Progetto 20050704 (P.20050704) - RainAir

Sebastiano Deodati

22 maggio 2024

Indice

Dati di interesse e funzionalità richieste

1. clienti
    1.1. nome
    1.2. cognome
    1.3. indirizzo (req. 8)
    1.4. frequent flyers
        1.4.1. codice
        1.4.2. data di affiliazione
        1.4.3. miglia accumulate
2. voli
    2.1. codice
    2.2. miglia percorse
    2.3. orario e aeroporto di partenza (req. 4)
    2.4. orario e aeroporto di arrivo (req. 4)
    2.5. velivolo (req. 5)
3. prenotazioni
    3.1. prenotante (req. 1)
    3.2. istante
    3.3. biglietti (req. 6)
    3.4. posti
    3.5. data
    3.6. eventuale hotel (req. 7) con date di check-in e check-out e stanze prenotate
4. aeroporti
    4.1. codice
    4.2. nome
    4.3. città (req. 9)
    4.4. tassa di decollo
    4.5. tassa di atterraggio

5. velivoli
    5.1. codice
    5.2. tipo
    5.3. posti
    5.4. costo/miglio
6. bliglietti
    6.1. volo
    6.2. prezzo base, calcolato sulla base dei costi che la compagnia deve sostenere $(\frac{[miglia\ effettuate]*[costo/miglio\ del\ velivolo]+[tasse\ di\ decollo\ e\ atterraggio]}{[posti\ del\ velivolo]} * 1,2)$
7. hotel
    7.1. nome
    7.2. indirizzo (req. 8)
    7.3. categoria (da 1 a 5)
    7.4. tariffa stanza per notte
    7.5. distanza dal centro
    7.6. aeroporto più vicino (req. 4) con relativa distanza
8. Indirizzi
    8.1. via/pza
    8.2. civico
    8.3. CAP
    8.4. città (req. 9)
9. città
    9.1. nome
    9.2. stato
10. funzionalità richieste
    10.1. Il Sistema prenotazioni 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).

10.2. 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:

- Il prezzo base, che dipende dal volo
- Il numero di posti disponibili al momento corrente per la data di volo richiesta.

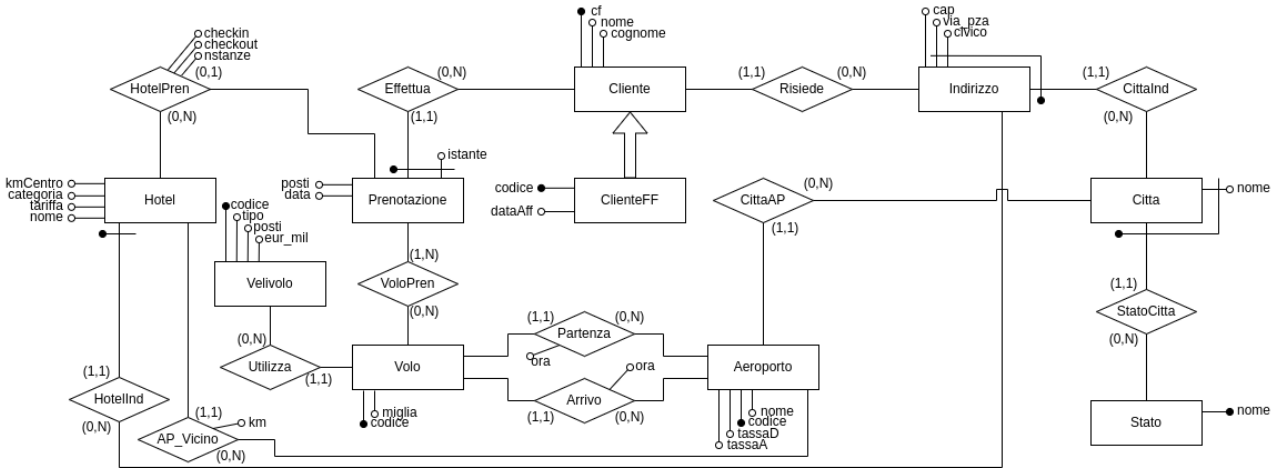
Il calcolo del prezzo del biglietto parte dal suo prezzo base, e subisce poi delle modifiche a seconda della disponibilità attuale di posti sulla data di volo richiesta. In particolare, al prezzo base si applica la seguente regola: se il numero di posti disponibili al momento della prenotazione per il 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.

10.3. 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.

10.4. Il sistema deve inoltre gestire il sistema di benefici dei "frequent flyers", dove il numero di miglia calcolate è dato dalla somma delle miglia per ogni singolo volo moltiplicate per il numero di posti della relativa prenotazione, effettuati dopo la data di affiliazione. Queste miglia raddoppiano se la prenotazione include anche un hotel fino a 4 stelle, e triplicano se l'hotel è a 5 stelle.

