



## 2

## Specifiche dei Requisiti

I dati di interesse per il sistema sono le crociere offerte dall'agenzia con le relative prenotazioni e le destinazioni in catalogo.

Il sistema deve essere in grado di rappresentare le crocieri offerte dall'agenzia, con codice, date di inizio e fine, e la nave utilizzata. Delle navi che hanno un nome (ad es. "LoveBoat"), interessa il grado di comfort, espresso in un numero di stelle che può variare da 3 a 5, e il numero massimo di passeggeri che possono ospitare.

Ciascuna crociera consta di un itinerario caratterizzato da un nome (ad es. "Panorami d'Oriente") il quale prevede una sequenza –ordinata– di destinazioni. Di queste interessa il nome e il continente in cui si trovano. Gli itinerari fissano, oltre che l'ordine delle destinazioni da visitare, anche la relativa data ed ora di arrivo e di partenza. Dato che, in generale, un itinerario può essere previsto da più di una crociera, le date di arrivo e partenza relative ad una destinazione vengono espresse come differenze rispetto la data di inizio della crociera stessa (ad es., l'itinerario "Panorami d'Oriente" prevede di raggiungere la destinazione  $x$  alle 16:00 del quinto giorno di crociera, e di ripartire alle 12:00 del giorno successivo, il sesto).

Inoltre, le destinazioni sono caratterizzate da un insieme di posti da vedere durante eventuali escursioni organizzate. Questi ultimi sono caratterizzati dal nome, dalla descrizione, e dalla fascia oraria consigliata per le visite. Il sistema deve permettere di risalire ai posti da vedere in ogni singola destinazione.

L'agenzia classifica le crocieri in crocieri di luna di miele e crocieri per famiglia (di queste ultime interessa conoscere se sono adatte o meno ai bambini), e le destinazioni in romantiche e divertenti. Si noti che possono esistere destinazioni che sono sia romantiche che divertenti. Per venire incontro alle nuove tendenze delle giovani coppie, le crocieri di luna di miele vengono ulteriormente classificate in tradizionali e alternative: sono definite tradizionali quelle che prevedono un numero di destinazioni romantiche maggiore o uguale al numero di destinazioni divertenti, alternative le altre.

Infine, il sistema deve anche permettere di gestire le prenotazioni di crocieri effettuate



dai clienti. In particolare, del clienti interessa nome, cognome, età ed indirizzo, mentre delle prenotazioni interessa l'istante di prenotazione, la crociera ed il numero di posti prenotati.

Le funzionalità richieste al sistema sono le seguenti:

1. Dato un cliente che desidera prenotare un certo numero di posti per una crociera  $c$ , il personale dell'Ufficio Prenotazioni deve poter effettuare la relativa prenotazione. La richiesta di prenotazione deve essere rifiutata nel caso il numero di posti disponibili, all'istante corrente, per la crociera  $c$  non sia sufficiente.
2. Dato un insieme di clienti, l'Ufficio Marketing deve poter calcolare l'età media di quelli che hanno prenotato almeno una crociera che prevede una destinazione esotica (ovvero che si trova in un continente diverso dall'Europa).
3. Dato un insieme di destinazioni, l'Ufficio Marketing deve poter calcolare la percentuale di quelle *gettonate*. Una destinazione si dice gettonata se è stata raggiunta da almeno dieci crociere di luna di miele, oppure da almeno quindici crociere per famiglie nel corso degli ultimi due anni.

## 1) Crociera

- codice
- data inizio/fine
- nave utilizzata(vedi 2)
- linea di miele
- { + tradizionali }  
{ + alternative } tipo
- croc per famiglie  
+ adatte ai bambini

## 3) Itinerario

- home
- prevede una sequenza di destinazioni(vedi 4)
- data arrivo/partenza
- può essere previsto da più di una crociera

## 2) Nave

- home
- grado di confort / [3,5]
- max num. passeggeri
- segue un itinerario(vedi 3)

