SAPIENZA
UNIVERSITÀ DI ROMA

Basi di Dati, Modulo 2

Sapienza Università di Roma
Facoltà di Ing. dell'Informazione, Informatica e Statistica
Laurea in Informatica
Prof. Toni Mancini
http://tmancini.di.uniroma1.it

Progetto 20050704 (P.20050704)

RainAir

Versione 2020-11-01

Indice

1 Introduzione

Si vuole progettare e realizzare un sistema per la gestione prenotazioni della *RainAir*, una nuova compagnia aerea a basso costo. Il sistema permetterà di gestire le prenotazioni di voli aerei, e anche prenotazioni di stanze in hotel convenzionati.

2 Specifica dei Requisiti

Per l'applicazione sono di interesse i clienti (con nome, cognome ed indirizzo di residenza), i voli aerei, tutti giornalieri, e le prenotazioni fatte dai clienti. In particolare, dei voli interessa codice (una stringa), numero di miglia percorse, orari ed aeroporti di partenza ed arrivo (ad es., il volo "RA 1234" parte ogni giorno alle 13:50 dall'aeroporto di Roma Ciampino e arriva a Londra Stansted alle 15:50, percorrendo 1000 miglia), ed il velivolo usato, mentre delle prenotazioni è importante conoscere il momento in cui vengono effettuate.

Degli aeroporti da cui partono e arrivano voli bisogna rappresentare il codice (una stringa di 3 caratteri, ad es. "CIA") il nome, la città e lo stato dove è collocato, e l'ammontare delle tasse di decollo e atterraggio.

Dei velivoli è di interesse conoscere codice (una stringa), tipo, numero di posti e costo al miglio (ovvero quanto il velivolo costa alla compagnia per ogni miglio di volo).

Dei biglietti di un volo è di interesse conoscere il loro "prezzo base", base di partenza per il calcolo del prezzo finale. Il prezzo base dipende dal costo che la compagnia deve sostenere per effettuare il volo. Quest'ultimo è dato da due addendi: (i) il prodotto tra il numero di miglia effettuate e il costo al miglio del velivolo in uso e (ii) la somma delle tasse di decollo e atterraggio negli aeroporti di partenza e arrivo, rispettivamente. Il prezzo base di un biglietto è dato dal costo su descritto diviso per il numero di posti del velivolo, e incrementato del 20% (che è il ricarico che la compagnia vuole applicare).

Una prenotazione da parte di un cliente può essere relativa anche a più voli (almeno uno). Di ogni volo prenotato, interessa la data in cui il cliente vuole volare, ed il numero di posti richiesti.

Una prenotazione può, ma non obbligatoriamente, essere relativa anche ad un hotel, nel qual caso interessa conoscere la data di check-in, quella di check-out, e il numero di stanze prenotate (si ignorino, per semplicità, le diverse tipologie di stanze). Di un hotel sono di interesse il nome, l'indirizzo, la categoria (ovvero il numero di stelle, da una a cinque), la tariffa per stanza per notte, la città dove è collocato, la distanza dal centro

cittadino, e la possibile informazione sull'aeroporto più vicino, con relativa distanza (le distanze siano tutte in Km).

Alcuni clienti sono "frequent flyers". Di questi interessa conoscere il codice, la data di affiliazione, e il numero di miglia che hanno accumulato fino ad un certo momento. Il numero di miglia accumulate fino al momento richiesto è la somma delle miglia guadagnate con le prenotazioni effettuate a suo nome per voli prenotati dopo la data di affiliazione, ed effettuati fino a quel momento. In particolare, il guadagno di miglia relative ad un singolo volo è pari al numero di miglia percorse dal volo per il numero di posti prenotati. Se una prenotazione è relativa a più voli, le miglia guadagnate sono la somma di quelle relative ad ogni singolo volo.

Inoltre, sono previsti benefici per le prenotazioni che includono anche hotel (indipendentemente dalle date di check-in e check-out). In particolare, le miglia relative ad una prenotazione raddoppiano se questa prevede anche un hotel fino a 4 stelle, e triplicano se l'hotel prenotato è a 5 stelle.

Il Sistema prenotazioni (un sistema esterno) necessita di calcolare il numero di posti disponibili all'istante corrente su un dato volo di una certa data. Il numero di posti disponibili è calcolato a partire dal numero di posti del velivolo che effettua il volo in questione, diminuito del numero di posti già prenotati (all'istante corrente).

Il Sistema prenotazioni deve anche poter calcolare il prezzo complessivo, all'istante corrente, di un certo numero di biglietti per un volo di un dato giorno. Tale prezzo è da considerarsi valido solo nel caso in cui il volo disponga, al momento corrente, di un numero sufficiente di posti per la data richiesta, e si calcola moltiplicando il prezzo di un biglietto singolo per il numero di biglietti richiesti.

Il prezzo di un singolo biglietto è altamente flessibile, ed è composto da diverse componenti, dovute a diversi fattori:

1. Il prezzo base, che dipende dal volo;

2. Il numero di posti disponibili al momento corrente per la data di volo richiesta.

Il calcolo del prezzo del biglietto parte dal suo prezzo base, e subisce poi delle modifiche a seconda della disponibilità attuale di posti sulla data di volo richiesta. In particolare, al prezzo base si applica la seguente regola: se il numero di posti disponibili al momento della prenotazione per il volo in questione è maggiore della metà dei posti totali (ovvero quelli del velivolo che effettua il volo), si applica uno sconto del 2% per ogni posto libero oltre la metà. Al contrario, se il numero di posti disponibili è minore della metà, si applica un sovrapprezzo del 2% in modo del tutto analogo. Ad esempio, se il velivolo che effettua il volo ha 100 posti in totale, e, al momento della prenotazione, ne sono ancora liberi 53 per la data di volo richiesta, si applicherà uno sconto del 2% al prezzo base per tre volte (cosa diversa dall'applicare il 6% di sconto), mentre, se i posti liberi fossero 47, si applicherebbe un sovrapprezzo del 2% per tre volte.

Il Sistema prenotazioni vuole offrire anche il seguente servizio: data una città e una tariffa massima, vuole suggerire un insieme di hotel in quella città che abbiano tutti una

tariffa al più pari a quella indicata. La scelta degli hotel avviene secondo le seguenti regole (a parte quella sulla tariffa, che deve essere sempre rispettata):

- Farà parte del risultato l'hotel più vicino al centro della città in questione con tariffa entro la soglia; sia questo hotel A.
- Farà inoltre parte del risultato qualunque altro hotel con lo stesso numero di stelle di A che abbia una distanza dal centro pari al più il 110% di quella di A.
- Infine, faranno parte del risultato anche quegli hotel che hanno più stelle di A, ma più lontani di A dal centro. In particolare, quelli per cui la distanza dal centro sia al massimo il 120% di quella di A.

Parte A

Analisi Concettuale

A.1 Raffinamento dei requisiti

A.1.1 Testo

Raffinare la specifica dei requisiti eliminando inconsistenze, omissioni o ridondanze producendo un elenco numerato di requisiti il meno ambiguo possibile.

A.1.2 Soluzione

1. Dati di interesse sui clienti:
   1.1. nome
   1.2. cognome
   1.3. indirizzo di residenza
   1.4. se è "frequent flyer". In questo caso, è di interesse:
      1.4.1. codice
      1.4.2. data affiliazione.
2. I voli sono tutti giornalieri.
3. Dati di interesse sui voli:
   3.1. codice (una stringa)
   3.2. numero miglia (un intero positivo)
   3.3. orario di partenza
   3.4. aeroporto di partenza
   3.5. orario di arrivo
   3.6. aeroporto di arrivo
   3.7. velivolo (Req. 7.).
4. Dati di interesse sugli aeroporti:
   4.1. nome
   4.2. codice (stringa di 3 caratteri, ad es. "CIA")
   4.3. città
   4.4. tasse di decollo
   4.5. tasse di atterraggio.
5. Dati di interesse sulle prenotazioni
   5.1. voli prenotati (almeno uno); di ognuno interessa:
      5.1.1. data
      5.1.2. numero di posti
   5.2. l'eventuale hotel prenotato, nel qual caso interessa:
      5.2.1. hotel
      5.2.2. data di check-in
      5.2.3. data di check-out

5.2.4. numero di stanze

5.3. istante

5.4. cliente.

6. Dati di interesse sulle città:

6.1. nome

6.2. stato

7. Dati di interesse sui velivoli:

7.1. codice (stringa)

7.2. tipo (stringa)

7.3. capacità

7.4. costo al miglio.

8. Dati di interesse sugli hotel:

8.1. nome

8.2. indirizzo

8.3. categoria (un intero tra 1 e 5)

8.4. tariffa per notte

8.5. città

8.6. distanza dal centro città

8.7. l'eventuale aeroporto più vicino, con relativa distanza (in Km).

9. Le funzionalità offerte dal sistema sono:

9.1. Il Sistema Prenotazioni (un sistema esterno) deve poter calcolare il prezzo base di un biglietto. Detto m il numero di miglia effettuate, c il costo al miglio del velivolo in uso, d e a le tasse di decollo e di atterraggio neigli aeroporti di partenza e arrivo, rispettivamente, e p il numero di posti del velivolo, il prezzo base è

$$1.2\left(\frac{mc + d + a}{p}\right);$$

9.2. Il Sistema Prenotazioni, dati un intero n e una data d, deve poter calcolare il prezzo complessivo, all'istante corrente, di n biglietti per un volo che verrà effettuato nella data d. Se il volo non dispone, al momento corrente, di almeno n posti nella data d, il sistema deve restituire un errore.