Diagramma ER



Dizionario dei dati

Entità Cliente:

Attributo	Tipo	Note
cf	cf	stringa in formato CF
nome	stringa	
cognome	stringa	

Entità ClienteFF:

Attributo	Tipo	Note
codice	intero > 0	
dataAff	data	

Entità Volo:

Attributo	Tipo	Note
codice	iata_fl	codice volo
miglia	intero > 0	

V.Volo.part_arr: $\forall v, p, a, o_p, o_a$ Volo$(v) \wedge$Partenza$(v, p) \wedge ora(v, p, o_p) \wedge$Arrivo$(v, a) \wedge ora(v, a, o_a)$ $\rightarrow\ p \neq a$

V.Volo.no_overbooking: $\forall v, d$ Volo$(v) \wedge$data$(d)\ \rightarrow$ Posti_Voli.Posti_Disponibili$(v, d) \geq 0$

Entità Prenotazione:

Attributo	Tipo	Note
istante	dataora	
posti	intero > 0	
data	data	

Entità Aeroporto:

Attributo	Tipo	Note
codice	iata_ap	codice aeroporto IATA
nome	stringa	
tassaD	valuta > 0	
tassaA	valuta > 0	

Entità Velivolo:

Attributo	Tipo	Note
codice	plane_reg	codice di registrazione velivolo
tipo	stringa	
eur_mil	valuta > 0	
posti	intero > 0	

Entità Hotel:

Attributo	Tipo	Note
nome	stringa	
categoria	intero [1, 5]	
tariffa	valuta > 0	
kmCentro	intero ≥ 0	

Entità Indirizzo:

Attributo	Tipo	Note
via_pza	stringa	
civico	intero ≥ 0	0 indica SNC
cap	stringa CAP	

Entità Citta:

Attributo	Tipo	Note
nome	stringa	

Entità Stato:

Attributo	Tipo	Note
nome	stringa	

Relationship HotelPren:

Attributo	Tipo	Note
checkin	data	
checkout	data	
nstanze	intero > 0	

V.HotelPren.checkin_checkout: $\forall h, p, i, o\ Hotel(h) \wedge HotelPren(h, p) \wedge checkin(h, p, i) \wedge checkout(h, p, o) \ \rightarrow \ i \leq o$

Relationship AP_Vicino:

Attributo	Tipo	Note
km	intero > 0	

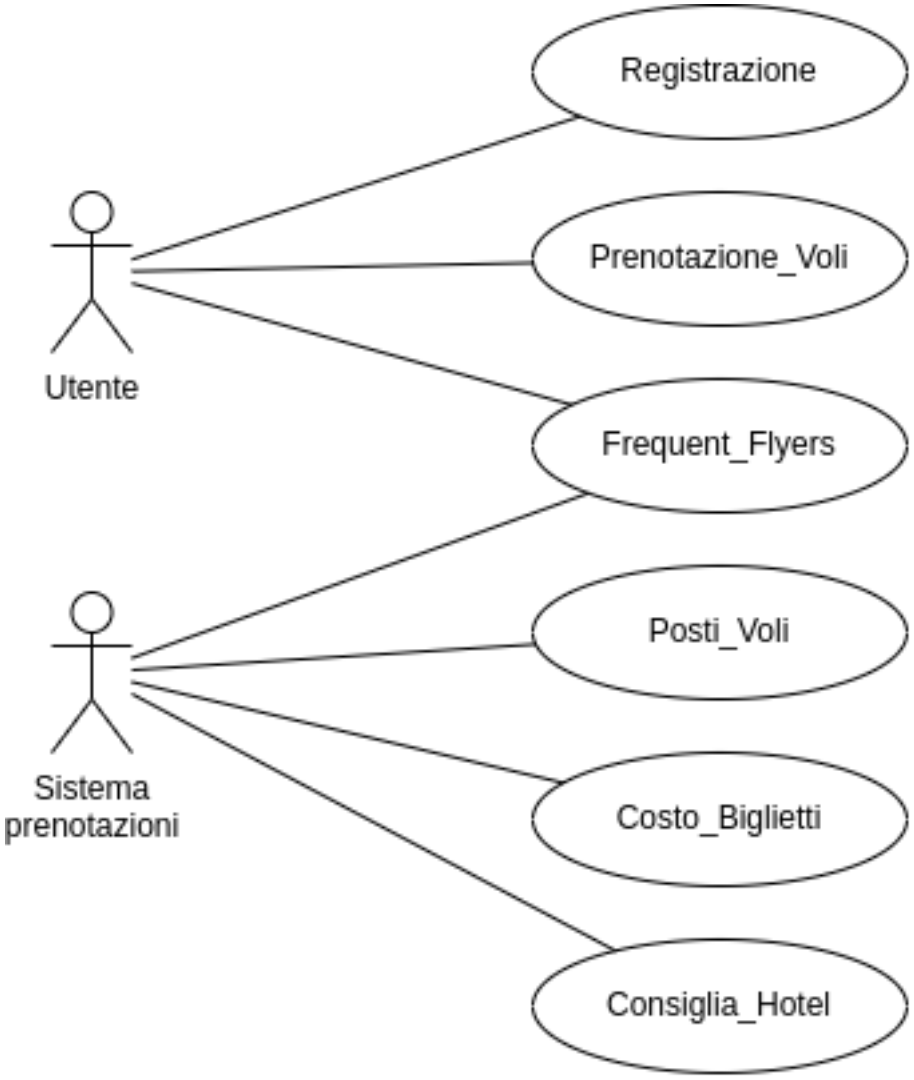
Relationship Partenza:

