
                   from  navigazione.corsaregolare
                   where corsasup =
        ↪ CorsaRegolareRiferita.idcorsa;
    it2 cursor for select *
                    from navigazione.corsaregolare
                    where corsasup =
        ↪ CorsaRegolareRiferita.corsasup and idcorsa <>
        ↪ CorsaRegolareRiferita.idcorsa;
begin
    --seleziona la corsa regolare della corsa in questione
    select * into CorsaRegolareRiferita from
        ↪ navigazione.corsaregolare where idcorsa =
        ↪ old.idcorsa;
    --se e' una corsa principale
    if (CorsaRegolareRiferita.corsasup is null) then
        ↪ --potrebbe avere delle sottocorse
        --per entrambe le sottocorse
        for x in it1 loop
            select cancellata into isCancellata
            from navigazione.corsaspecifica
            where idcorsa = x.idcorsa and data = thisData;
            --se non sono gia' cancellate le cancella
            if isCancellata = false then
                update navigazione.corsaspecifica
                set cancellata = true
```

```
                where idcorsa = x.idcorsa and data =
        ↪ thisData;
            end if;
        end loop;
    else --se invece e' una sottocorsa ha sicuramente una
        ↪ sorella
        for x in it2 loop
            --e cancella entrambe
            select cancellata into isCancellata
            from navigazione.corsaspecifica
            where idcorsa = x.idcorsa and data = thisData;
            if isCancellata = false then
                update navigazione.corsaspecifica
                set cancellata = true
                where idcorsa = x.idcorsa and data =
        ↪ thisData;
            end if;
        end loop;
    end if;
    return new;
end;
$$;
----------
create trigger propagacancellazione
    after update
        of cancellata
    on corsaspecifica
    for each row
    when (new.cancellata = true)
execute procedure propagacancellazione();
```

4.7 Procedure

```
--Procedura per calcolare gli incassi di una corsa regolare
        ↪ in un determinato arco di tempo
create procedure calcolaincassicorsainperiodo(IN thisidcorsa
        ↪ integer, IN ini_periodo date, IN fin_periodo
        ↪ date, OUT incasso double precision)
    language plpgsql
as
$$
begin
    select sum(prezzo) into incasso
    from navigazione.biglietto
    where idcorsa = thisidcorsa and data between ini_periodo
        ↪ and fin_periodo;
end;
$$;
```

```
--Procedura per fare in modo che quando venga creato uno
        ↪ scalo in una corsa regolare, per ogni data
--in cui e' attiva, le sottocorse create abbiano gli stessi
        ↪ posti disponibili della corsa madre.
create procedure aggiornapostisottocorse(IN idcorsa_in
        ↪ integer)
    language plpgsql
as
$$
declare
    --seleziona le sottocorse
    cur_sottocorse cursor for
        select CR.idcorsa
        from navigazione.corsaregolare as CR
        where CR.corsasup = idCorsa_in;

    --seleziona i dati che vanno aggiornati nelle sottocorse
    cur_postidisp cursor for
        select CS.idcorsa, CS.data, CS.postidisppass,
        ↪ CS.postidispvei, CS.cancellata
        from navigazione.corsaspecifica as CS
        where CS.idcorsa = idCorsa_in;
begin
    for k in cur_postidisp loop
        for i in cur_sottocorse loop
            --aggiorna i valori delle sottocorse
            update navigazione.corsaspecifica
            set postidisppass = k.postidisppass,
        ↪ postidispvei = k.postidispvei, cancellata =
        ↪ k.cancellata
            where idcorsa = i.idcorsa AND data = k.data;
        end loop;
    end loop;
end;
$$;
```

4.8 Dizionario dei vincoli

4.8.1 Vincoli Intra-Relazionali

Vincolo	**Descrizione**
accountsocial_pkey	Vincolo di chiave primaria
accountfkcompagnia	Vincolo di chiave esterna. Se viene eliminata la compagnia vengono eliminati anche tutti i suoi account social.