Il prezzo complessivo è calcolato moltiplicando il costo di un biglietto singolo per n.

Il prezzo di un biglietto singolo è ottenuto come segue, a partire dal suo prezzo base b e dal numero di posti disponibili al momento della prenotazione:

- se il numero di posti disponibili al momento della prenotazione per il volo in questione è esattamente la metà dei posti totali, allora il prezzo di un singolo biglietto è dato dal suo prezzo base;
- se il numero di posti disponibili al momento della prenotazione per il volo in questione è maggiore della metà dei posti totali, allora il prezzo di un singolo biglietto è calcolato applicando uno sconto (composto) del 2% al suo prezzo base per ogni posto libero oltre la metà.
- altrimenti (ovvero, se il numero di posti disponibili è minore della metà dei posti totali), allora il prezzo di un singolo biglietto è calcolato applicando un sovrapprezzo (composto) del 2% al suo prezzo base per ogni posto occupato oltre la metà.

9.3. Il Sistema Prenotazioni deve poter calcolare il numero di miglia accumulate da un frequent flyer f (la cui data di affiliazione è a) fino a un dato istante i. Tale numero è calcolato come la somma delle miglia guadagnate con prenotazioni effettuate a nome di f nel periodo compreso tra a e i. Il numero di miglia guadagnate per un singolo volo è pari al numero di miglia percorse dal volo per il numero di posti prenotati. Se una prenotazione è relativa a più voli, le miglia guadagnate sono la somma di quelle relative ad ogni singolo volo. Se la prenotazione prevede anche un hotel, le miglie relative raddoppiano se l'hotel ha fino a 4 stelle, mentre triplicano se l'hotel è a 5 stelle.

9.4. Il Sistema Prenotazioni deve poter calcolare il numero di posti disponibili all'istante corrente su un dato volo v in una data d. Tale numero è pari al numero di posti del velivolo w che effettua v meno il numero di posti già prenotati (all'istante corrente);

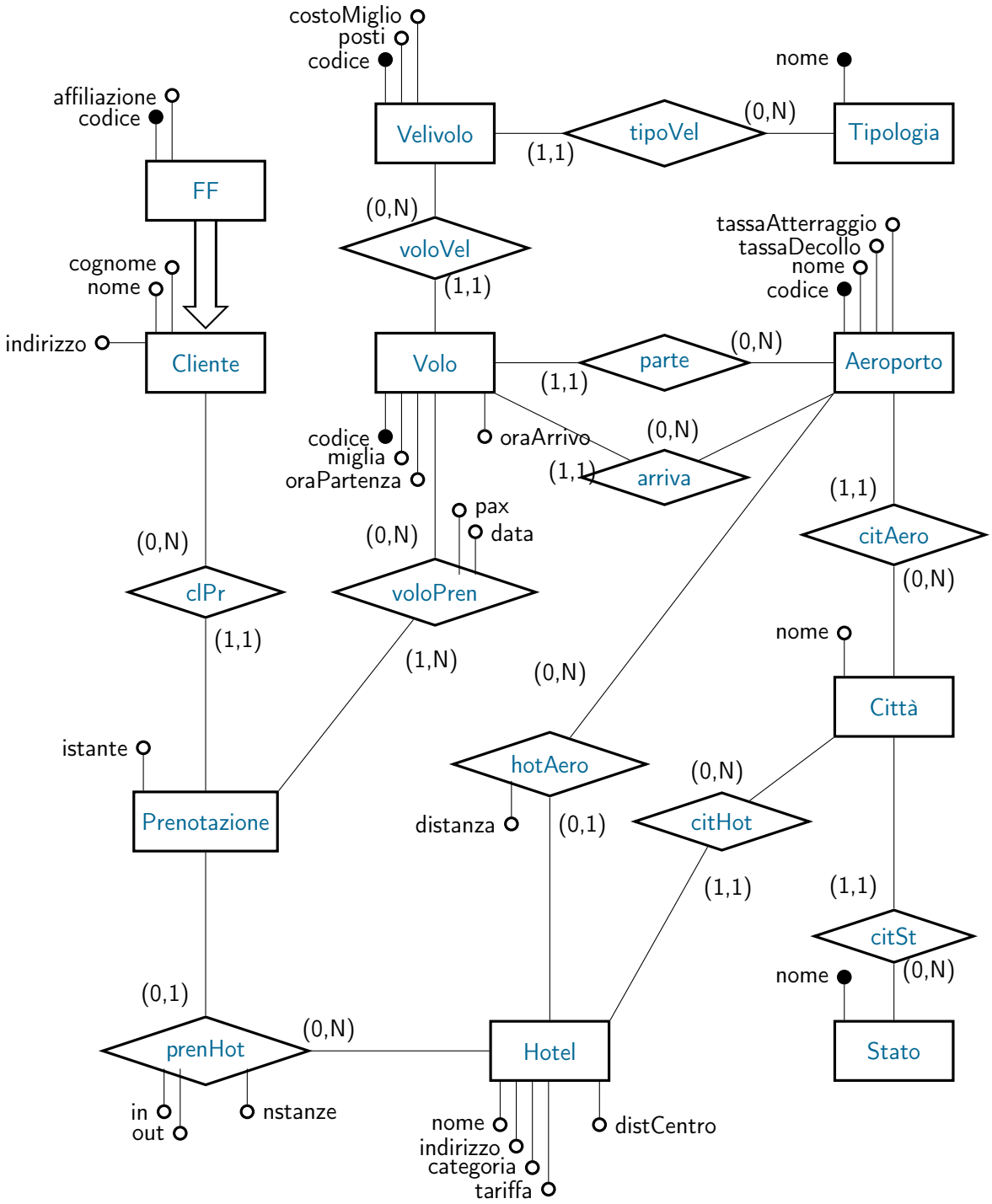
9.5. Il Sistema Prenotazioni, data una città c e una tariffa massima t, deve ottenere l'insieme H di hotel in c la cui tariffa sia minore o uguale a t. H deve contenere A, l'hotel più vicino al centro della città c con tariffa entro la soglia. Inoltre, H conterrà ogni altro hotel B con lo stesso numero di stelle di A che abbia una distanza dal centro minore o uguale al 110% di quella di A. Infine, H conterrà ogni altro hotel D che ha più stelle di A, ma la cui distanza dal centro sia minore o uguale al 120% di quella di A.

A.2 Diagramma ER e specifiche dei dati

A.2.1 Testo

Proseguire la fase di Analisi Concettuale dei requisiti a partire dall'output della fase di raffinamento effettuata al passo A.1. In particolare, produrre il diagramma ER concettuale per l'applicazione e le specifiche dei dati (dizionario dei dati e vincoli esterni definiti usando il linguaggio della logica del primo ordine).

A.2.2 Soluzione



Specifiche dei Dati

Entità Cliente

Ogni istanza di questa entità rappresenta un cliente (Req. 1.)

attributo	dominio	molteplicità	descrizione
nome	stringa		Il nome del cliente
cognome	stringa		Il cognome del cliente
indirizzo	Indirizzo		L'indirizzo del cliente

Entità FF

Ogni istanza di questa entità rappresenta un cliente "frequent flyer" (Req. 1.4.)

attributo	dominio	molteplicità	descrizione
codice	stringa		Il codice del frequent flyer
affiliazione	data		La data di affiliazione del frequent flyer

Entità Aeroporto

Ogni istanza di questa entità rappresenta un aeroporto (Req. 4.)

attributo	dominio	molteplicità	descrizione
codice	stringa di 3 caratteri		Il codice dell'aeroporto
nome	stringa		Il nome dell'aeroporto
tassaDecollo	Denaro		L'importo delle tasse di decollo dell'aeroporto
tassaAtterraggio	Denaro		L'importo delle tasse di atterraggio dell'aeroporto

Entità Volo

Ogni istanza di questa entità rappresenta un volo (Req. 3.)

attributo	dominio	molteplicità	descrizione
codice	stringa		Il codice del volo
miglia	intero > 0		Il numero di miglia percorse dal volo
oraPartenza	ora		L'orario di partenza del volo
oraArrivo	ora		L'orario di arrivo del volo

Vincoli:

[V.Volo.aeroportiDiversi] Gli aeroporti di partenza e di arrivo di un volo devono essere diversi.

Formalmente:

$$\forall v,ap,aa\ \text{Volo}(v) \wedge \text{parte}(v,ap) \wedge \text{arrivo}(v,aa) \rightarrow ap \neq aa$$

[V.Volo.maiOverbooking] Il numero di posti prenotati non deve mai superare la capacità del velivolo che effettua il volo. Ciò equivale a dire che il numero di posti disponibili per un volo in una certa data deve essere sempre maggiore o uguale a zero.

Formalmente:

$$\forall v,d,i\ \text{Volo}(v) \wedge \text{data}(d) \wedge \text{dataora}(i) \rightarrow \text{Posti.postiDisponibili}(v,d,i) \geq 0$$

dove Posti.postiDisponibili è una operazione dello Use-Case Posti.