Attributo	Tipo	Note
ora	ora	

Relationship Arrivo:

Attributo	Tipo	Note
ora	ora	

UML


Registrazione
Prenotazione_Voli
Frequent_Flyers
Posti_Voli
Costo_Biglietti
Consiglia_Hotel
Utente
Sistema
prenotazioni

Specifiche degli use-case

5.1 Specifiche use-case Registrazione

Registra(cf: cf, nome: stringa, cognome: stringa, residenza: Indirizzo) : Cliente
- precondizioni: $\neg\exists c Cliente(c) \wedge cf(c, cf)$
- postcondizioni:
  - modifica al livello estensionale dei dati:
    - nuovi elementi del dominio di interpretazione: c
    - nuove ennuple di predicati:
      - Cliente(c)
      - cf(c, cf)
      - nome(c, nome)
      - cognome(c, cognome)
      - Risiede(c, residenza)
  - valore di ritorno: result = c

5.2 Specifiche use-case Prenotazione_Voli

Prenota(c: Cliente, v: volo, nposti: intero > 0, giorno: data) : Prenotazione
- precondizioni: $"Posti\_Voli".Posti\_Disponibili(v, giorno) \geq nposti$
- postcondizioni:
  - modifica al livello estensionale dei dati:
    - nuovi elementi del dominio di interpretazione: p
    - nuove ennuple di predicati:
      - Prenotazione(p)
      - istante(p, *adesso*)
      - posti(p, nposti)
      - data(p, giorno)
      - Effettua(c, p)
      - VoloPren(v, p)
  - valore di ritorno: result = p

Annulla_Pren(p: Prenotazione)
precondizioni: $\exists d \text{data}(p, d) \wedge d < oggi$
postcondizioni:
modifica al livello estensionale dei dati:
elementi del dominio di interpretazione che non esistono più: p
ennuple di predicati non più valide:
Prenotazione(p)
valore di ritorno: nessuno

5.3 Specifiche use-case Frequent_Flyers

Affilia(c: Cliente, cod: intero > 0) : ClienteFF
precondizioni: $\neg ClienteFF(c)$
postcondizioni:
modifica al livello estensionale dei dati:
dominio di interpretazione: invariato
nuove ennuple di predicati:
ClienteFF(c)
codice(c, cod)
dataAff(c, *oggi*)
valore di ritorno: result = c

Miglia(c: ClienteFF) : intero ≥ 0
precondizioni: nessuna
postcondizioni:
modifica al livello estensionale dei dati: nessuna
valore di ritorno:
Sia $P = \{(p, n) | Prenotazione(p) \wedge Effettua(c, p) \wedge posti(p, n)$
$\wedge [\exists d, d_a \text{ ata}(p, d) \wedge dataAff(c, d_a) \wedge d \geq d_a]\}$
Sia $P_h = \{(p, n) \in P | \exists h \text{HotelPren}(h, p)\}$
Siano $P_5 = \{(p, n) \in P_h | \exists h \text{HotelPren}(h, p) \wedge \text{categoria}(h, 5)\}$,
$P_4 = P_h \setminus P_5$ e $P_{nh} = P \setminus P_h$
Sia $V_{nh} = \{(v, m, p, n) | \text{Volo}(v) \wedge \text{miglia}(v, m) \wedge (p, n) \in P \wedge \text{VoloPren}(v, p)]\}$
Sia $V_4 = \{(v, m, p, n) | \text{Volo}(v) \wedge \text{miglia}(v, m) \wedge (p, n) \in P_4 \wedge \text{VoloPren}(v, p)]\}$
Sia $V_5 = \{(v, m, p, n) | \text{Volo}(v) \wedge \text{miglia}(v, m) \wedge (p, n) \in P_5 \wedge \text{VoloPren}(v, p)]\}$
result $= \Big(\sum_{(v,m,p,n) \in V_{nh}} m * n\Big) + 2 * \Big(\sum_{(v,m,p,n) \in V_4} m * n\Big) + 3 * \Big(\sum_{(v,m,p,n) \in V_5} m * n\Big)$

