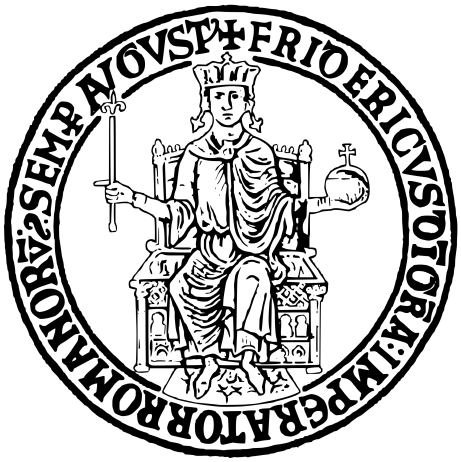


Università degli Studi di Napoli Federico II

Documentazione del progetto per Basi di dati

Traccia 2: gruppo da tre persone

Iavarone Simone
Illiano Eliana
Lamore Antonio

Corso: Basi di Dati I

Indice

1 Descrizione traccia del progetto

Si sviluppa un sistema informativo che consente ad un viaggiatore di conoscere e scegliere i viaggi utili per raggiungere le isole.

La base di dati progettata, permette di contenere tutte le corse con eventuali scali, di ogni singola compagnia. Ogni compagnia può fornire diversi natanti associati ad una corsa, tra questi ci possono essere traghetti, aliscafi, motonavi oppure altre tipologie non specificate.

Nel nostro minimondo, ad un passeggero è data la possibilità di effettuare una prenotazione individuale di una corsa, riservandogli un identificativo univoco associato alla prenotazione che effettua, per una corsa specifica legata ad un natante. Alla prenotazione è associato anche un biglietto, che può essere intero o ridotto, su questo viene riportato il prezzo e il nominativo del passeggero. All'interno del database la distinzione tra biglietto intero e ridotto viene fatta sulla base dell'età del passeggero, se questo ha meno di 18 anni rientra nell'acquisto ridotto del biglietto, altrimenti in quello intero.

Al termine della prenotazione, la disponibilità della corsa andrà a diminuire, a causa dell'occupazione del posto.

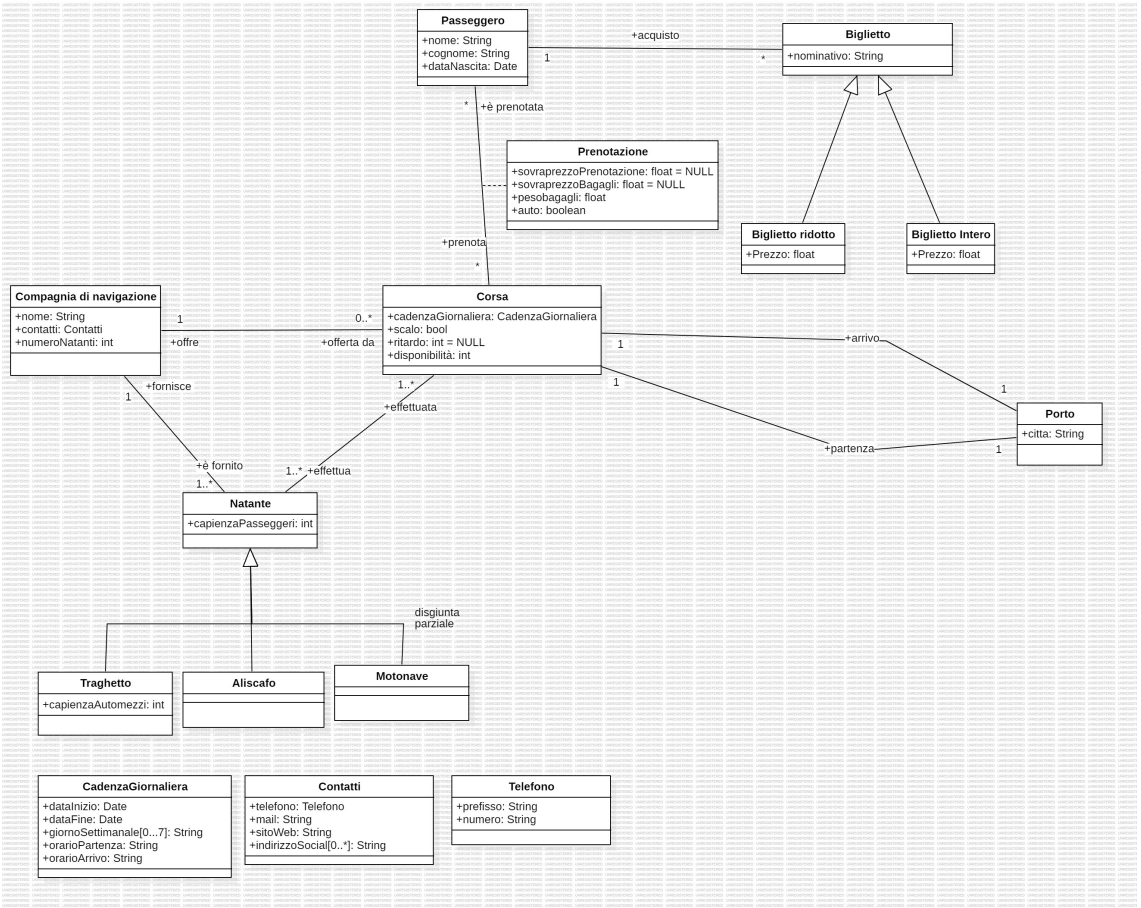
La progettazione prevede anche la possibilità di eliminare una prenotazione, cancellando di conseguenza anche l'acquisto e il biglietto associati, e aumentando nuovamente la disponibilità della corsa.