Entità Velivolo

Ogni istanza di questa entità rappresenta un velivolo (Req. 7.)

attributo	dominio	molteplicità	descrizione
codice	stringa		Il codice del velivolo
posti	intero > 0		Il numero di posti del velivolo
costoMiglio	Denaro		Il costo al miglio del velivolo

Entità Tipologia
Ogni istanza di questa entità rappresenta una tipologia di velivolo (Req. 7.2.)

attributo	dominio	molteplicità	descrizione
nome	stringa		Il nome della tipologia di velivolo

Entità Città
Ogni istanza di questa entità rappresenta una città (Req. 6.)

attributo	dominio	molteplicità	descrizione
nome	stringa		Il nome della città

Entità Stato
Ogni istanza di questa entità rappresenta uno stato (Req. 6.2.)

attributo	dominio	molteplicità	descrizione
nome	stringa		Il nome dello stato

Entità Hotel
Ogni istanza di questa entità rappresenta un hotel (Req. 8.)

attributo	dominio	molteplicità	descrizione
nome	stringa		Il nome dell'hotel
indirizzo	Indirizzo		L'indirizzo dell'hotel
categoria	$[1, 5]$		La categoria dell'hotel
tariffa	Denaro		La tariffa dell'hotel
distCentro	Denaro		La tariffa dell'hotel

Entità Prenotazione
Ogni istanza di questa entità rappresenta una prenotazione (Req. 5.)

attributo	dominio	molteplicità	descrizione
istante	dataora		L'istante della prenotazione

Vincoli:

[V.Prenotazione.date] Le prenotazioni devono essere effettuate prima dell'istante di partenza di ogni volo relativo.

Formalmente:

$$\begin{aligned}&\forall p,i,v,dv,ov,iv\\&\quad \text{Prenotazione}(p) \wedge \text{istante}(p,i) \wedge \text{voloPren}(v,p)\\&\quad \text{data}(v,p,dv) \wedge \text{oraPart}(v,ov) \wedge \text{dataora}(iv)\\&\quad \text{data}(iv,dv) \wedge \text{ora}(iv,ov)\\&\qquad \rightarrow\ iv > i\end{aligned}$$

[V.Prenotazione.checkin] L'istante di prenotazione per un Hotel deve essere antecedente alla data di check-in.

Formalmente:

$$\begin{aligned}&\forall\ p,i,ci,id,h\\&\quad \text{Prenotazione}(p) \wedge \text{Hotel}(h) \wedge \text{prenHot}(p,h) \wedge \text{istante}(p,i) \wedge \text{in}(p,h,ci)\\&\quad \text{data}(i,id)\ \rightarrow\ id \leq ci\end{aligned}$$

Relationship tipoVel

Ogni istanza di questa relationship lega un Velivolo a una Tipologia di velivolo
Attributi: Nessuno

Relationship prenHot

Ogni istanza di questa relationship lega una Pranotazione a un Hotel

attributo	dominio	molteplicità	descrizione
in	data		La data di check-in
out	data		La data di check-out
nstanze	intero > 0		Il numero di stanze prenotate

Vincoli:

[V.prenHot.date] La data di check-in per ogni prenotazione di Hotel deve essere antecedente alla data di check-out.

Formalmente:

$$\begin{aligned}&\forall p,i,o,h\\&\quad \text{Prenotazione}(p) \wedge \text{Hotel}(h) \wedge \text{prenHot}(p,h) \wedge\\&\quad \text{in}(p,h,i) \wedge \text{out}(p,h,o)\\&\qquad \rightarrow\ i < o\end{aligned}$$

Relationship voloPren
Ogni istanza di questa relationship lega un Volo a una Prenotazione

attributo	dominio	molteplicità	descrizione
data	data		La data del Volo prenotato
pax	intero > 0		Il numero di posti prenotati

Relationship parte
Ogni istanza di questa relationship lega un Volo all'Aeroporto di partenza
Attributi: Nessuno

Relationship arriva
Ogni istanza di questa relationship lega un Volo all'Aeroporto di arrivo
Attributi: Nessuno

Relationship citSt
Ogni istanza di questa relationship lega una Città allo Stato in cui si trova
Attributi: Nessuno

Relationship citHot
Ogni istanza di questa relationship lega un Hotel alla Città in cui si trova
Attributi: Nessuno

Relationship citAero
Ogni istanza di questa relationship lega un Aeroporto alla Città in cui si trova
Attributi: Nessuno

Relationship voloVel
Ogni istanza di questa relationship lega un Volo al Velivolo che lo effettua
Attributi: Nessuno

Relationship clPr
Ogni istanza di questa relationship lega un Cliente a una sua Prenotazione
Attributi: Nessuno

Relationship hotAero

Ogni istanza di questa relationship lega un Hotel all'Aeroporto più vicino

attributo	dominio	molteplicità	descrizione
distanza	intero > 0		La distanza tra l'Hotel e l'Aeroporto

Dominio Indirizzo

Il dominio è un record composto dai seguenti campi:

- via: stringa
- civico: intero > 0 (0,1)
- CAP: stringa di 5 cifre

Dominio Denaro

Il dominio è definito come un reale ≥ 0.

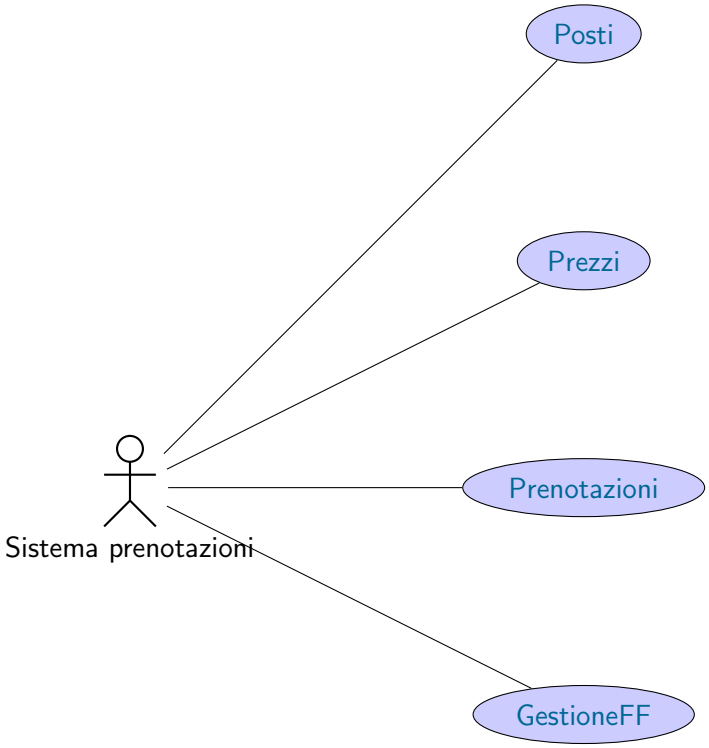
Nota: A differenza di altri progetti, abbiamo per semplicità evitato di gestire valute multiple (e la loro aritmetica) e implicitamente assunto che tutti gli importi siano espressi in una unica valuta di riferimento, ad es., l'Euro.

A.3 Diagramma UML degli Use-Case e Specifiche Concettuali degli Use-Case

A.3.1 Testo

Proseguire la fase di Analisi Concettuale dei requisiti producendo un diagramma UML degli use-case e le specifiche concettuali di ogni use-case.

A.3.2 Soluzione



Specifica Use-Case Posti

- **postiDisponibili(v : Volo, d : data, i: dataora) : intero (Req. 9.4.)**

  precondizioni: nessuna

  postcondizioni:

  Modifica del livello estensionale dei dati: Il livello estensionale dei dati al termine dell'esecuzione della funzione differisce da quello di partenza come segue:

  Variazioni nel dominio di interpretazione: nessuna

  Variazioni nelle ennuple di predicati:

  Valore di ritorno: Sia Z l'insieme delle prenotazioni (pren) effettuate per il volo v del giorno d prima dell'istante i, con l'associato numero di posti (p):

$$Z = \left\{ \text{pren}, \text{p} \;\middle|\; \begin{array}{l} \text{Prenotazione}(\text{pren}) \wedge \text{voloPren}(\text{v}, \text{pren}) \wedge \text{pax}(\text{v}, \text{pren}, \text{p}) \wedge \\ \text{data}(\text{v}, \text{pren}, \text{d}) \wedge \\ \exists\ \text{ip} \;\; \text{istante}(\text{pren}, \text{ip}) \wedge \text{ip} < \text{i} \end{array} \right\}$$

  Sia cap la capacità del velivolo usato per il volo v, ovvero tale da soddisfare la seguente formula (l'assegnamento a cap è unico per costruzione):

$$\exists\ \text{vel} \;\; \text{voloVel}(\text{v}, \text{vel}) \wedge \text{pax}(\text{vel}, \text{cap}).$$

  Si ha:

$$\text{result} = \left(\text{cap} - \left(\sum_{(\text{pren},\text{p}) \in Z} \text{p} \right) \right)$$

Specifica Use-Case Prezzi

- **prezzoBase(v : Volo) : Denaro (Req. 9.1.)**

  precondizioni: nessuna

  postcondizioni:

  Modifica del livello estensionale dei dati: nessuna

  Valore di ritorno: Siano m, cm, cap, td, ta (rispettivamente: numero di miglia, costo al miglio, capacità del velivolo, tasse di decollo e tasse di atterraggio relative al volo v) tali da soddisfare:

  $$\begin{aligned}&\text{miglia}(v, m) \wedge \exists\ \text{vel}\ \ \text{voloVel}(v, \text{vel}) \wedge \text{costoMiglio}(\text{vel}, cm)\\ &\text{pax}(\text{vel}, \text{cap}) \wedge \exists\ \text{ap}\ \ \text{parte}(v, \text{ap}) \wedge \text{tasseDec}(\text{ap}, td) \wedge\\ &\exists\ \text{aa}\ \ \text{arriva}(v, \text{aa}) \wedge \text{tasseDec}(\text{aa}, ta).\end{aligned}$$

  $$\text{result} = \left(\frac{m \times cm + td + ta}{\text{cap}}\right) \times (1 + \text{RICARICO})$$

  dove RICARICO è un simbolo di costante (che, in fase di progettazione, diventerà un parametro di configurazione del sistema, inizialmente assegnato al valore 0.2, come da requisiti).