Tabella AccountSocial

Vincolo	**Descrizione**
attivain_pkey	Vincolo di chiave primaria
attivain_idcorsa_fkey	Vincolo di chiave esterna. Se viene eliminata la corsa regolare vengono eliminate anche le tuple associate in AttivaIn.
attivain_idperiodo_fkey	Vincolo di chiave esterna. Se viene eliminato un periodo vengono eliminate anche le tuple associate in AttivaIn.

Tabella AttivaIn

Vincolo	**Descrizione**
biglietto_pkey	Vincolo di chiave primaria
biglietto_idcorsa_data_fkey	Vincolo di chiave esterna. Se viene eliminata una corsa specifica anche i biglietti associati verranno eliminati
bigliettofkcliente	Vincolo di chiave esterna. Se viene eliminato un cliente anche i biglietti associati verrano eliminati
bigliettofkveicolo	Vincolo di chiave esterna. Se viene eliminato un veicolo il valore in biglietto sarà settato a null
biglietto_etapasseggero_check	Controlla che l'età del passeggero sia un numero non negativo.

biglietto_prezzo_check	Controlla che il prezzo del biglietto sia un numero non negativo.

Tabella Biglietto

Vincolo	**Descrizione**
cliente_pkey	Vincolo di chiave primaria.

Tabella Cliente

Vincolo	**Descrizione**
compagnia_pkey	Vincolo di chiave primaria.

Tabella Compagnia

Vincolo	**Descrizione**
corsaregolare_pkey	Vincolo di chiave primaria.
corsafkcompagnia	Vincolo di chiave esterna. Se viene eliminata la compagnia verrano eliminati anche tutte le sue corse regolari.
corsafkcorsasup	Vincolo di chiave esterna. Se viene eliminata la corsa principale anche le sottocorse verranno eliminate.
corsafkportoarrivo	Vincolo di chiave esterna. Se viene eliminato il porto di arrivo, verranno eliminate tutte le corse che arrivano in quel porto.
corsafkportopartenza	Vincolo di chiave esterna. Se viene eliminato il porto di partenza, verranno eliminate tutte le corse che partivano da quel porto.

corsafknatante	Vincolo di chiave esterna. Se viene eliminato il natante, verrano eliminate tutte le corse che usavano quel natante.
corsaregolare_check	Controlla che il porto di arrivo sia diverso dal porto di partenza.
corsaregolare_costobagaglio_check	Controlla che il costo del bagaglio sia un numero non negativo.
corsaregolare_costointero_check	Controlla che il costo intero della corsa sia un numero non negativo.
corsaregolare_costoprevendita_check	Controlla che il costo della prevendita sia un numero non negativo.
corsaregolare_costoveicolo_check	Controlla che il costo del veicolo sia un numero non negativo.
corsaregolare_scontoridotto_check	Controlla che la percentuale di sconto sia un numero compreso tra 0 e 100.

Tabella CorsaRegolare

Vincolo	**Descrizione**
corsaspecifica_pkey	Vincolo di chiave primaria.
corsaspecifica_idcorsa_fkey	Vincolo di chiave esterna. Se viene eliminata una corsa regolare, tutte le corse specifiche associato verranno eliminate.
corsaspecifica_postidisppass_check	Controlla che i posti passeggeri disponibili sia un numero non negativo
corsaspecifica_postidisppass_check	Controlla che i posti veicoli disponibili siano un numero non negativo

Tabella CorsaSpecifica

Vincolo	Descrizione
email_pkey	Vincolo di chiave primaria.
emailfkcompagnia	Vincolo di chiave esterna. Se una compagnia viene eliminata anche tutte le sue email verrano eliminate.

Tabella Email

Vincolo	Descrizione
natante_pkey	Vincolo di chiave primaria.
natantefkcompagnia	Vincolo di chiave esterna. Se una compagnia viene eliminata anche i natanti da essa posseduta verranno eliminati.

Tabella Natante

Vincolo	Descrizione
periodo_pkey	Vincolo di chiave primaria.
periodo_check	Controlla che la data di inizio periodo sia minore o uguale alla data di fine periodo.

Tabella Periodo

Vincolo	Descrizione
porto_pkey	Vincolo di chiave primaria.