## 7) Tappa

- home
- desenzione
- fascia oraria

## 4) Destinazione

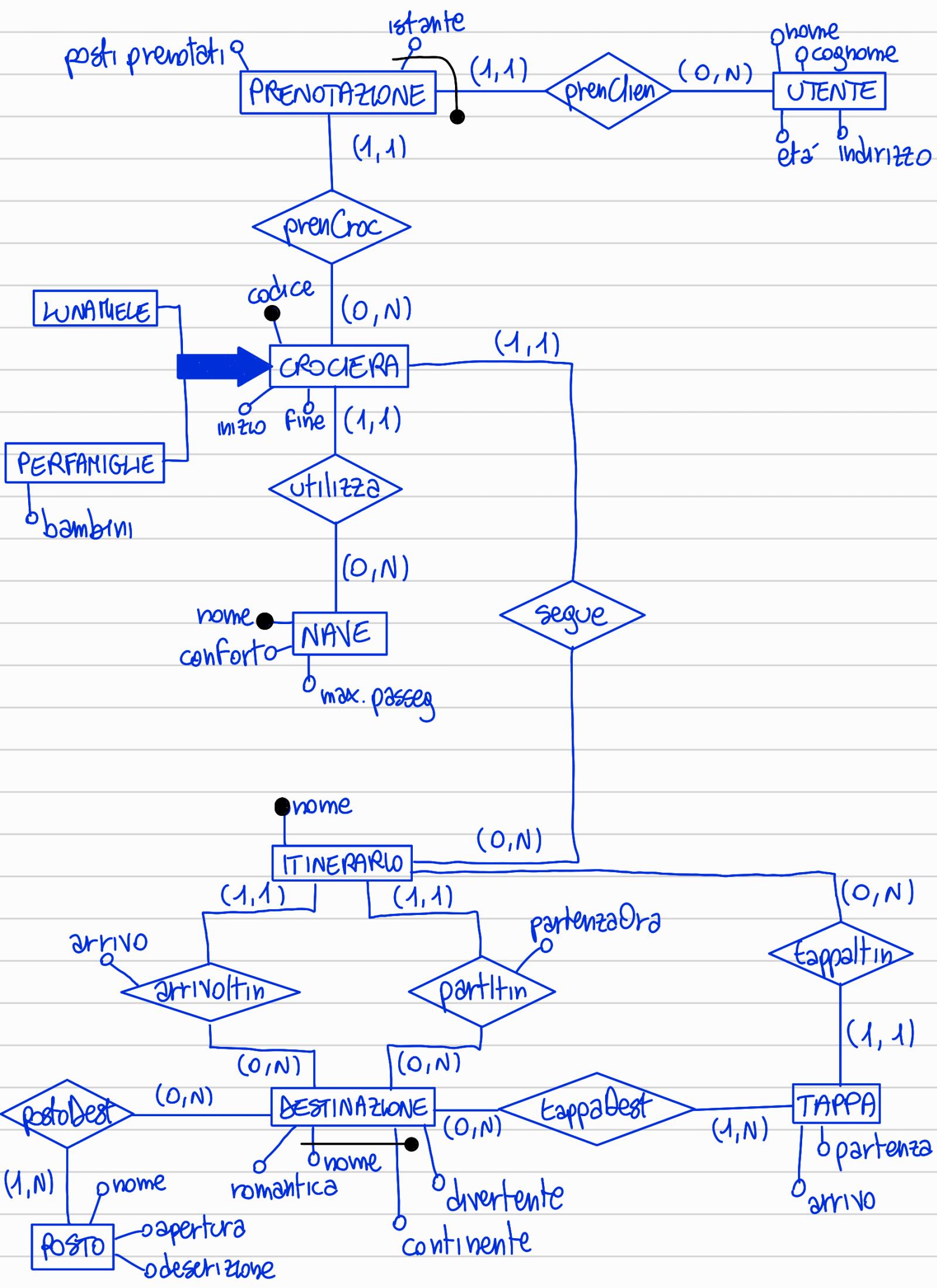
- home
- continente / {EU, AN, AS, AF, OC}
- hanno un insieme di posti da vedere(vedi 7)
- romantiche [ possono 3 destina. sia rom sia div ]
- divertenti

## 5) Prenotazioni

- fatte da utenti(vedi 6)
- istante di prenotazione
- la crociera(vedi 1)
- num. posti prenotati