- **prezzoComplessivoSingoloPosto(v : Volo, d : data) : Denaro**

  precondizioni: $\text{Posti.postiDisponibili}(v, d, \text{adesso}) > 0$.

  postcondizioni:

  Modifica del livello estensionale dei dati: nessuna

  Valore di ritorno: Sia $pd = \text{Posti.postiDisponibili}(v, d, \text{adesso})$.
  Sia inoltre cap tale da soddisfare la formula:

  $$\exists\ \text{vel}\ \ \text{Velivolo}(\text{vel}) \wedge \text{voloVel}(v, \text{vel}) \wedge \text{posti}(\text{vel}, \text{cap})$$

  Sia $\text{diff} = \left(\left\lfloor \frac{\text{cap}}{2} \right\rfloor - pd\right)$.
  Result deve soddisfare la seguente formula:

  $$\begin{aligned}&\text{diff} \leq 0 \ \rightarrow\ \text{result} = \text{Prezzi.prezzoBase}(v) \times (1 - \text{SCONTO})^{-\text{diff}}\\ &\qquad\wedge\\ &\text{diff} > 0 \ \rightarrow\ \text{result} = \text{Prezzi.prezzoBase}(v) \times (1 + \text{SOVRAPPREZZO})^{\text{diff}}\end{aligned}$$

  dove SCONTO e SOVRAPPREZZO sono simboli di costante (che, in fase di progettazione, diventeranno parametri di configurazione del sistema, inizialmente entrambi assegnati al valore 0.02, come da requisiti).

- **prezzoComplessivo(v : Volo, d : data, n:intero > 0) : Denaro (Req. 9.2.)**

  precondizioni: Posti.postiDisponibili(v, d, adesso) $\geq$ n.

  postcondizioni:

  Modifica del livello estensionale dei dati: nessuna

  Valore di ritorno: result = n$\times$Prezzi.prezzoComplessivoSingoloPosto(v, d).

Specifica Use-Case Prenotazioni

- **suggerisciHotel(c : Città, t : Denaro) : Hotel (0,N) (Req. 9.5.)**

  precondizioni: nessuna

  postcondizioni:

  Modifica del livello estensionale dei dati: Il livello estensionale dei dati al termine dell'esecuzione della funzione differisce da quello di partenza come segue:

  Variazioni nel dominio di interpretazione: nessuna

  Variazioni nelle ennuple di predicati:

  Valore di ritorno: Sia $\mathsf{CAT_{dmin}}$ l'insieme delle categorie degli hotel nella città c a distanza minima dal centro e a tariffa entro la soglia t (per comodità, l'insieme definisce anche la distanza dal centro, uguale per tutte le categorie di hotel riportate):

$$\mathsf{CAT_{dmin}} = \left\{ \mathsf{cat, d} \;\middle|\; \begin{array}{ll} \exists \mathsf{h, ta} & \mathsf{Hotel(h)} \wedge \mathsf{citHot(c,h)} \wedge \\ & \mathsf{categoria(h,cat)} \wedge \mathsf{tariffa(h,ta)} \wedge \\ & \mathsf{ta} \leq \mathsf{t} \wedge \mathsf{distCentro(h,d)} \wedge \\ & \nexists \mathsf{h', ta', d'} \quad \mathsf{Hotel(h')} \wedge \mathsf{citHot(c,h')} \wedge \\ & \qquad\qquad \mathsf{tariffa(h', ta')} \wedge \mathsf{ta'} \leq \mathsf{t} \wedge \\ & \qquad\qquad \mathsf{d'} < \mathsf{d} \end{array} \right\}$$

  Sia $\mathsf{cat_A}$ la categoria più alta rappresentata in $\mathsf{CAT_{dmin}}$, e sia $\mathsf{d_A}$ la distanza dal centro associata. Formalmente, $(\mathsf{cat_A}, \mathsf{d_A})$ soddisfa la seguente formula (l'assegnamento è unico, perché gli elementi dell'insieme hanno tutti lo stesso valore per la componente d):

$$(\mathsf{cat_A}, \mathsf{d_A}) \in \underset{(\mathsf{cat,d}) \in \mathsf{CAT_{dmin}}}{\operatorname{argmax}} (\mathsf{cat})$$

  Siano:

$$\mathsf{result} = \left\{ \mathsf{h} \;\middle|\; \begin{array}{ll} \exists \mathsf{cat, d} & \mathsf{Hotel(h)} \wedge \mathsf{citHot(c,h)} \wedge \mathsf{categoria(h,cat)} \wedge \\ & \mathsf{distCentro(h,d)} \wedge \mathsf{cat} \geq \mathsf{cat_A} \wedge \\ & \mathsf{cat} = \mathsf{cat_A} \rightarrow \\ & \quad \mathsf{d} \leq (1 + \mathsf{DELTA\_DIST\_STESSA\_CAT}) \times \mathsf{d_A} \wedge \\ & \mathsf{cat} > \mathsf{cat_A} \rightarrow \\ & \quad \mathsf{d} \leq (1 + \mathsf{DELTA\_DIST\_CAT\_SUP}) \times \mathsf{d_A} \end{array} \right\}$$

  dove DELTA_DIST_STESSA_CAT e DELTA_DIST_CAT_SUP sono simboli di costante (che, in fase di progettazione, diventeranno parametri di configurazione del sistema, inizialmente assegnati rispettivamente ai valori 0.1 e 0.2, come da requisiti).

Specifica Use-Case GestioneFF

- **numMigliaVoli(p : Prenotazione) : intero > 0 (Req. 9.3.)**

  precondizioni: nessuna

  postcondizioni:

  Modifica del livello estensionale dei dati: nessuna

  Valore di ritorno: Detto:

$$V = \{(v, n, m) \mid \text{voloPren}(v, p) \land \text{pax}(v, p, n) \land \text{miglia}(v, m)\}$$

$$\text{result} = \sum_{(v,n,m)\in V} m \times n.$$

- **amplificazione(p : Prenotazione) : intero $\in [1, 3]$ (Req. 9.3.)**

  precondizioni: nessuna

  postcondizioni:

  Modifica del livello estensionale dei dati: nessuna

  Valore di ritorno: Result soddisfa la seguente formula:

$$\begin{array}{ll}
[(\neg\exists h\ \text{prenHot}(p, h)) \rightarrow & \text{(p non è relativa ad alcun hotel)} \\
\quad \text{result} = 1] & \\
\land & \\
[(\exists h\ \text{prenHot}(p, h)) \rightarrow & \text{(p è relativa ad un hotel)} \\
\quad \forall\ h, c & \\
\qquad \text{prenHot}(p, h) \land \text{cat}(h, c)\ \rightarrow (& \\
\qquad\quad (c \leq 4\ \rightarrow\ \text{result} = 2) & \\
\qquad\quad \land & \\
\qquad\quad (c = 5\ \rightarrow\ \text{result} = 3) & \\
\qquad) & \\
] &
\end{array}$$

- **numMiglia(p : Prenotazione) : intero ≥ 0 intero (Req. 9.3.)**

  precondizioni: nessuna

  postcondizioni:

  Modifica del livello estensionale dei dati: nessuna

  Valore di ritorno:

$$\text{result} = \text{GestioneFF.numMigliaVoli}(p) \times \text{GestioneFF.amplificazione}(p)$$

- **numeroMigliaFrequentFlyer(f : FF, mom : dataora) : intero $>$ 0** (Req. 9.3.)

  precondizioni: nessuna

  postcondizioni:

  Modifica del livello estensionale dei dati: nessuna

  Valore di ritorno: Detto:

$$P = \{p \mid \text{clPr}(f, p) \wedge \exists\, a, i, di \\ \text{affiliazione}(f, a) \wedge \text{istante}(p, i) \wedge \text{data}(i, di) \wedge \\ di \geq a \wedge i < mom\}$$

$$\text{result} = \sum_{p \in P} \text{GestioneFF.numMiglia}(p)$$

Parte P

Progettazione della Base Dati e delle Funzionalità

P.1 Scelta del DBMS e Ristrutturazione del Diagramma ER e delle Specifiche dei Dati

P.1.1 Testo

Iniziare la fase di progettazione logica della base di dati decidendo il DBMS da utilizzare e ristrutturando il diagramma ER concettuale e le specifiche dei dati.