Tabella Porto

Vincolo	Descrizione
scalo_pkey	Vincolo di chiave primaria.
scalofkcorsa	Vincolo di chiave esterna. Se una corsa viene eliminata, anche lo scalo associato viene eliminato.
scalofkporto	Vincolo di chiave esterna. Se un porto viene eliminato, anche gli scali in quel porto verranno eliminati.

scalo_check	Controlla che l'orario di attracco sia minore dell'orario di ripartenza.

Tabella Scalo

Vincolo	**Descrizione**
telefono_pkey	Vincolo di chiave primaria.
telefonofkcompagnia	Vincolo di chiave esterna. Se una compagnia viene eliminata anche i suoi numeri telefonici verranno eliminati.

Tabella Telefono

Vincolo	**Descrizione**
veicolo_pkey	Vincolo di chiave primaria.
veicolofkproprietario	Vincolo di chiave esterna. Se un cliente viene eliminato anche tutti i suoi veicoli verranno eliminati.

Tabella Veicolo

4.8.2 Vincoli Inter-Relazionali

L'implementazione dei trigger è riportata nella sezione Trigger e trigger function.

Trigger	**Descrizione**
attivaSottoCorseTrigger	Dopo aver inserito una tupla in *AttivaIn*, anche le eventuali sottocorse della corsa inserita verranno inserite in *AttivaIn*.
cancellaCorseTrigger	Dopo aver eliminato una una tupla in *AttivaIn*, vengono eliminate tutte le corse specifiche relative a quella corsa, in tutte le date appartenenti a quel periodo

generaCorse	Dopo aver inserito una tupla in *AttivaIn*, per tutte le date appartenti al periodo in questione verranno inserite delle corse specifiche.
triggerAggiornaPostiPasseggero	Dopo aver inserito una tupla in *Biglietto*, il trigger aggiorna i posti disponibili per quella corsa specifica, inoltre se la corsa in questione è una corsa principale, il trigger aggiornerà i posti anche per le sottocorse, se invece la corsa è una sottocorsa, il trigger aggionerà i posti anche per la corsa principale ma non per l'altra sottocorsa.
triggerAggiornaPostiVeicolo	Dopo aver inserito una tupla in *Biglietto*, il trigger aggiorna i posti disponibili per quella corsa specifica, inoltre se la corsa in questione è una corsa principale, il trigger aggiornerà i posti anche per le sottocorse, se invece la corsa è una sottocorsa, il trigger aggionerà i posti anche per la corsa principale ma non per l'altra sottocorsa. Il trigger si attiva solo quando il valore di *veicolo* è diverso da null.
cambiaOrarioArrivoInSottocorsa	Dopo aver modificato il valore di *orarioArrivo* in *CorsaRegolare*, il trigger modificherà il valore anche nelle eventuali sottocorse. Il trigger si attiva quindi solo quando *corsaSup* è null.
cambiaOrarioPartenzaInSottocorsa	Dopo aver modificato il valore di *orarioPartenza* in *CorsaRegolare*, il trigger modificherà il valore anche nelle eventuali sottocorse. Il trigger si attiva quindi solo quando *corsaSup* è null.
eliminaAltraSottocorsa	Dopo aver eliminato una tupla in *CorsaRegolare* e quando *corsaSup* è diverso da null, quindi si è appena eliminata una sottocorsa, il trigger andrà ad eliminare anche l'altra sottocorsa che ha lo stesso valore di *corsaSup*.

propagaCancellazione	Dopo aver settato a *true* il valore di *cancellata* in *CorsaSpecifica*, il trigger andrà a settare a *true* anche il valore di *cancellata* delle corse specifiche che sono sottocorse della corsa in questione. Se invece la corsa cancellata è una sottocorsa, il trigger andrà a cancellare anche l'altra sottocorsa, dal momento che una sottocorsa non può esistere senza l'altra.
eliminaTratteScalo	Dopo aver eliminato una tupla in *Scalo*, il trigger elimina le sottocorse della corsa in questione.
generaTratteScalo	Dopo aver inserito una tupla in *Scalo*, il trigger genera le due sottocorse, una avrà come porto di partenza lo stesso della corsa inserita e come porto di arrivo il porto di scalo, l'altra invece avrà come porto di partenza il porto di scalo e come porto di arrivo il porto di arrivo della corsa inserita.