2 Diagramma Class Diagram della base di dati

2.1 Progettazione Concettuale

Nella fase iniziale del progetto, si analizzano le informazioni utili per progettare la base di dati. Si procede ad individuare le diverse classi e gli attributi, e successivamente collegarle tramite associazioni.

2.2 Class Diagram



3 Ristrutturazione del Class Diagram

La ristrutturazione del Class Diagram prevede sei fasi:

3.1 Analisi delle ridondanze

All'interno della progettazione non sono presenti ridondanze.

3.2 Eliminazione degli attributi multivalore

Gli attributi multivalore presenti nel Class Diagram sono: *giornoSettimanale* nella classe cadenzaGiornaliera, e *indirizzoSocial* in Contatti. Per eliminare un attributo multivalore, abbiamo tre possibilità:

- la prima, consiste nella creazione di una classe esterna associata alla classe che conteneva l'attributo;
- la seconda, tratta l'attributo multivalore come se fosse singolo, quindi tutti i valori possibili vengono memorizzati in un unico attributo;
- la terza, prevede la replica dell'attributo più volte nella stessa classe. Questo potrebbe però portare alla perdita di alcuni valori nel caso in cui ci fossero più dati da memorizzare, oppure ad un numero elevato di valori a null.

Per l'attributo *giornoSettimanale*, si è pensato fosse opportuno trattarlo come se fosse singolo, perché è possibile selezionare i giorni della settimana anche da una sola stringa.

Invece, l'attributo *indirizzoSocial*, viene trattato come una classe esterna associata, data la possibilità di avere molteplici istanze associate a questa classe.

3.3 Eliminazione degli attributi complessi

L'eliminazione degli attributi complessi può avere tre diverse soluzioni:

- la creazione di una classe apposita per l'attributo;
- gli attributi dell'attributo complesso vengono portati nella classe in cui era contenuto;
- la struttura dell'attributo viene trascurata, quindi i valori vengono rappresentati in un'unica stringa.

Nel Class Diagram abbiamo i seguenti attributi complessi: *cadenzaGiornaliera*, *Contatti* e *Telefono*. L'attributo *cadenzaGiornaliera* viene reso una classe esterna collegata tramite un'associazione alla classe *Corsa*. Questa scelta è dovuta alla possibilità di dover memorizzare in una cadenzaGiornaliera più corse. La struttura dell'attributo *Telefono* viene trascurata. Viene quindi reso come un attributo semplice, dato che non ci interessa distinguere il prefisso dal numero. Gli attributi di *Contatti* vengono portati nella classe *Compagnia di navigazione*, considerando la rara possibilità che una compagnia abbia più di un valore per ciascuno degli attributi e considerando anche che possa non avere dei contatti.

3.4 Eliminazione delle generalizzazioni

Le generalizzazioni presenti sono due: la prima, tra la classe padre *Natante* e le classi figlie *Traghetto*, *Aliscafo* e *Motonave*; la seconda, tra la classe padre *Biglietto* e le classi figlie *Biglietto intero* e *Biglietto ridotto*. La generalizzazione tra Natante e le diverse classi figlie, è intesa come disgiunta parziale, quindi un natante può essere o un traghetto, o un aliscafo, o una motonave, o anche un'altra tipologia di imbarcazione. In questo caso, le classi figlie vengono portate nella classe padre, e viene aggiunto un attributo *tipoNatante*, per indicarne la tipologia. Inizialmente, avendo escluso la possibilità di ulteriori natanti, questa generalizzazione veniva intesa come disgiunta totale, si pensava perciò di legarle tramite un'associazione. Tuttavia, avendo chiarito la possibilità di ulteriori natanti all'interno del database è stata fatta una scelta diversa.

La generalizzazione tra Biglietto e le classi figlie è intesa come disgiunta totale, quindi la soluzione ritenuta più opportuna, è quella di portare la classe padre nelle figlie. In alternativa avremmo potuto portare le classi figlie nel padre, ma questo comportava molti valori a null; oppure avremmo potuto sostituire la generalizzazione con associazioni, tuttavia ci sarebbe stata la possibilità di numerosi accessi inutili.