P.1.2 Soluzione

Disclaimer

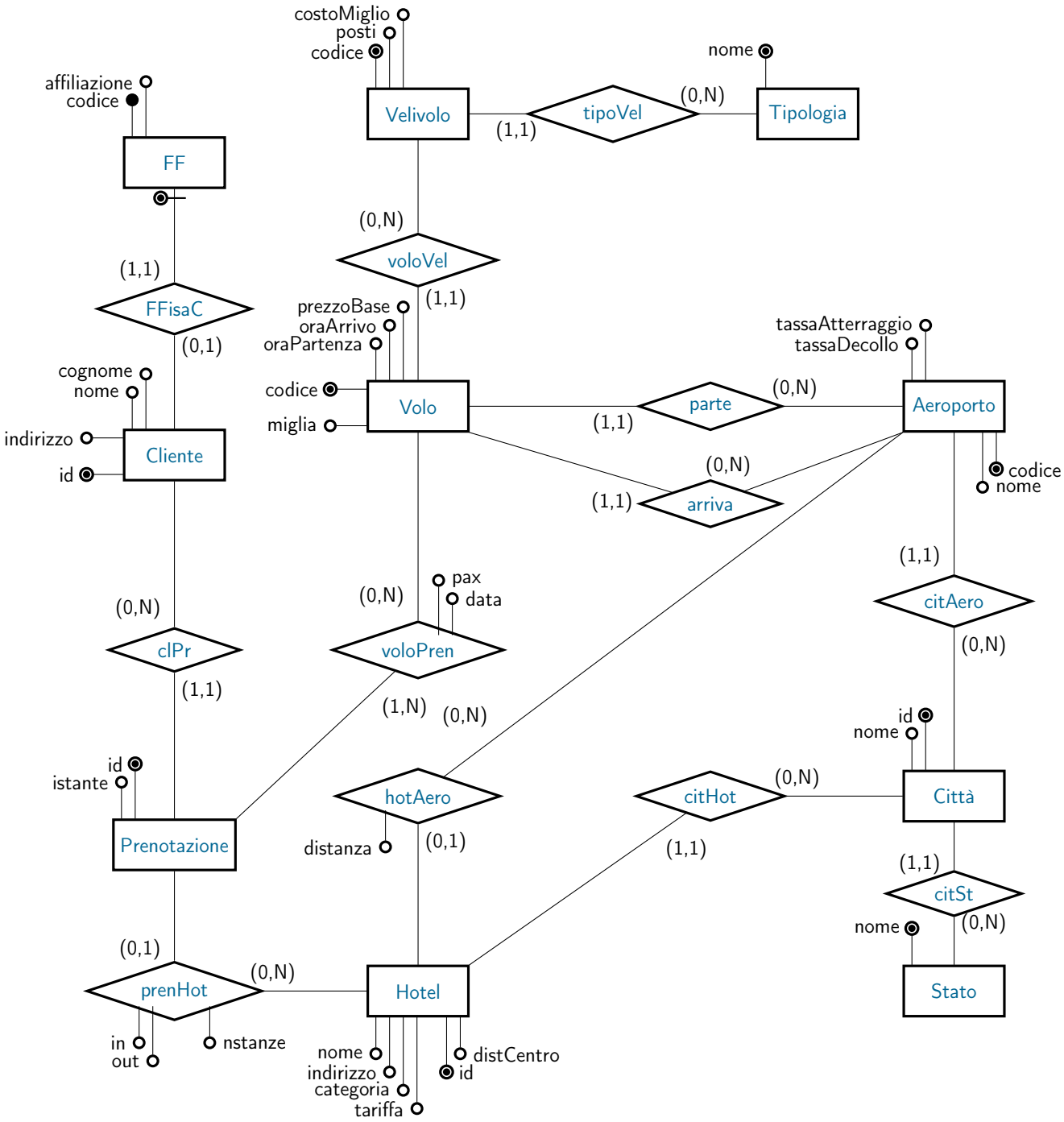
Questa sezione ha bisogno di essere attentamente verificata.
Si pregano gli studenti di controllare la correttezza di tutti i dettagli e di riportare al docente eventuali errori rilevati. Grazie.

P.1.2.1 Scelta del DBMS

Si decide di utilizzare il DBMS PostgreSQL.

Nota: La scelta del DBMS è importante, in quanto può avere un impatto sul modo in cui vengono progettati i domini, i vincoli e le operazioni di use-case. In una fase di progetto completa, andrebbero prese anche altre decisioni, come l'architettura dell'applicazione ed il linguaggio di programmazione per quest'ultima. Tuttavia, dato lo scopo di questo corso, non prenderemo decisioni in tal senso.

P.1.2.2 Ristrutturazione del Diagramma ER e delle Specifiche dei Dati



Specifiche dei Dati

Entità Cliente

Ogni istanza di questa entità rappresenta un cliente (Req. 1.)

attributo	dominio	molteplicità	descrizione
nome	StringM		Il nome del cliente
cognome	StringM		Il cognome del cliente
indirizzo	Indirizzo		L'indirizzo del cliente
id	integer		L'identificatore univoco del cliente

Entità FF

Ogni istanza di questa entità rappresenta un cliente "frequent flyer" (Req. 1.4.)

attributo	dominio	molteplicità	descrizione
codice	StringS		Il codice del frequent flyer
affiliazione	date		La data di affiliazione del frequent flyer

Entità Aeroporto

Ogni istanza di questa entità rappresenta un aeroporto (Req. 4.)

attributo	dominio	molteplicità	descrizione
codice	char(3)		Il codice dell'aeroporto
nome	StringM		Il nome dell'aeroporto
tassaDecollo	MoneyGEZ		L'importo delle tasse di decollo dell'aeroporto
tassaAtterraggio	MoneyGEZ		L'importo delle tasse di atterraggio dell'aeroporto

Entità Volo
Ogni istanza di questa entità rappresenta un volo (Req. 3.)

attributo	dominio	molteplicità	descrizione
codice	StringS		Il codice del volo
miglia	IntegerGZ		Il numero di miglia percorse dal volo
oraPartenza	time with timezone		L'orario di partenza del volo
oraArrivo	time with timezone		L'orario di arrivo del volo
prezzoBase	MoneyGEZ		Il prezzo base del volo

Vincoli:

[V.Volo.aeroportiDiversi] Gli aeroporti di partenza e di arrivo di un volo devono essere diversi.

Formalmente:

$$\begin{aligned}&\forall v,ap,aa,\\&\quad \text{Volo}(v) \wedge \text{partenza}(v,ap) \wedge \text{arrivo}(v,aa)\\&\quad \rightarrow\ ap \neq aa)\end{aligned}$$

[V.Volo.maiOverbooking] Il numero di posti prenotati non deve mai superare la capacità del velivolo che effettua il volo. Ciò equivale a dire che il numero di posti disponibili per un volo in una certa data deve essere sempre maggiore o uguale a zero.

Formalmente:

$$\begin{aligned}&\forall v,d,i,\\&\quad \text{Volo}(v) \wedge \text{data}(v,d) \wedge \text{dataora}(i)\\&\quad \rightarrow\ \text{Posti.postiDisponibili}(v,d,i) \geq 0\end{aligned}$$

dove Posti.postiDisponibili è una operazione dello Use-Case Posti.

[V.Volo.prezzoBase] Il valore ridondato dell'attributo prezzoBase deve essere consistente.

Formalmente:

$$\begin{aligned}&\forall\ v, ap, aa, td, ta, vel, cm, m, val, cap\\&\text{valVolo}(v,val) \wedge \text{prezzoBase}(v,p)\ \rightarrow\\&p = \left(\frac{m * cm + td + ta}{cap}\right) \times (1 + \text{RICARICO})\end{aligned}$$

dove RICARICO è un parametro di configurazione del sistema, inizialmente assegnato al valore 0.2.

Entità Velivolo

Ogni istanza di questa entità rappresenta un velivolo (Req. 7.)

attributo	dominio	molteplicità	descrizione
codice	StringS		Il codice del velivolo
posti	IntegerGZ		Il numero di posti del velivolo
costoMiglio	MoneyGEZ		Il costo al miglio del velivolo

Entità Tipologia

Ogni istanza di questa entità rappresenta una tipologia di velivolo (Req. 7.2.)

attributo	dominio	molteplicità	descrizione
nome	StringM		Il nome della tipologia di velivolo

Entità Città

Ogni istanza di questa entità rappresenta una città (Req. 6.)

attributo	dominio	molteplicità	descrizione
nome	StringM		Il nome della città
id	integer		L'identificatore univoco della città

Entità Stato

Ogni istanza di questa entità rappresenta uno stato (Req. 6.2.)

attributo	dominio	molteplicità	descrizione
nome	StringM		Il nome dello stato

Entità Hotel
Ogni istanza di questa entità rappresenta un hotel (Req. 8.)

attributo	dominio	molteplicità	descrizione
nome	StringM		Il nome dell'hotel
indirizzo	Indirizzo		L'indirizzo dell'hotel
categoria	CategoriaHotel		La categoria dell'hotel
tariffa	MoneyGEZ		La tariffa dell'hotel
distCentro	IntegerGZ		La tariffa dell'hotel
id	integer		L'identificatore univoco dell'hotel

Entità Prenotazione
Ogni istanza di questa entità rappresenta una prenotazione (Req. 5.)

attributo	dominio	molteplicità	descrizione
istante	timestamp with timezone		L'istante della prenotazione
id	integer		L'identificatore univoco della prenotazione

Vincoli:

[V.Prenotazione.date] L'istante di prenotazione per un volo deve essere antecedente all'orario di partenza del volo.