5.4 Specifiche use-case Posti_Voli

Posti_Disponibili(v: Volo, d: data) : intero ≥ 0
precondizioni: nessuna
postcondizioni:
modifica al livello estensionale dei dati: nessuna
valore di ritorno:
Sia $P = \{(p,n)|Prenotazione(p) \wedge VoloPren(v,p) \wedge posti(p,n)\}$
Sia $a|Velivolo(a) \wedge Utilizza(v,a)$ e sia $max|$"$intero > 0$"$(max) \wedge posti(a, max)$
result $= max - \sum_{(p,n)\in P} n$

5.5 Specifiche use-case Costo_Biglietti

Costo(v: Volo, d: data) : valuta > 0
precondizioni: nessuna
postcondizioni:
modifica al livello estensionale dei dati: nessuna
valore di ritorno:
Siano $A_d|$Partenza(v, A_d) e $A_a|$Arrivo(v, A_a)
Siano $t_d|$"$valuta > 0$"$(t_d) \wedge$tassaD(A_d, t_d) e $t_a|$"$valuta > 0$"$(t_a) \wedge$tassaA(A_a, t_a)
Siano $mil|$"$intero > 0$"$(mil) \wedge$ miglia(v, mil),
$eur\_mil|$"$intero > 0$"$(eur\_mil) \wedge eur\_mil(v, eur\_mil)$
Siano $A|$Velivolo$(A) \wedge$Utilizza(v, A) e $posti|$"$intero > 0$"$(posti) \wedge$posti$(A, posti)$
Sia $base = \left(\frac{mil \times eur\_mil + t_d + t_a}{posti}\right) \times 1.2$
Sia $disp =$ "$Posti\_Voli$".Posti_Disponibili(v, d)
Sia $disp\_coeff = I(disp \geq posti/2) \times 0.8 \times disp +$
$+ I(disp < posti/2) \times 1.2 \times disp$
result $= base \times disp\_coeff$

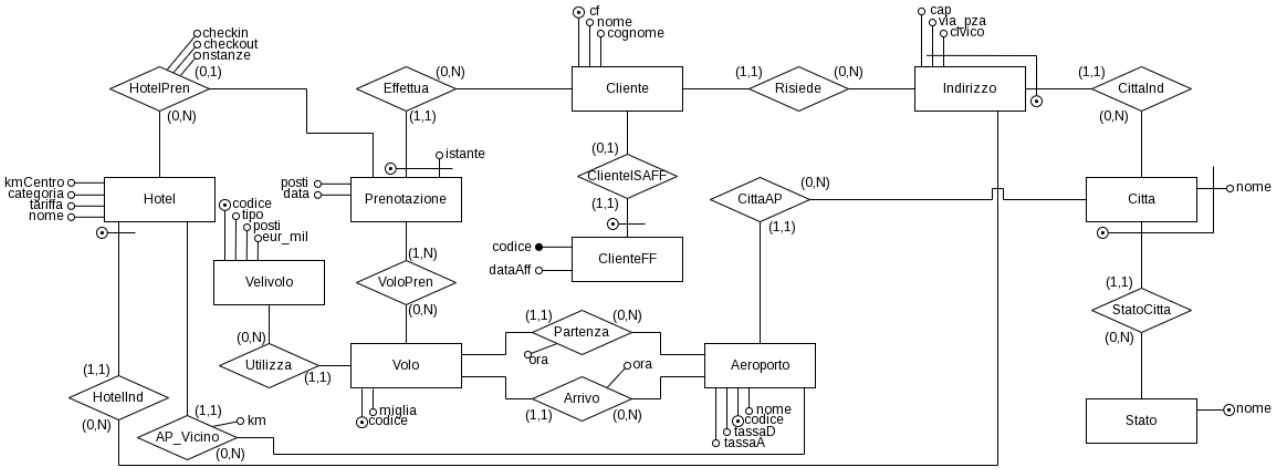
5.6 Specifiche use-case Consiglia_Hotel

Consiglia_Hotel(c: Citta, max: valuta > 0) : Hotel(1,N)
precondizioni: $\exists h, i$ Hotel$(h) \wedge$ HotelInd$(h, i) \wedge$ CittaInd(c, i)
postcondizioni:
modifica al livello estensionale dei dati: nessuna
valore di ritorno:
Sia $H = \{(h,t,d,cat)|$Hotel$(h) \wedge$ tariffa$(h,t) \wedge$ kmCentro$(h,d) \wedge$
categoria$(h, cat) \wedge t \leq max \wedge [\exists i$ HotelInd$(h, i) \wedge$ CittaInd$(c, i)]\}$
Sia $(A, t_A, d_A, cat_A) \in H|\neg\exists(h,t,d,cat) \in H[d < d_A]$
Sia $h_{same} = \{(h,t,d,cat)$ in $H|cat = cat_A \wedge d \leq d_A \times 1.1\}$
Sia $h_{better} = \{(h,t,d,cat)$ in $H|cat > cat_A \wedge d \leq d_A \times 1.2\}$
result $= \{A\} \cup h_{same} \cup h_{better}$