3.5 Accorpamento/Partizionamento di classi e associazioni

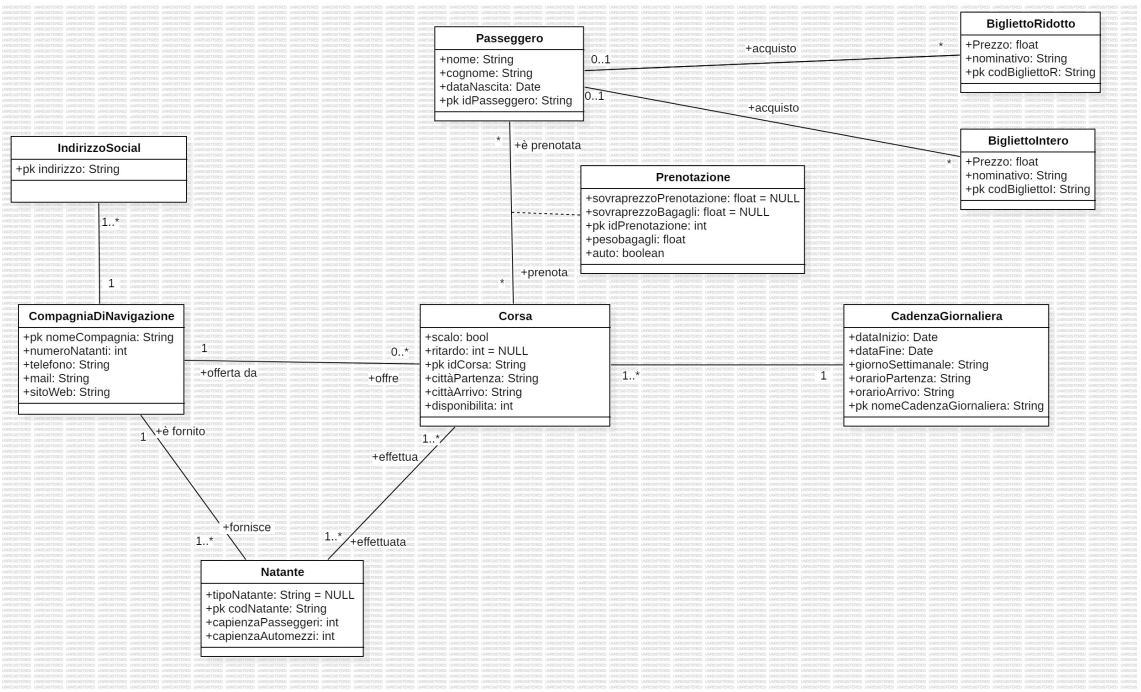
L'accorpamento di classi si verifica generalmente con associazioni 1:1. Nel nostro Class Diagram, le uniche associazioni che possiamo accorpare sono quelle fra *Corsa* e *Porto*. Portiamo quindi gli attributi di *Porto* in *Corsa*.

3.6 Identificazione di chiavi primarie

Procediamo a scegliere un attributo per identificare univocamente le diverse classi: nelle classi *Compagnia di navigazione e indirizzosocial* sono già presenti attributi che le identificano univocamente, e sono rispettivamente *nomeCompagnia* e *indirizzo*. Nelle classi *Corsa, Passeggero,*

Biglietto_intero e Biglietto_ridotto, cadenzaGiornaliera e Prenotazione abbiamo aggiunto un attributo per la chiave primaria, rispettivamente *idCorsa, idPasseggero, codBigliettoI, codBigliettoR, nomeCadenzaGiornaliera e idPrenotazione.*

3.7 Class Diagram ristrutturato



4 Dizionario delle classi e degli attributi del Class Diagram ristrutturato

Classe	Attributi	Descrizioni
PASSEGGERO	**Nome** ***(String)***, **Cognome** ***(String)*** e **dataNascita** ***(Date)***: rappresentano i dati anagrafici del passeggero che prenota una corsa.	La classe rappresenta ogni utente che utilizza il programma.
BIGLIETTO INTERO	**Nominativo** ***(String)***: attributo che rappresenta i nominativi dei passeggeri che acquistano un biglietto intero. **Prezzo** ***(float)***: indica il prezzo pieno di un singolo biglietto.	Classe che rappresenta un biglietto standard per una corsa.
BIGLIETTO RIDOTTO	**Nominativo** ***(String)***: attributo che rappresenta i nominativi dei passeggeri che acquistano un biglietto intero. **Prezzo** ***(float)***: indica il prezzo di un singolo biglietto.	Classe che rappresenta un biglietto con un costo ridotto per una corsa.
NATANTE	**tipoNatante** ***(String)***: rappresenta la tipologia di natante presente nel minimondo. **CapienzaPasseggeri** ***(int)***: rappresenta il numero disponibile di posti in un natante. **CapienzaAutomezzi** ***(int)***: indica il numero disponibile di posti per veicoli all'interno del natante.	Classe che rappresenta tutti i possibili natanti forniti da una compagnia di navigazione. Questi possono essere traghetti, aliscafi, motonavi o altri natanti.
COMPAGNIA DI NAVIGAZIONE	**nomeCompagnia** ***(String)***: indica il nome della compagnia. **numeroNatanti** ***(int)***: numero dei natanti di cui dispone l'azienda. **Telefono**, attributo che rappresenta il numero di telefono di una compagnia, formato da un prefisso e da un numero. **mail** ***(String)***: posta elettronica per lo scambio di messaggi tramite internet. **sitoWeb** ***(String)***: insieme di pagine web per rappresentare informazioni e contenuti multimediali della compagnia.	E' una società che esegue il trasporto marittimo delle persone e/o merci utilizzando natanti di proprietà.