Formalmente:

$$\begin{aligned}&\forall p,i,v,dv,ov,iv\\ &\quad \text{Prenotazione}(p) \wedge \text{istante}(p,i) \wedge \text{voloPren}(v,p)\\ &\quad \text{data}(v,dv) \wedge \text{oraPart}(v,ov) \wedge \text{dataora}(iv)\\ &\quad \text{data}(iv,dv) \wedge \text{ora}(iv,ov)\\ &\quad\quad \rightarrow\ iv > i\end{aligned}$$

[V.Prenotazione.checkin] L'istante di prenotazione per un Hotel deve essere antecedente alla data di check-in.

Formalmente:

$$\begin{aligned}&\forall\ p,i,ci,co,id,h\\ &\quad \text{Prenotazione}(p) \wedge \text{Hotel}(h) \wedge \text{prenHot}(p,h,ci,co) \wedge \text{istante}(p,i) \wedge \text{in}(p,ci)\\ &\quad \text{data}(i,id)\ \rightarrow\ id \leq ci\end{aligned}$$

Relationship FFisaC
Ogni istanza di questa relationship lega un'istanza di FF al Cliente relativo
Attributi: Nessuno

Relationship tipoVel
Ogni istanza di questa relationship lega un Velivolo a una Tipologia di velivolo
Attributi: Nessuno

Relationship prenHot
Ogni istanza di questa relationship lega una Pranotazione a un Hotel

attributo	dominio	molteplicità	descrizione
in	date		La data di check-in
out	date		La data di check-out
nstanze	IntegerGZ		Il numero di stanze prenotate

Vincoli:

[V.prenHot.date] La data di check-in per ogni prenotazione di Hotel deve essere antecedente alla data di check-out.

Formalmente:

$$\forall p,i,o,h \quad \text{Prenotazione}(p) \wedge \text{Hotel}(h) \wedge \text{in}(p,i) \wedge \text{out}(p,o) \wedge \text{prenHot}(p,h,i,o) \rightarrow i < o$$

Relationship voloPren
Ogni istanza di questa relationship lega un Volo a una Prenotazione

attributo	dominio	molteplicità	descrizione
data	date		La data del Volo prenotato
pax	IntegerGZ		Il numero di posti prenotati

Relationship parte
Ogni istanza di questa relationship lega un Volo all'Aeroporto di partenza
Attributi: Nessuno

Relationship arriva
Ogni istanza di questa relationship lega un Volo all'Aeroporto di arrivo
Attributi: Nessuno

Relationship citSt
Ogni istanza di questa relationship lega una Città allo Stato in cui si trova
Attributi: Nessuno

Relationship citHot
Ogni istanza di questa relationship lega un Hotel alla Città in cui si trova
Attributi: Nessuno

Relationship citAero
Ogni istanza di questa relationship lega un Aeroporto alla Città in cui si trova
Attributi: Nessuno

Relationship voloVel
Ogni istanza di questa relationship lega un Volo al Velivolo che lo effettua
Attributi: Nessuno

Relationship clPr
Ogni istanza di questa relationship lega un Cliente a una sua Prenotazione
Attributi: Nessuno

Relationship hotAero
Ogni istanza di questa relationship lega un Hotel all'Aeroporto più vicino

attributo	dominio	molteplicità	descrizione
distanza	IntegerGZ		La distanza tra l'Hotel e l'Aeroporto

Dominio Indirizzo
Il dominio è un record composto dai seguenti campi:

- via: StringM
- civico: IntegerGZ (0,1)
- CAP: char(5)

Dominio IntegerGZ
Il dominio è dato dal sottoinsieme del dominio integer formato dai valori > 0.

Dominio MoneyGEZ
Il dominio è dato dal sottoinsieme del dominio real formato dai valori $>= 0$.

Dominio CategoriaHotel
Il dominio è dato dal sottoinsieme del dominio integer formato dai valori $>= 1$ e $<= 5$.

Dominio StringS
Il dominio è dato dall'insieme delle stringhe di al più 20 caratteri.

Dominio StringM
Il dominio è dato dall'insieme delle stringhe di al più 100 caratteri.

Dominio StringL
Il dominio è dato dall'insieme delle stringhe di al più 1000 caratteri.

P.2 Schema Relazionale della Base Dati

P.2.1 Testo

Proseguire la fase di progettazione logica della base di dati producendo lo schema relazionale della base dati e i relativi vincoli (vincoli di chiave, di chiave primaria, di foreign key, di inclusione, di ennupla, di dominio) a partire dallo schema ER ristrutturato.

P.2.2 Soluzione

P.2.2.1 Definizione delle Relazioni Derivanti da Entità e Relationship Accorpate ad Entità

Cliente (id:serial, nome:StringM, cognome:StringM, indirizzo:Indirizzo)

FF (cliente:integer, codice: StringS, affiliazione: date)
[VincoloDB.1] *foreign key:* cliente references Cliente(id)
[VincoloDB.2] *chiave:* codice
La relazione accorpa la relationship FFisaC.

Prenotazione (id:integer, istante: timestamptz, cliente: integer, hotel*: integer, in*: date, out*:date, nstanze*:IntegerGZ)
[VincoloDB.3] *foreign key:* cliente references Cliente(id)
[VincoloDB.4] *foreign key:* hotel references Hotel(id)
[VincoloDB.5] *inclusione:* id $\subseteq$ voloPren(prenotazione)
[VincoloDB.6] *serial:* I valori dell'attributo id sono generati automaticamente dal DBMS
[VincoloDB.7] *ennupla:*

$$(\text{hotel} == \text{NULL} \wedge \text{in} == \text{NULL} \wedge \text{out} == \text{NULL} \wedge \text{nstanze} == \text{NULL}) \vee (\text{hotel} \neq \text{NULL} \wedge \text{in} \neq \text{NULL} \wedge \text{out} \neq \text{NULL} \wedge \text{nstanze} \neq \text{NULL})$$

[VincoloDB.8] *ennupla:* $\text{in} \neq \text{NULL} \rightarrow \text{out} > \text{in}$ (implementa [V.prenHot.date])
[VincoloDB.9] *ennupla:* $\text{in} \neq \text{NULL} \rightarrow \text{istante} \leq \text{in}$ (implementa [V.Prenotazione.checkin])

La relazione accorpa le relationship clPr e prenHot.

Aeroporto (codice:char(3), nome: StringM, tassaDecollo: MoneyGEZ, tassaAtterraggio: MoneyGEZ, citta: integer)
[VincoloDB.10] *foreign key:* citta references Citta(id)
La relazione accorpa le relationship valAerop e citAero.

Volo (codice:StringS, aeropPart: char(3), aeropArr: char(3), oraPartenza: timestamp with timezone, oraArrivo: timestamp with timezone, miglia: IntegerGZ, prezzoBase: MoneyGEZ, velivolo: StringS)
[VincoloDB.11] *foreign key:* aeropPart references Aeroporto(codice)
[VincoloDB.12] *foreign key:* aeropArr references Aeroporto(codice)
[VincoloDB.13] *foreign key:* velivolo references Velivolo(codice)
[VincoloDB.14] *ennupla:* aeropPart $\neq$ aeropArr (implementa [V.Volo.aeroportiDiversi])

La relazione accorpa le relationship parte, arriva e voloVel.

Velivolo (codice:StringS, pax: IntegerGZ, costoMiglio: MoneyGEZ, tipoVel: StringM)

[VincoloDB.15] *foreign key:* tipoVel references Tipologia(nome)
La relazione accorpa la relationship tipoVel.

Tipologia (nome:StringM)

Hotel (id:serial, nome: StringM, indirizzo: Indirizzo, categoria: CategoriaHotel, tariffa: MoneyGEZ, citta: integer, aeroporto*: char(3), distanzaAero*:IntegerGZ)
[VincoloDB.16] *foreign key:* citta references Citta(id)
[VincoloDB.17] *foreign key:* aeroporto references Aeroporto(codice)
[VincoloDB.18] *ennupla:* aeroporto $\neq$ NULL $\iff$ distanzaAero $\neq$ NULL
[VincoloDB.19] *serial:* I valori dell'attributo id sono generati automaticamente dal DBMS
La relazione accorpa le relationship hotAero e citHot.

Citta (id:serial, nome: StringM, stato: StringM)
[VincoloDB.20] *foreign key:* stato references Stato(nome)
[VincoloDB.21] *serial:* I valori dell'attributo id sono generati automaticamente dal DBMS
La relazione accorpa la relationship citSt.

Stato (nome:StringM)

P.2.2.2 Definizione delle Relazioni Derivanti da Relationship non Accorpate ad Entità

voloPren (prenotazione:integer, volo:StringS, data: date, pax: IntegerGZ)
[VincoloDB.22] *foreign key:* prenotazione references Prenotazione(id)
[VincoloDB.23] *foreign key:* volo references Volo(codice)

P.3 Progettazione dei Vincoli Esterni

P.3.1 Testo

Proseguire la fase di progettazione logica della base di dati progettando come imporre i vincoli di integrità sui dati non esprimibili come vincoli di chiave, di chiave primaria, di foreign key, di inclusione, di ennupla, di dominio.