Scelta del DBMS

Il database verrà implementato sul DBMS PostGreSQL.

Ristrutturazione del diagramma ER



Transizione dei tipi di dati concettuali in tipi standard SQL

Entità Cliente:

Attributo	Tipo	Note
cf	cf_t	stringa in formato CF
nome	varchar(30)	
cognome	varchar(30)	

Entità ClienteFF:

Attributo	Tipo	Note
codice	int_pos	
dataAff	date	

Entità Volo:

Attributo	Tipo	Note
codice	iata_fl	
miglia	int_pos	

V.Volo.part_arr: $\forall v, p, a, o_p, o_a$ Volo$(v) \wedge$Partenza$(v, p) \wedge ora(v, p, o_p) \wedge$Arrivo$(v, a) \wedge ora(v, a, o_a)$ $\rightarrow\ p \neq a$

V.Volo.no_overbooking: $\forall v, d$ Volo$(v) \wedge$date$(d)\ \rightarrow$ Posti_Voli.Posti_Disponibili$(v, d) \geq 0$

Entità Prenotazione:

Attributo	Tipo	Note
istante	datetime	
posti	int_pos	
data	date	

Entità Aeroporto:

Attributo	Tipo	Note
codice	iata_ap	
nome	varchar(50)	
tassaD	money_pos	
tassaA	money_pos	

Entità Velivolo:

Attributo	Tipo	Note
codice	plane_reg	
tipo	varchar(50)	
eur_mil	money_pos	
posti	int_pos	

Entità Hotel:

Attributo	Tipo	Note
nome	varchar(50)	
categoria	cat_t	
tariffa	money_pos	
kmCentro	int_nonneg	

Entità Indirizzo:

Attributo	Tipo	Note
via_pza	varchar(100)	
civico	int_nonneg	0 indica SNC
cap	cap_t	

Entità Citta:

Attributo	Tipo	Note
nome	varchar(30)	

Entità Stato:

Attributo	Tipo	Note
nome	varchar(30)	

Relationship HotelPren:

Attributo	Tipo	Note
checkin	date	
checkout	date	
nstanze	int_pos	

V.HotelPren.checkin_checkout: $\forall h, p, i, o\ Hotel(h) \wedge HotelPren(h, p) \wedge checkin(h, p, i) \wedge checkout(h, p, o) \rightarrow i < o$

Relationship AP_Vicino:

Attributo	Tipo	Note
km	int_pos	

Relationship Partenza:

Attributo	Tipo	Note
ora	time	

Relationship Arrivo:

Attributo	Tipo	Note
ora	time	

8.1 Tipi di dati personalizzati

```
create domain int_pos as integer check value > 0;
create domain int_nonneg as integer check value ≥ 0;
create domain money_pos as money check value > 0;
create domain cf_t as char(16)
     check value ~* '^[a-z]{6}[0-9]{2}[a-z][0-9]{2}[a-z][0-9]{3}[a-z]$';
create domain iata_fl as varchar(4) check value ~* '^[a-z]{2,3}[0-9]{2,4}$';
create domain iata_ap as char(3) check value ~* '^[a-z]{3}$';
create domain plane_reg as varchar(5) check value ~* '^[a-z]{1,2}\-[a-z0-9]{4,}$';
create domain cap_t as char(5) check value ~* '^[0-9]{5}$';
```

Schema concettuale

Cliente(cf: cf_t, nome: varchar(30), cognome: varchar(30), via_pza: varchar(100),
civico: int_nonneg, citta: varchar(30), stato: varchar(30))
Vincolo foreign key: (via_pza, civico, citta, stato) references
Indirizzo(via_pza, civico, citta, stato)

ClienteFF(cf: cf_t, codice: int_pos, dataAff: date)
Vincolo foreign key: cf references Cliente(cf)
Vincolo chiave: codice
Vincolo dominio: dataAff $\leq$ CURRENT_DATE

Indirizzo(via_pza: varchar(100), civico: int_nonneg, citta: varchar(30), stato: varchar(30),
cap: cap_t)
Vincolo foreign key: (citta, stato) references Citta(nome, stato)