Classe	Attributi	Descrizioni
INDIRIZZO SOCIAL	**indirizzo** ***(String)***: attributo che rappresenta l'indirizzo social, utile per interagire con utenti e per lo scambio di informazioni.	L'indirizzo della pagina web che porta alla corrispondente pagina social della compagnia.
CORSA	**Scalo** ***(string)***: può essere null se non sono presenti scali oppure indicare direttamente la città in cui avviene lo scalo. **cittaPartenza**: citta da cui parte una corsa. **cittaArrivo**: citta di arrivo di una corsa. **disponibilità**: numero di posti disponibili di una corsa che possono diminuire in seguito ad una prenotazione o aumentare in seguito ad una cancellazione. **ritardo** ***(string)***:	La corsa è la tratta marittima fornita dalla compagnia da un porto di partenza ad un porto d'arrivo.
CADENZA GIORNALIERA	**dataInizio** ***(Date)***: l'inizio del periodo dell'anno in cui viene offerta la corsa. **dataFine** ***(Date)***: la fine del periodo dell'anno in cui viene offerta la corsa **giornoSettimanale** ***(String)***: indica i giorni della settimana in cui la tratta è disponibile; **orarioPartenza** ***(String)***: orario di partenza dal porto del natante; **orarioArrivo** ***(String)***: orario di arrivo in porto del natante.	Sono i giorni, in un determinato periodo dell'anno, in cui si verificano una o più corse.

5 Dizionario delle associazioni

Relazioni	**Descrizioni**
PRENOTAZIONE	Associazione N:N tra *Corsa* e *Passeggero*, in cui un passeggero può prenotare più corse, e una corsa può essere prenotata da più passeggeri. Questa associazione ha una classe associativa **Prenotazione** che indica eventuali sovrapprezzi, che si aggiungono se la prenotazione viene effettuata prima del periodo di attivazione della corsa, se una prenotazione comprende anche un'auto, oppure per i bagagli in base al loro peso.
ACQUISTO	Associazione 1:N tra *Passeggero* e *BigliettoRidotto*. In questo caso un passeggero può acquistare più biglietti ridotti, ma un biglietto ridotto può essere acquistato da 1 o nessun passeggero. Questa associazione è equivalente anche con la classe *BigliettoIntero*.
FREQUENZA	Associazione N:1 tra *Corsa* e *CadenzaGiornaliera*. Una corsa può essere effettuata in una sola cadenza giornaliera, e in una cadenza giornaliera possono essere ripetute più corse.
GESTIONE	Associazione 1:N tra *Compagnia di navigazione* e *Corsa*. Una compagnia di navigazione può offrire da 0 a N corse, mentre una corsa è offerta da una sola compagnia.
POSSIEDE	Associazione 1:N tra *compagnia di navigazione* e *indirizzoSocial*. Una compagnia può avere N indirizzi social e ogni indirizzo social è associato ad una sola compagnia.
EFFETTUA	Associazione N:N tra *Corsa* e *Natante*. Una corsa può essere effettuata da N natanti, e un natante può effettuare N corse.
DISPONE	Associazione 1:N tra *Compagnia di navigazione* e *Natante*. Una compagnia fornisce N natanti, un natante è fornito da una sola compagnia.

5.1 Mapping delle associazioni

- *Passeggero - acquisto - Bigliettointero/Bigliettoridotto*: viene creata una classe associativa *Acquisto*, che contiene la chiave primaria di Passeggero e quella di biglietto (intero o ridotto).
- *Corsa - frequenza - cadenzaGiornaliera*: la chiave primaria di cadenzaGiornaliera viene inserita nella classe corsa.
- *Compagnia di navigazione - gestione - corsa*: la chiave primaria di compagnia di navigazione viene inserita in corsa.
- *Compagnia di navigazione - possiede - indirizzo social*: la chiave primaria di compagnia di navigazione viene portata in corsa.
- *Compagnia di navigazione - dispone - natante*: la chiave primaria di compagnia di navigazione viene portata in natante.
- *Corsa - Prenotazione - Passeggero*: si crea la classe associativa Prenotazione quindi in quest'ultima vengono riportate le chiavi primarie di corsa e passeggero.
- *Corsa - effettua - Natante*: si crea la classe associativa nominata *Navigazione*, dove vengono riportate le chiavi primarie di Corsa e Natante.