## 6) Utente

- nome
- cognome
- età
- indirizzo



# Dizionario

## Cliente

indirizzo /INDRIZZO

nascita /data

nome /str

cognome /str

## CroceriaFarm

bambini /boolean

## Posto

nome /str

deserzione /str

apertura /FaseOraria

## Destinazione

divertente /boolean

romantica /boolean

continente /{E, AS, AM, AMN, AF, O}

nome /str

## Crociata

inizio /data

fine /data

codice /str

<\.Crociata .posti >

$\forall c \text{ Crociata}(c) \rightarrow \text{GestionePrenotazioni}.\text{postiDisponibili}(cr) \geq 0$

## Prenotazione

istante / dataora

num. passeggeri / int > 0

< \forall. Prenotazione. partenza >

\forall p, i, in, c

Prenotazione(p)  $\wedge$  istante(p, i)  $\wedge$  prenCroc(p, c)  $\wedge$

$\wedge$  inizio(c, in)  $\rightarrow$  i < in

## Nave

max. posti / int > 0

confort / [3, 5]

nome / str

< \forall. Nave. crociera. disj >

\forall c, c', n, c-in, c'-in, dest-arr-c, dest-arr-c', i-c, i-c'  
arr-c, arr-c', arr-g-c, arr-g-c'

[Crociera(c)  $\wedge$  Crociera c'  $\wedge$  c  $\neq$  c'  $\wedge$  utilizza(n, c)  $\wedge$

$\wedge$  utilizza(n, c')  $\wedge$  inizio(c, c-in)  $\wedge$  inizio(c', c'-in)  $\wedge$

segue(c, i-c)  $\wedge$  segue(c, i-c')  $\wedge$  arrivoltin(dest-arr-c, i-c)  $\wedge$

$\wedge$  arrivoltin(dest-arr-c', i-c')  $\wedge$  arrivo(dest-arr-c, i-c, arr-c)  $\wedge$

$\wedge$  arrivo(dest-arr-c', i-c', arr-c')  $\wedge$  giorni(arr-c, arr-g-c)  $\wedge$

giorni(arr-c', arr-g-c')]  $\rightarrow$  [c-in > c-in + arr-g-c  $\vee$   
c-in > c-in + arr-g-c']

## Itinerario

nome / str

## Tappa

partenza / DeltaDataOra  
arrivo / DeltaDataOra

< V.Tappa.date >

$\forall t, p, a$

$Tappa(t) \wedge partenza(t, p) \wedge arrivo(t, a) \rightarrow (a < p)$

< V.Tappa.succPart >

$\forall i, p, t, \alpha_t, \sigma_t, \sigma_p$

Itinerario(i)  $\wedge$  partitin(p, i)  $\wedge$  partenzaOra(p, i,  $\sigma_p$ )  $\wedge$   
 $Tappa(t) \wedge tappaltin(t, i) \wedge arrivo(t, \alpha_t) \wedge giorno(\alpha_t, 1) \wedge$   
 $\wedge giorno(\alpha_t, \sigma_t) \rightarrow \sigma_p < \sigma_t$

< V.Tappa.primaArrivo >

$\forall i, dest\_arr, arr, t\_part, t$

Itinerario(i)  $\wedge$  arrivoltin(i, dest\_arr)  $\wedge$  arrivo(i, dest\_arr, arr)  $\wedge$   
 $Tappa(t) \wedge tappaltin(t, i) \wedge partenza(t, t\_part) \rightarrow t\_part < arr$

partitin

partenzaOra / ora

arrivoltin

arrivo / DeltaDataOra

tappaltin

< V.tappaltin.soste.disj >

$\forall i, t, t', t\_part, t'\_part, t\_arr, t'\_arr$

$\text{Itinerario}(i) \wedge \text{Tappa}(t) \wedge \text{tappaltn}(t, i) \wedge \text{arrivo}(t, t\_arr) \wedge$   
 $\text{partenza}(t, t\_part) \wedge \text{Tappa}(t') \wedge \text{tappaltn}(t', i) \wedge$   
 $\wedge \text{arrivo}(t', t'\_arr) \wedge \text{partenza}(t, t'\_part) \wedge t \neq t'$   
 $\rightarrow (t\_arr > t'\_arr) \vee (t'\_part > t\_part)$

INDRIZZO:

via / str

civico / int  $\geq 0$  (0, 1)

CAP / str di s char

città / str

regione / str

FaseOraria

da / ora

a / ora

DeltaDataOra

giorno / int  $> 0$

ora / ora

$\forall d', d'', g', o', g'', o$

$[\Delta\text{DataOra}(d') \wedge \Delta\text{DataOra}(d'') \wedge$

$\text{giorni}(d', g') \wedge \text{ore}(d', o') \wedge \text{giorni}(d'', g'') \wedge \text{ore}(d'', o'')$