Citta(nome: varchar(30), stato: varchar(30))
Vincolo foreign key: stato references Stato(nome)

Stato(nome: varchar(30))

Volo(codice: iata_fl, miglia: int_pos, velivolo: plane_reg, apPart: iata_ap, oraPart: ora, apArr: iata_ap, oraArr: ora)
Vincolo foreign key: velivolo references Velivolo(codice)
Vincolo foreign key: apPart references Aeroporto(codice)
Vincolo foreign key: apArr references Aeroporto(codice)
Vincolo ennupla: oraPart $<$ oraArr
Vincolo ennupla: apPart $\neq$ apArr

Velivolo(codice: plane_reg, tipo: varchar(50), eur_mil: money_pos, posti: int_pos)

Prenotazione(cliente: cf_t, istante: datetime, posti: int_pos, data: date)
Vincolo foreign key: cliente references Cliente(cf)
Vincolo inclusione: (cliente, istante) $\subseteq$ VoloPren(cpren, ipren)
Vincolo dominio: istante $\leq$ CURRENT_TIMESTAMP

VoloPren(cpren: cf_t, ipren: datetime, volo: iata_fl)
Vincolo foreign key: (cpren, ipren) references Prenotazione(cliente, istante)
Vincolo foreign key: volo references Volo(codice)

Aeroporto(codice: iata_ap, nome: varchar(50), tassaD: money_pos, tassaA: money_pos,
citta: varchar(30), stato: varchar(30))
Vincolo foreign key: (citta, stato) references Citta(nome, stato)

Hotel(via_pza: varchar(100), civico: int_nonneg, citta: varchar(30), stato: varchar(30),
nome: varchar(50), categoria: cat_t, tariffa: money_pos, kmCentro: int_nonneg)
Vincolo foreign key: (via_pza, civico, citta, stato) references
Indirizzo(via_pza, civico, citta, stato)
Vincolo foreign key: (via_pza, civico, citta, stato) references
AP_Vicino(via_pza, civico, citta, stato)

AP_Vicino(via_pza: varchar(100), civico: int_nonneg, citta: varchar(30), stato: varchar(30),
ap: iata_ap, km: int_pos)
Vincolo foreign key: (via_pza, civico, citta, stato) references
Hotel(via_pza, civico, citta, stato)
Vincolo foreign key: ap references Aeroporto(codice)

HotelPren(cliente: cf_t, istante: datetime, via_pza: varchar(100), civico: int_nonneg,
citta: varchar(30), stato: varchar(30), checkIn: date, checkOut: date,
nstanze: int_pos)
Vincolo foreign key: (via_pza, civico, citta, stato) references
Hotel(via_pza, civico, citta, stato)
Vincolo chiave: (via_pza, civico, citta, stato)
Vincolo foreign key: (cliente, istante) references Prenotazione(cliente, istante)
Vincolo ennupla: checkIn $<$ checkOut

Progettazione dei vincoli esterni

10.1 Trigger per V.Volo.no_overbooking

- Operazioni:
  - inserimento, modifica in VoloPren
  - modifica in Volo
  - modifica in Velivolo
- Istante di invocazione: prima dell'operazione intercettata
- Funzione:
  1. Sia error = FALSE;
  2. Sia new l'ennupla che si sta inserendo oppure il risultato della modifica;
  3. Sia old il valore dell'ennupla prima della modifica;
  4. Se l'operazione ha agito su VoloPren:
     4.1. d := select data from Prenotazione where (cliente, istante) = (new.cpren, new.ipren);
     4.2. error := check (Posti_Voli.Posti_Disponibili(new.volo, d) - new.posti) < 0;
  5. Se l'operazione ha agito su Volo:
     5.1. Se new.velivolo $\neq$ old.velivolo:
        5.1.1. pOld := select posti from Velivolo where codice = old.velivolo;
        5.1.2. pNew := select posti from Velivolo where codice = new.velivolo;
        5.1.3. Se pNew < pOld:
           5.1.3.1. N := sum(select p.posti from VoloPren vp, Prenotazione p where (vp.cpren, vp.ipren) = (p.cliente, p.istante) and vp.volo = old.codice);
           5.1.3.2. error := check N > pNew;
  6. Se l'operazione ha agito su Velivolo:
     6.1. Se new.posti < old.posti:
        6.1.1. N := sum(select p.posti from Volo v, VoloPren vp, Prenotazione p where vp.volo = v.codice and (vp.cpren, vp.ipren) = (p.cliente, p.istante) and v.velivolo = old.codice);
        6.1.2. error := check N > new.posti;
  7. Se error = TRUE blocca l'operazione;
  8. Altrimenti permetti l'operazione;