6 Dizionario dei vincoli

Vincoli	**Descrizioni**
ck_tiponatante	vincolo di check che limita la scelta del tipo di natante a "traghetto", "aliscafo", "motonave" o "altro".
ck_capienzapasseggeri	vincolo di check che indica che la capienza passeggeri di un natante deve essere maggiore di 0.
numeronatanti default 0	il numero dei natanti viene messo di default a 0 poichè viene aumentato successivamente con l'inserimento dei natanti di una compagnia nella tabella apposita.
datanascita not null	è utile specificare che questo attributo venga inserito obbligatoriamente perché in base all'età del passeggero, viene acquistato un biglietto a prezzo intero o ridotto.
sovrapprezzo prenotazione default 3.00	viene inserito un valore di default per il sovrapprezzo della prenotazione nell'eventualità che un passeggero effettui una prenotazione prima del periodo di attivazione della corsa.
idprenotazione	i valori di questo attributo sono generati da una sequenza specifica che parte da 1 (incluso).
mail, sitoweb, telefono unique	vincolo per indicare che il numero di telefono, la mail e il sito web di una compagnia devono essere unici
ck_data	vincolo di check per indicare che la data di inizio di una cadenza giornaliera deve essere minore della data fine

VARI ED EVENTUALI All'interno del database sono presenti ulteriori vincoli di not null, ad esempio per il nome e il cognome dei passeggeri, in modo che il nominativo del biglietto sia sempre riempito.

7 Schema logico

CORSA(IdCorsa, Ritardo, Scalo, NomeCompagnia↑, cittaPartenza, cittaArrivo, disponibilità, nomecadenzagiornaliera ↑)
PASSEGGERO(IdPasseggero, Nome,Cognome,DataNascita)
PRENOTAZIONE(IdPrenotazione, IdCorsa↑, IdPasseggero↑, SovrapprezzoPrenotazione, SovrapprezziBagagli, pesobagagli)
BIGLIETTORIDOTTO(CodBigliettoR, Prezzo, Nominativo)
ACQUISTORIDOTTO(CodBigliettoR ↑, IdPasseggero ↑)
BIGLIETTOINTERO(CodBigliettoI, Prezzo, Nominativo)
ACQUISTOINTERO(CodBigliettoI ↑, IdPasseggero ↑)
CADENZAGIORNALIERA(DataInizio, DataFine, GiornoSettimanale, OrarioPartenza, OrarioArrivo, nomeCadenzaGiornaliera)
COMPAGNIADINAVIGAZIONE(NomeCompagnia, NumeroNatanti, Telefono, Mail, SitoWeb)
INDIRIZZOSOCIAL(Indirizzo, NomeCompagnia↑)
NATANTE(codNatante, nomeCompagnia ↑, tipoNatante, capienzapasseggeri, capienzaautomezzi)
NAVIGAZIONE(idCorsa↑, codNatante↑)

8 Descrizione di Trigger e Funzioni individuate

8.1 after_insert_prenotazione

Funzione che, dopo l'inserimento di una tupla in prenotazione, attiva il trigger che permette di aggiungere una tupla corrispondente in bigliettoridotto e in acquistoridotto se l'età è minore di 18, oppure in bigliettointero e acquistointero se l'età è maggiore di 18. Questa funzione inoltre permette di indicare l'eventuale sovrapprezzo della prenotazione o il sovrapprezzo dei bagagli:

```
CREATE OR REPLACE FUNCTION public.after_insert_prenotazione()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
    AS $BODY$
DECLARE
    data_pass DATE;
    rand_numb INTEGER;
    nome_pass passeggero.nome%type;
    cognome_pass passeggero.cognome%type;
    result_string VARCHAR(100);
    age_pass INTEGER;
    disponibilita_corsa INTEGER;
    data_corsa cadenzagiornaliera.datainizio%type;
    tempo_year INTEGER;
    tempo_month INTEGER;
    tempo_day INTEGER;
BEGIN

    SELECT disponibilita INTO disponibilita_corsa
    FROM corsa
    WHERE idcorsa = NEW.idcorsa;
```

```
IF disponibilita\_corsa = 0 THEN
    RAISE EXCEPTION 'I posti per questa corsa sono esauriti.';
ELSE
    SELECT nome, cognome, datanascita
    INTO nome_pass, cognome_pass, data_pass
    FROM passeggero
    WHERE idpasseggero = new.idpasseggero;

    SELECT datainizio INTO data_corsa
    FROM cadenzagiornaliera
    WHERE nomecadenzagiornaliera in (SELECT nomecadenzagiornaliera
                                     FROM corsa
                                     WHERE idcorsa = NEW.idcorsa);

    result_string := concat(nome_pass, ' ', cognome_pass);
    rand_numb := floor(random() * 1000000) :: INTEGER + 1;

    SELECT EXTRACT(YEAR FROM age(current_date, data_pass))
           INTO age_pass;
    SELECT EXTRACT(YEAR FROM age(data_corsa, current_date))
           INTO tempo_year;
    SELECT EXTRACT(MONTH FROM age(data_corsa, current_date))
           INTO tempo_month;
    SELECT EXTRACT(DAY FROM age(data_corsa, current_date))
           INTO tempo_day;

    IF(age_pass < 18) THEN
        IF (tempo_year > 0 OR tempo_month > 0 OR tempo_day > 0) THEN
            INSERT INTO bigliettoridotto
            VALUES (rand_numb, 10.50 + NEW.sovrapprezzoprenotazione +
                    + NEW.sovrapprezzobagagli, result_string);

            INSERT INTO acquistoridotto
            VALUES (rand_numb, new.idpasseggero);

        ELSE
            INSERT INTO bigliettoridotto
            VALUES (rand_numb, 10.50 + NEW.sovrapprezzobagagli,
                    result_string);

            INSERT INTO acquistoridotto
            VALUES (rand_numb, new.idpasseggero);

        END IF;
    ELSE

        IF (tempo_year > 0 OR tempo_month > 0 OR tempo_day > 0) THEN
            INSERT INTO bigliettointero
            VALUES (rand_numb, 15.50 + NEW.sovrapprezzoprenotazione
```

```
                        + NEW.sovrapprezzobagagli, result_string);

                    INSERT INTO acquistointero
                    VALUES (rand_numb, NEW.idpasseggero);
                ELSE
                    INSERT INTO bigliettointero
                    VALUES (rand_numb, 15.50 + NEW.sovrapprezzobagagli,
                            result_string);

                    INSERT INTO acquistointero
                    VALUES (rand_numb, NEW.idpasseggero);

                END IF;
            END IF;
        END IF;
        RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.after_insert_prenotazione()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER after_insert_prenotazione
    AFTER INSERT
    ON public.prenotazione
    FOR EACH ROW
    EXECUTE FUNCTION public.after_insert_prenotazione();
```