P.3.2 Soluzione

Disclaimer

Questa sezione ha bisogno di essere attentamente verificata.
Si pregano gli studenti di controllare la correttezza di tutti i dettagli e di riportare al docente eventuali errori rilevati. Grazie.

Vincolo V.Volo.maiOverbooking

Il vincolo impone che il numero di posti prenotati non deve mai superare la capacità del velivolo che effettua il volo. Ciò equivale a dire che il numero di posti disponibili per un volo in una certa data deve essere sempre maggiore o uguale a zero.

Dalla specifica logica del vincolo, isoliamo le operazioni che possono portare una base dati legale a violarlo, ovvero:

1. Inserimento o modifica di una ennupla nella relazione voloPren.

2. Modifica di una ennupla nella relazione Volo, in particolare se dovesse essere modificato il velivolo utilizzato.

3. Modifica di una ennupla nella relazione Velivolo, in particolare se dovesse essere modificato il numero di posti.

Si decide di procedere definendo una funzione postiDisponibili () nel DBMS, un trigger e un'opportuna politica di accesso ai dati. La funzione di DBMS sarà utilizzata nel corpo del trigger e sarà invocata dalle operazioni di use-case che ne avranno bisogno.

DB.postiDisponibili(v : StringS, d : date, i : timestamp) : integer

algoritmo:

// La funzione restituisce il numero di posti disponibili, all'istante 'i', sul volo 'v' previsto per il giorno 'd'

1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':v', ':d' e ':i' i valori dei parametri attuali v, d, i:

```
select vel.pax - x.postiPrenotati as postiLiberi
from Volo v, Velivolo vel,
  ( select sum(vp.pax) as postiPrenotati
      from Prenotazione p, voloPren vp
    where vp.prenotazione = p.id and vp.volo = :v
    and vp.data = :d and p.istante <= :i
  ) x
where v.codice = :v and v.velivolo = vel.codice
```

if *Q == NULL* **then**
generare l'errore 'Volo non trovato';
else
return *il valore della colonna 'postiLiberi' dell'unica ennupla in* Q;

[VincoloDB.24] *trigger:*

Operazioni: inserimento di una ennupla nella relazione voloPren

Istante di Invocazione: dopo l'operazione intercettata

Funzione:

1 new ← l'ennupla inserita;
2 isError ←

```
exists (
  select *
  from Volo v   where v.codice = new.volo
    and DB.postiDisponibili(v.codice, new.data,
      CURRENT_TIMESTAMP) < 0
)
```

3 **if** isError **then** blocca l'operazione;
4 **else** permetti l'operazione;

Politiche di accesso ai dati

Sia <utenti> la lista degli utenti delle applicazioni che dovranno interagire con la base dati. Si definiscono le seguenti politiche di accesso ai dati:

Blocco modifiche di ennuple della relazione voloPren

```
revoke update on volopren from <utenti>;
```

Difatti, poiché una ennupla di questa relazione rappresenta una prenotazione per un certo volo, si vuole impedire che vengano modificati i suoi attributi. In questo modo costringiamo il Sistema Prenotazioni (sistema esterno), in caso volesse modificare il numero di posti associati ad una prenotazione per un certo volo, a cancellare la ennupla e crearne una nuova (il controllo dei posti disponibili sarebbe comunque corretto, in quanto verrebbe effettuato per l'istante CURRENT_TIMESTAMP).

Blocco modifiche di ennuple della relazione Velivolo

```
revoke update on velivolo from <utenti>;
```

In questo semplice progetto, decidiamo di impedire del tutto la modifica di ennuple della relazione Velivolo, in quanto queste potrebbero portare la base dati a violare il vincolo [V.Volo.maiOverbooking] (si pensi al tentativo di riduzione della capacità di un velivolo). Si lascia per esercizio la definizione di una politica più flessibile.

Limitazione modifiche di una ennupla nella relazione Volo Si vuole impedire che si possano modificare orari e aeroporti di partenza e di arrivo di un volo, in quanto ciò equivarrebbe a definire un nuovo volo. D'altra parte, è necessario che l'attributo 'prezzoBase', il quale ridonda il prezzo base del volo, possa essere modificato, così da permettere il suo aggiornamento nel caso in cui i valori da cui dipende cambino (*p.es.*, modifica dell'importo delle tasse di decollo dell'aeroporto di partenza).

```
revoke update on volo from <utenti>;
grant update(prezzoBase) on volo to <utenti>;
```

Alternativamente (se il DBMS utilizzato non dovesse supportare politiche di accesso ai dati a livello di singoli attributi) è possibile intercettare, mediante un altro trigger, i tentativi di update sulle relazioni coinvolte e generare una eccezione.

Vincolo V.Prenotazione.date

Il vincolo impone che ogni prenotazione debba essere effettuata prima dell'istante di partenza di ogni volo prenotato.

Dalla specifica logica del vincolo, isoliamo le operazioni che possono portare una base dati legale a violarlo, ovvero:

1. Inserimento o modifica di una ennupla nella relazione voloPren.
2. Modifica di una ennupla nella relazione Volo.

La modifica di ennuple delle relazioni voloPren e Volo è stata già impedita con un'opportuna politica di accesso ai dati per gestire il vincolo [V.Volo.maiOverbooking].

Si decide di procedere definendo un opportuno trigger.

[VincoloDB.25] *trigger:*

Operazioni: inserimento di una ennupla nella relazione voloPren

Istante di Invocazione: prima dell'operazione intercettata

Funzione:

1 new ← l'ennupla inserita;
2 isError ←

```
exists (
  select *
  from Volo v
  where v.codice = new.volo
    and CURRENT_TIMESTAMP >= (new.data + v.oraPartenza)
)
```

3 **if** isError **then** blocca l'operazione;
4 **else** permetti l'operazione;

Vincolo V.Volo.prezzoBase

Il vincolo impone che il valore dell'attributo 'prezzoBase' delle ennuple della relazione Volo sia consistente con il valore calcolabile, come da requisiti, a partire dalle tasse di decollo e atterraggio degli aeroporti di partenza e arrivo, il costo al miglio del velivolo e il numero di posti totali del velivolo.

Dalla specifica logica del vincolo, isoliamo le operazioni che possono portare una base dati legale a violarlo, ovvero:

1. Inserimento o modifica di una ennupla nella relazione Volo, in particolare se si dovesse modificare l'aeroporto di partenza o quello di arrivo.

2. Modifica di una ennupla nella relazione Aeroporto.

3. Modifica di una ennupla nella relazione Velivolo, in particolare se si dovesse modificare il costo al miglio o il numero di posti totali.

La modifica delle ennuple nelle relazioni Volo e Velivolo è già stata impedita tramite un'opportuna politica dell'accesso ai dati per gestire altri vincoli.

Per l'inserimento nella relazione Volo e la modifica nella relazione Aeroporto si procede definendo opportuni trigger.

Si definisce prima di tutto la seguente funzione nel DBMS, che permetterà di calcolare il prezzo base di un volo (è necessaria una funzione nel DBMS affinché possa essere invocata da altre query e comandi SQL).

DB.prezzoBase(v : StringS) : MoneyGEZ

algoritmo:

1 Q ← risultato della query SQL ottenuto dallo scheletro seguente sostituendo ad ogni segnaposto denotato da ':' il valore dell'omonima variabile locale o parametro attuale dell'operazione, o parametro di configurazione del sistema:

```
select ((v.miglia * vel.costoMiglio +
     ap.tassaDecollo +
     aa.tassaAtterraggio) /
   vel.pax) * (1 + :RICARICO) as prezzo
from Volo v, Velivolo vel, Aeroporto ap, Aeroporto aa
where v.codice = :v and v.velivolo = vel.codice
  and v.aeropPart = ap.codice and v.aeropArr = aa.codice
```

2 **if** *Q* == ***NULL*** **then**
3 generare l'errore 'Volo non trovato';
4 **else**
5 **return** *il valore della colonna 'prezzo' dell'unica ennupla in* Q;

[VincoloDB.26] *trigger:*

Operazioni: inserimento di una ennupla nella relazione Volo

Istante di Invocazione: dopo l'operazione intercettata

Funzione:

1 new ← l'ennupla inserita o risultato della modifica;
2 Eseguire il seguente comando SQL:

```
update Volo
  set prezzoBase = DB.prezzoBase(new.codice)
    where v.codice = new.codice
```

if *il comando precedente ha restituito un errore* **then**
impedisci l'operazione;
else permetti l'operazione;

[VincoloDB.27] *trigger:*

Operazioni: modifica di una ennupla nella relazione Aeroporto

Istante di Invocazione: dopo l'operazione intercettata

Funzione:

1 old ← l'ennupla oggetto della modifica;
2 new ← l'ennupla risultato della modifica;
3 **if** *new.tassaDecollo* ≠ *old.tassaDecollo or new.tassaAtterraggio* ≠ *old.tassaAtterraggio* **then**
4 Eseguire il seguente comando SQL:

```
update Volo
set prezzoBase = DB.prezzoBase(codice)
where aeropPart = new.codice or aeropArr = new.codice
```

if *il comando precedente ha restituito un errore* **then**
impedisci l'operazione;
else permetti l'operazione;
5 **else**
6 permetti l'operazione;

P.4 Specifiche Realizzative degli Use-Case

P.4.1 Testo

Proseguire la fase di progettazione dell'applicazione producendo le specifiche realizzative delle operazioni di use-case.