Specifiche realizzative degli use-case

11.1 Registrazione

Registra(cf: cf_t, nome: varchar(30), cognome: varchar(30), via_pza: varchar(100),
civico: int_nonneg, citta: varchar(30), stato: varchar(30), cap: cap_t) : cf_t
algoritmo:

```
 1 Q ← risultato della query SQL ottenuta sostituendo a :cf
       il valore dell'omonimo parametro attuale
       select exists (select 1 from Cliente where cf = :cf);

 2 if Q rappresenta un errore then

 3       inoltra l'errore;

 4 else if Q = TRUE then

 5       inoltra l'errore 'cliente già esistente';

 6 else

 7       A ← risultato della query SQL ottenuta sostituendo :via_pza, :civico,
             :citta, :stato, :cap il valore degli omonimi parametri attuali
             select exists (select 1 from Indirizzo where (via_pza, civico, citta,
                 stato, cap) = (:via_pza, :civico, :citta, :stato, :cap);

 8       if A rappresenta un errore then

 9           inoltra l'errore;

10       else if A = FALSE then

11           IAddr ← risultato della query SQL ottenuta sostituendo :via_pza,
                 :civico, :citta, :stato, :cap il valore degli omonimi
                 parametri attuali
                 insert into Indirizzo values (:via_pza, :civico, :citta,
                     :stato, :cap);

12           if IAddr rappresenta un errore then

13               inoltra l'errore;
```

```
14      ICl ← risultato della query SQL ottenuta sostituendo a :cf,
            :nome, :cognome, :via_pza, :civico, :citta, :stato il valore
            degli omonimi parametri attuali
            insert into Clienti values (:cf, :nome, :cognome, :via_pza,
                :civico, :citta, :stato);

15      if ICl rappresenta un errore then

16          inoltra l'errore;

17      else return :cf;
```

11.2 Prenotazione_Voli

Prenota(cliente: cf_t, v: iata_fl, nposti: int_pos, giorno: date) : (cf_t, datetime)
algoritmo:

```
1 disp ← Posti_Voli.Posti_Disponibili(v, giorno);

2 if nposti > disp then

3       inoltra l'errore 'posti non disponibili';

4 else

5       ist ← CURRENT_TIMESTAMP;

6       I ← risultato della transazione SQL ottenuta sostituendo :cliente, :posti,
            :giorno, :volo con gli omonimi parametri attuali
            begin transaction;
            insert into Prenotazione values(:cliente, ist, :posti, :giorno);
            insert into VoloPren values(:cliente, ist, :volo);
            commit work;

7       if I rappresenta un errore then

8           inoltra l'errore;

9       else return (:cliente, ist);
```

Annulla_Pren(cliente: cf_t, ist: datetime)
algoritmo:

```
1 Q ← risultato della query SQL ottenuta sostituendo :cliente, :ist,
            con gli omonimi parametri attuali
            select exists(select 1 from Prenotazione where (cliente, istante)
  = (:cliente, :ist);

2 if Q rappresenta un errore then

3       inoltra l'errore;

4 else if Q = FALSE then

5       inoltra l'errore 'prenotazione non trovata';
```

```
6 else
7       D ← risultato della transazione SQL ottenuta sostituendo :cliente, :ist,
            con gli omonimi parametri attuali
            begin transaction;
            delete from VoloPren where (cpren, ipren) = (:cliente, :ist);
            delete from Prenotazione where (cliente, istante) = (:cliente, :ist);
            commit work;
8       if D rappresenta un errore then
9           inoltra l'errore;
10      else return;
```

11.3 Frequent_Flyers

Affilia(c: cf_t, cod: int_pos) : cf_t
algoritmo:

```
1 Q ← risultato della query SQL ottenuta sostituendo a :cf
      il valore dell'omonimo parametro attuale
      select exists (select 1 from Cliente where cf = :cf);
2 if Q rappresenta un errore then
3       inoltra l'errore;
4 else if Q = FALSE then
5       inoltra l'errore 'cliente insesistente';
6 else
7       I ← risultato della query SQL ottenuta sostituendo a :cf, :cod
            i valori degli omonimi parametri attuali
            insert into ClienteFF values (:cf, :cod, CURRENT_DATE);
8       if I rappresenta un errore then
9           inoltra l'errore;
10      else return :cf;
```