8.2 aggiungi_navigazione

Funzione che, in seguito all'inserimento di una corsa all'interno del database, attiva un trigger che permette di inserire automaticamente una tupla in navigazione, recuperando il codice del natante in maniera casuale da un natante già esistente e della stessa compagnia. Se per la corsa che si vuole inserire, non esiste un natante di quella compagnia, si stamperà un messaggio di errore

```
CREATE OR REPLACE FUNCTION public.aggiungi_navigazione()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$

DECLARE
    cod_natante VARCHAR(15);
BEGIN

    SELECT codnatante INTO cod_natante
    FROM natante
    WHERE nomecompagnia = NEW.nomecompagnia
```

```
    ORDER BY random()
    LIMIT 1;

    IF cod_natante is not null THEN
        INSERT INTO navigazione VALUES (NEW.idcorsa, cod_natante);
    ELSE
        RAISE EXCEPTION 'Nessun natante trovato per la compagnia
        di cui si vuole inserire la corsa';
    END IF;

    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.aggiungi_navigazione()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER aggiungi_navigazione
    AFTER INSERT
    ON public.corsa
    FOR EACH ROW
    EXECUTE FUNCTION public.aggiungi_navigazione();
```

8.3 diminuisci_disponibilita

Funzione che attiva un trigger in seguito all'inserimento nella tabella Prenotazione. Questa funzione permette di diminuire la disponibilità dei posti nella corsa.

```
CREATE OR REPLACE FUNCTION public.diminuisci_disponibilita()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
    AS $BODY$
BEGIN

    UPDATE corsa
    SET disponibilita = disponibilita - 1
    WHERE idcorsa = new.idcorsa;

    RETURN NEW;

END;
$BODY$;

ALTER FUNCTION public.diminuisci_disponibilita()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER diminuisci_disponibilita
    AFTER INSERT
```

```
    ON public.prenotazione
    FOR EACH ROW
    EXECUTE FUNCTION public.diminuisci_disponibilita();
```

8.4 elimina_prenotazione

Funzione che attiva un trigger in seguito all'eliminazione di una prenotazione all'interno del database. Questo trigger permette di conseguenza, di eliminare anche le tuple corrispondenti alla prenotazione nelle tabelle acquisto e biglietto (intero o ridotto).

```
CREATE OR REPLACE FUNCTION public.elimina_prenotazione()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
    cod_bigl_r bigliettoridotto.codbigliettor%TYPE;
    cod_bigl_i bigliettointero.codbiglietto i%TYPE;
    data_pass DATE;
    age_pass INTEGER;
BEGIN

    SELECT codbigliettor INTO cod_bigl_r
    FROM acquistoridotto
    WHERE idpasseggero = OLD.idpasseggero;

    SELECT codbiglietto i INTO cod_bigl_i
    FROM acquistointero
    WHERE idpasseggero = OLD.idpasseggero;

    SELECT datanascita INTO data_pass
    FROM passeggero
    WHERE idpasseggero = OLD.idpasseggero;

    SELECT EXTRACT(YEAR FROM age(current_date, data_pass))
        INTO age_pass;

    IF (age_pass < 18) THEN

        DELETE FROM acquistoridotto
        WHERE idpasseggero = OLD.idpasseggero;
        DELETE FROM bigliettoridotto
        WHERE codbigliettor = cod_bigl_r;

    ELSE

        DELETE FROM acquistointero
        WHERE idpasseggero = OLD.idpasseggero;
        DELETE FROM bigliettointero
```

```
        WHERE codbigliettoi = cod_bigl_i;

    END IF;

    UPDATE corsa
    SET disponibilita = disponibilita + 1
    WHERE idcorsa = OLD.idcorsa;

    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.elimina_prenotazione()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER elimina_prenotazione
    AFTER DELETE
    ON public.prenotazione
    FOR EACH ROW
    EXECUTE FUNCTION public.elimina_prenotazione();
```