P.4.2 Soluzione

Disclaimer

Questa sezione ha bisogno di essere attentamente verificata.
Si pregano gli studenti di controllare la correttezza di tutti i dettagli e di riportare al docente eventuali errori rilevati. Grazie.

Specifica Use-Case Posti

- **postiDisponibili(v : StringS, d : date, i : timestamp) : integer**

  algoritmo:

  1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':v', ':d' e ':i' il valore degli omonimi parametri attuali:

  ```
  select DB.postiDisponibili(:v,:d,:i) as postiLiberi
  ```

  2 **if** Q *rappresenta un errore* **then**
  3 inoltra l'errore;
  4 **else**
  5 **return** *il valore della colonna 'postiLiberi' dell'unica ennupla in* Q;

Specifica Use-Case Prezzi

- **prezzoBase(v : StringS) : MoneyGEZ**

  algoritmo:

  1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':v' il valore del parametro attuale v:

  ```
  select v.prezzoBase as prezzoBase
  from Volo v
  where v.codice = :v
  ```

  2 **if** Q *rappresenta un errore* **then**
  3 inoltra l'errore;
  4 **else**
  5 **return** *il valore della colonna 'prezzoBase' dell'unica ennupla in* Q;

- **prezzoComplessivoSingoloPosto(v : StringS, d : date) : MoneyGEZ**

  algoritmo:

  1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':v' e ':d' il valore degli omonimi parametri attuali:

  ```
  select DB.prezzoComplessivoSingoloPosto(:v,:d)
     as prezzoComplessivo
  ```

  2 **if** Q *rappresenta un errore* **then**
  3 inoltra l'errore;
  4 **else**
  5 **return** *il valore della colonna 'prezzoComplessivo' dell'unica ennupla in* Q;

- **prezzoComplessivo(v : StringS, d : date, n : IntegerGZ) : MoneyGEZ**

  algoritmo:

  1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':v', ':d' e ':n' il valore degli omonimi parametri attuali:

  ```
  select :n * DB.prezzoComplessivoSingoloPosto(:v,:d)
     as prezzoComplessivo
  ```

  2 **if** Q *rappresenta un errore* **then**
  3 inoltra l'errore;
  4 **else**
  5 **return** *il valore della colonna 'prezzoComplessivo' dell'unica ennupla in* Q;

Specifica Use-Case Prenotazioni

- **suggerisciHotel(c : integer, t : MoneyGEZ) : Insieme(⟨id: integer, nome: stringM, indirizzo: Indirizzo, categoria: CategoriaHotel, tariffa: MoneyGEZ, aeroporto*: char(3), distanzaAero*:IntegerGZ⟩)**

  algoritmo:

  1 H ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':c' e ':t' il valore degli omonimi parametri attuali

```
select h.id, h.nome, h.indirizzo, h.categoria, h.tariffa, h.
    aeroporto, h.distanzaAero
from Hotel h, Hotel piu_vic_max_cat
where
  piu_vic_max_cat.citta = :c and piu_vic_max_cat.tariffa <= :t
  and piu_vic_max_cat.distCentro <= all(
    select altro_t.distCentro
    from Hotel altro_t
    where altro_t.citta = :c and altro_t.tariffa <= :t)
  and piu_vic_max_cat.categoria >= all(
    select altro_t.categoria
    from Hotel altro_t
    where altro_t.citta = :c and altro_t.tariffa <= :t)
  and h.citta = :c
  and (
    (h.categoria = piu_vic_max_cat.categoria and h.distCentro <=
     (1+DELTA_DIST_STESSA_CAT) * piu_vic_max_cat.distCentro)
    or (h.categoria > piu_vic_max_cat.categoria and h.distCentro
        <=
     (1+DELTA_DIST_CAT_SUP) * piu_vic_max_cat.distCentro)
  )
```

  if H *rappresenta un errore* **then**
  inoltra l'errore;
  else
  if H *è NULL* **then**
  inoltra l'errore 'non esistono hotel che soddisfano i requisiti richiesti';
  else restituisci H;

Specifica Use-Case GestioneFF

- **numMigliaVoli(p : integer) : IntegerGZ**

  algoritmo:

  1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':p' il valore del parametro attuale p:

  ```
  select DB.numMigliaVoli(:p) as numeroMiglia
  ```

  2 **if** Q *rappresenta un errore* **then**
  3 inoltra l'errore;
  4 **else**
  5 **return** *il valore della colonna 'numeroMiglia' dell'unica ennupla in* Q;

- **numMiglia(p : integer) : IntegerGZ**

  algoritmo:

  1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':p' il valore del parametro attuale p:

  ```
  select DB.numMiglia(:p) as numeroMiglia
  ```

  2 **if** Q *rappresenta un errore* **then**
  3 inoltra l'errore;
  4 **else**
  5 **return** *il valore della colonna 'numeroMiglia' dell'unica ennupla in* Q;

- **numeroMigliaFrequentFlyer(f : integer, mom : timestamp) : IntegerGZ**

  algoritmo:

  1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':f' e ':mom' il valore degli omonimi parametri attuali:

  ```
  select sum(DB.numMiglia(:p)) as numeroMiglia
  from Prenotazione p, FF f
  where f.cliente = :f and p.cliente = f.cliente
    and p.istante >= f.affiliazione and p.istante < :mom
  ```

  2 **if** Q *rappresenta un errore* **then**
  3 inoltra l'errore;
  4 **else**
  5 **return** *il valore della colonna 'numeroMiglia' dell'unica ennupla in* Q;

Si definiscono le seguenti funzioni nel DBMS:

DB.prezzoComplessivoSingoloPosto(v : StringS, d : date) : MoneyGEZ

algoritmo:

1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':v' e ':d' il valore degli omonimi parametri attuali:

```
select v.prezzoBase *
  (case when x.diff > 0
    then power(1 - SCONTO, x.diff)
    else power(1 + SOVRAPPREZZO, -x.diff) end) as
       prezzoComplessivo
from Volo v, (
  select (vel.pax/2)
    - DB.postiDisponibili(:v, :d, CURRENT_TIMESTAMP) as diff
  from Velovolo vel, Volo v
  where v.codice = :v and v.velivolo = vel.codice
  ) x
where v.codice = :v
```

2 **if** Q *è l'insieme vuoto* **then**
3 genera l'errore 'Volo con codice v non trovato';
4 **else**
5 **return** *il valore della colonna 'prezzoComplessivo' dell'unica ennupla di* Q;

DB.numMigliaVoli(p : integer) : IntegerGEZ

algoritmo:

1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':p' il valore del parametro attuale 'p':

```
select sum(v.miglia * vp.pax) as numeroMiglia
from Prenotazione p, Volo v, voloPren vp
where p.id = :p and vp.prenotazione = p.id and vp.volo = v.
    codice
```

2 **if** Q *è l'insieme vuoto* **then**
3 genera l'errore 'Prenotazione con id p non trovata';
4 **else**
5 **return** *il valore della colonna 'numeroMiglia' dell'unica ennupla di* Q;

DB.amplificazione(p : integer) : IntegerGZ

algoritmo:

1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':p' il valore del parametro attuale 'p':

```
select (case
  when (h.categoria is NULL) then 1
  when (h.categoria <= 4) then 2
  else 3 end) as fattore
from Prenotazione p LEFT OUTER JOIN Hotel h
  on p.hotel = h.id
where p.id = :p
```

if Q *è vuoto* **then**
genera l'errore 'Prenotazione con id p non trovata';
else
return *il valore della colonna 'fattore' dell'unica ennupla di* Q;

DB.numMiglia(p : integer) : IntegerGEZ

algoritmo:

1 Q ← risultato della query SQL ottenuta dallo scheletro seguente sostituendo a ':p' il valore del parametro attuale 'p':

```
select DB.numMigliaVoli(:p) * DB.amplificazione(:p) as
    migliaTotali
```

2 **if** Q *è l'insieme vuoto* **then**
3 genera l'errore 'Prenotazione con id p non trovata';
4 **else**
5 **return** *il valore della colonna 'migliaTotali' dell'unica ennupla di* Q;

Esercizio

Le operazioni numMigliaVoli() e amplificazione () sono state definite in fase di Analisi solo per semplificare la specifica di questa operazione, e non sono richieste dalla specifica dei requisiti.

In fase di Analisi, per semplicità, tali operazioni ausiliarie sono state definite nello use-case GestioneFF accessibile dall'esterno. Dunque, in fase di Progettazione, abbiamo in realtà perso l'informazione che tali operazioni sono state aggiunte dall'analista solo per comodità, e che non è richiesto che siano accessibili dagli attori del sistema.

Riscrivere il diagramma concettuale degli use-case affinché le due operazioni ausiliarie siano definite in uno use-case incluso da GestioneFF e non accessibile direttamente dagli attori del sistema. Questo fa sì che, in fase di Progettazione, il progettista possa rendersi conto che tali operazioni non sono comprese

tra le funzionalità che il sistema deve offrire ai suoi attori. Riscrivere quindi la specifica realizzativa di questa operazione in termini di una *singola* query SQL, senza l'uso delle operazioni ausiliarie. Tutto questo migliorerà l'efficienza.