Miglia(cf: cf_t) : int_pos
algoritmo:

```
1 Q ← risultato della query SQL ottenuta sostituendo a :cf
      il valore dell'omonimo parametro attuale
      select exists (select 1 from ClienteFF where cf = :cf);
2 if Q rappresenta un errore then
3       inoltra l'errore;
4 else if Q = FALSE then
```

```
5       inoltra l'errore 'cliente insesistente o non affiliato';
6  else
7       Vnh ← risultato della query SQL ottenuta sostituendo a :cf
             il valore dell'omonimo parametro attuale
             select sum(v.miglia*p.posti) from Volo v, VoloPren vp, Prenotazione
  p
                  where v.codice = vp.volo
                  and (vp.cpren, vp.ipren) = (p.cliente, p.istante)
                  and p.cliente = :cf and (p.cliente, p.istante) not in
                       (select cpren, ipren from HotelPren);
8       if Vnh rappresenta un errore then
9            inoltra l'errore;
10      else
11           Vh4 ← risultato della query SQL ottenuta sostituendo a :cf
                  il valore dell'omonimo parametro attuale
                  select sum(v.miglia*p.posti) from Volo v, VoloPren vp,
                       Prenotazione p, HotelPren hp, Hotel h where v.codice = vp.volo
                       and (vp.cpren, vp.ipren) = (p.cliente, p.istante)
                       and (hp.cpren, hp.ipren) = (p.cliente, p.istante)
                       (hp.via_pza, hp.civico, hp.citta, hp.stato) =
                            (h.via_pza, h.civico, h.citta, h.stato)
                       and p.cliente = :cf and h.categoria < 5;
12           if Vh4 rappresenta un errore then
13                inoltra l'errore;
14           else
15                Vh5 ← risultato della query SQL ottenuta sostituendo a :cf
                       il valore dell'omonimo parametro attuale
                       select sum(v.miglia*p.posti) from Volo v, VoloPren vp,
                            Prenotazione p, HotelPren hp, Hotel h
                            where v.codice = vp.volo
                            and (vp.cpren, vp.ipren) = (p.cliente, p.istante)
                            and (hp.cpren, hp.ipren) = (p.cliente, p.istante)
                            (hp.via_pza, hp.civico, hp.citta, hp.stato) =
                                 (h.via_pza, h.civico, h.citta, h.stato)
                            and p.cliente = :cf and h.categoria = 5;
16                if Vh5 rappresenta un errore then
17                     inoltra l'errore;
18                else return Vnh + 2*Vh4 + 3*Vh5;
```

11.4 Posti_Voli

Posti_Disponibili(v: iata_fl, d: date) : int_nonneg
algoritmo:

```
1 N ← risultato della query SQL ottenuta sostituendo a :v e :d
       i valori degli omonimi parametri attuali
       select a.posti as maxp sum(p.posti) as occp from Prenotazione p,
       VoloPren vp, Volo v, Velivolo a
       where (p.cliente, p.istante) = (vp.cpren, vp.ipren) and vp.volo = volo.codice
       and volo.codice and volo.velivolo = a.codice and v.codice = :v
       and p.data = :d group by (a.posti);

2 if N rappresenta un errore then

3       inoltra l'errore;

4 else return N.maxp - N.occp;
```

11.5 Costo_Biglietti

Costo(v: iata_fl, d: date) : money_pos
algoritmo:

```
1 N ← risultato della query SQL ottenuta sostituendo a :v e :d
       i valori degli omonimi parametri attuali
       select apD.tassaD, apA.tassaA, v.miglia, v.eur_mil, a.posti
       from Aeroporto apD, Aeroporto apA, Volo v, Velivolo a
       where apD.codice = v.apPart and apA.codice = v.apArr
       and v.velivolo = a.codice;

2 if N rappresenta un errore then

3       inoltra l'errore;

4 else

5       base ← (N.miglia*N.eur_mil + N.tassaD + N.tassaA)/N.posti * 1.2;

6       disp ← Posti_Voli.Posti_Disponibili(:v, :d);

7       disp_coeff ← (disp ≥ N.posti/2) ?  0.8 * disp :  1.2 * disp;

8       return base * disp_coeff;
```

11.6 Consiglia_Hotel

Consiglia_Hotel(c: varchar(30), s: varchar(30), maxt: money_pos) :
(varchar(100), int_nonneg, varchar(30), varchar(30))(1,N)
algoritmo:

```
1  H ← risultato della query SQL ottenuta sostituendo a :c, :s, :maxt
       i valori degli omonimi parametri attuali
       select via_pza, civico, citta, stato, categoria, kmCentro from Hotel
       where (citta, stato) = (:c, :s) and tariffa ≤ :maxt
       order by kmCentro asc;
2  if H rappresenta un errore then
3       inoltra l'errore;
4  else
5       cons ← {H[0]};
6       for i ← 1 to length(H-1) do
7            if (H[i].categoria = H[0].categoria
8                    and H[i].kmCentro ≤ H[0].kmCentro*1.1)
9                    or (H[i].categoria > H[0].categoria
10                   and H[i].kmCentro ≤ H[0].kmCentro*1.2) then
11                cons ← append(cons, H[i]);
12      done;
13      return cons;
```