8.5 imposta_disponibilita

Dopo l'inserimento in corsa viene attivato un trigger che permette di impostare la disponibilita della corsa alla capienza del natante corrispondente per quella corsa. Se il natante in questione è un traghetto, la disponibilità viene calcolata sommando la capienza passeggeri alla capienza automezzi, altrimenti viene considerata solo la capienza passeggeri.

```
CREATE OR REPLACE FUNCTION public.imposta_disponibilita()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
DECLARE
    capienzap INTEGER;
    capienzaa INTEGER;
    tipo_natante VARCHAR(50);

    SELECT capienzapasseggeri, tiponatante INTO capienzap, tipo_natante
    FROM natante
    WHERE codnatante IN (SELECT codnatante
                         FROM navigazione
                         WHERE idcorsa = new.idcorsa);

    SELECT capienzaautomezzi INTO capienzaa
    FROM natante
    WHERE codnatante IN (SELECT codnatante
                         FROM navigazione
```

```
                        WHERE idcorsa = new.idcorsa);

    IF tipo_natante = 'traghetto' THEN

        UPDATE corsa
        SET disponibilita = capienzap + capinezaa
        WHERE idcorsa = new.idcorsa;

    ELSE

        UPDATE corsa
        SET disponibilita = capienzap
        WHERE idcorsa = new.idcorsa;

    END IF;

    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.imposta_disponibilita()
OWNER TO postgres;

CREATE OR REPLACE TRIGGER imposta_disponibilita
    AFTER INSERT
    ON public.corsa
    FOR EACH ROW
    EXECUTE FUNCTION public.imposta_disponibilita();
```

8.6 incrementa_id_passeggero

Funzione che incrementa l'id del passeggero in maniera automatica. Viene infatti creata una sequenza che parte da 1 e ad ogni inserimento di un passeggero all'interno del database, il suo identificativo viene incrementato.

```
CREATE SEQUENCE IF NOT EXISTS public.sequenza_id_passeggero
    INCREMENT 1
    START 1
    MINVALUE 1
    MAXVALUE 9223372036854775807
    CACHE 1;

ALTER SEQUENCE public.sequenza_id_passeggero
    OWNER TO postgres;


CREATE OR REPLACE FUNCTION public.incrementa_id_passeggero()
    RETURNS trigger
    LANGUAGE 'plpgsql'
```

```
    COST 100
    VOLATILE NOT LEAKPROOF
    AS $BODY$

BEGIN

    NEW.idpasseggero = nextval('sequenza_id_passeggero');
    RETURN NEW;

END;
$BODY$;

ALTER FUNCTION public.incrementa_id_passeggero()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER incrementa_id_passeggero
    BEFORE INSERT
    ON public.passeggero
    FOR EACH ROW
    EXECUTE FUNCTION public.incrementa_id_passeggero();
```

8.7 incrementa_numero_natanti

Funzione che, in seguito all'inserimento di un natante di una determinata compagnia esistente nel database, incrementa il numero di natanti per quella compagnia.

```
CREATE OR REPLACE FUNCTION public.incrementa_numero_natanti()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
BEGIN

    UPDATE compagniadinavigazione
    SET numeronatanti = numeronatanti + 1
    WHERE nomecompagnia = new.nomecompagnia;

    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.incrementa_numero_natanti()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER incrementa_numero_natanti
    AFTER INSERT
    ON public.natante
    FOR EACH ROW
```

```
    EXECUTE FUNCTION public.incrementa_numero_natanti();
```

8.8 modifica_ritardo

Questa funzione permette la modifica di un ritardo nella tabella corsa. Il ritardo può essere modificato indicando i minuti di ritardo della corsa, oppure in 'canc' se si desidera cancellare la corsa. In quest'ultimo caso, viene anche impostata la disponibilita della corsa a 0 in modo che non sia possibile effettuare una prenotazione per quella corsa.

```
CREATE OR REPLACE FUNCTION public.modifica_ritardo()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
    AS $BODY$
BEGIN

    IF NEW.ritardo IS DISTINCT FROM OLD.ritardo THEN

        IF NEW.ritardo IS NOT NULL AND NEW.ritardo != 'canc' THEN

            UPDATE corsa
            SET ritardo = NEW.ritardo
            WHERE idcorsa = NEW.idcorsa;

        ELSE

            UPDATE corsa
            SET ritardo = 'canc'
            WHERE idcorsa = NEW.idcorsa;

            UPDATE corsa
            SET disponibilita = 0
            WHERE idcorsa = NEW.idcorsa;

        END IF;

    END IF;

    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.modifica_ritardo()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER modifica_ritardo
    AFTER UPDATE OF ritardo
    ON public.corsa
```

```
    FOR EACH ROW
    EXECUTE FUNCTION public.modifica_ritardo();
```

8.9 prezzo_bagaglio

Funzione che, prima dell'inserimento in prenotazione, setta il sovrapprezzo dei bagagli sulla base del peso indicato.

```
CREATE OR REPLACE FUNCTION public.prezzo_bagaglio()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
    AS $BODY$

BEGIN

    IF NEW.peso_bagaglio <= 5 THEN
        NEW.sovrapprezzobagagli = 0.0;
    ELSIF NEW.peso_bagaglio > 5 AND NEW.peso_bagaglio <= 50 THEN
        NEW.sovrapprezzobagagli = 10.0;
    ELSIF NEW.peso_bagaglio > 50 THEN
        NEW.sovrapprezzobagagli = 15.0;
    END IF;

    RETURN NEW;

END;
$BODY$;

ALTER FUNCTION public.prezzo_bagaglio()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER prezzo_bagaglio
    BEFORE INSERT
    ON public.prenotazione
    FOR EACH ROW
    EXECUTE FUNCTION public.prezzo_bagaglio();
```

8.10 setta_sovrapprezzoprenotazione

Funzione che setta il sovrapprezzo della prenotazione prima dell'inserimento di una prenotazione Il sovrapprezzo si basa sulla data in cui avviene la prenotazione. Se infatti viene effettuata prima del periodo di attivazione della corsa, viene sommato al prezzo del biglietto, un sovrapprezzo standard di 3,00 euro. Altrimenti il sovrapprezzo viene impostato a 0 e il costo del biglietto resta invariato.

```
CREATE OR REPLACE FUNCTION public.setta_sovrapprezzoprenotazione()
    RETURNS trigger
    LANGUAGE 'plpgsql'
```

```
    COST 100
    VOLATILE NOT LEAKPROOF
    AS $BODY$
DECLARE
    data_corsa DATE;
    tempo_year INTEGER;
    tempo_month INTEGER;
    tempo_day INTEGER;
BEGIN

    SELECT datainizio INTO data_corsa
    FROM cadenzagiornaliera
    WHERE nomecadenzagiornaliera in (SELECT nomecadenzagiornaliera
                                     FROM corsa
                                     WHERE idcorsa = new.idcorsa);

    SELECT EXTRACT(YEAR FROM age(data_corsa, current_date))
        INTO tempo_year;
    SELECT EXTRACT(MONTH FROM age(data_corsa, current_date))
        INTO tempo_month;
    SELECT EXTRACT(DAY FROM age(data_corsa, current_date))
        INTO tempo_day;

    IF (tempo_year > 0 OR tempo_month > 0 OR tempo_day > 0) THEN
        NEW.sovrapprezzoprenotazione = 3.00;
    ELSE
        NEW.sovrapprezzoprenotazione = 0;
    END IF;

    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.setta_sovrapprezzoprenotazione()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER setta_sovrapprezzoprenotazione
    BEFORE INSERT
    ON public.prenotazione
    FOR EACH ROW
    EXECUTE FUNCTION public.setta_sovrapprezzoprenotazione();
```

8.11 verifica_disponibilita_auto

Questa funzione permette, attraverso l'inserimento di una prenotazione, e in particolare attraverso l'inserimento nell'attributo auto di un valore, di verificare se sia possibile prenotare un posto auto nella corsa verificando se il natante della corsa è effettivamente un traghetto. Altrimenti viene lanciata un'eccezione per indicare che non è possibile prenotare un posto auto.

```
CREATE OR REPLACE FUNCTION public.verifica_disponibilita_auto()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
    AS $BODY$
DECLARE
    tipo natante.tiponatante%TYPE;
BEGIN
    SELECT tiponatante INTO tipo
    FROM natante
    WHERE codnatante IN (SELECT codnatante
                         FROM navigazione
                         WHERE idcorsa = NEW.idcorsa);

    IF tipo = 'traghetto' AND NEW.auto = true THEN
        UPDATE prenotazione
        SET auto = true
        WHERE idcorsa = NEW.idcorsa;

        UPDATE corsa
        SET disponibilita = disponibilita - 1
        WHERE idcorsa = NEW.idcorsa;
    ELSE
        RAISE EXCEPTION 'Impossibile aggiungere l'auto, perch  l'imbarcazione n
    END IF;

    RETURN NEW;
END;
$BODY$;

ALTER FUNCTION public.verifica_disponibilita_auto()
    OWNER TO postgres;

CREATE OR REPLACE TRIGGER verifica_disponibilita_auto
    AFTER INSERT
    ON public.prenotazione
    FOR EACH ROW
    EXECUTE FUNCTION public.verifica_disponibilita_auto();
```