$\rightarrow [(d' \leq d'') \leftrightarrow (g' < g'' \vee (g' = g'' \wedge o' \leq o''))]$

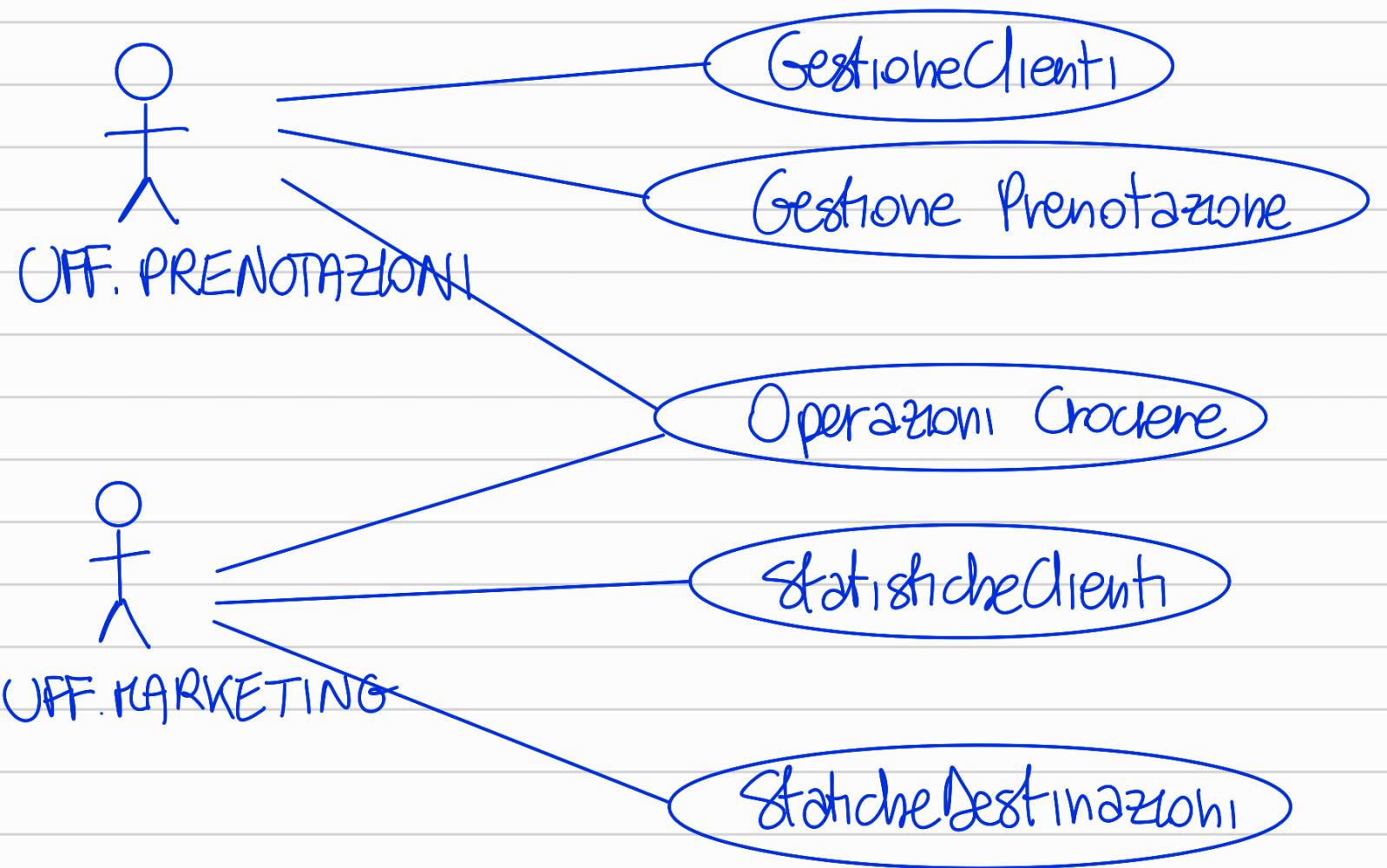
$\forall d', d'' (\Delta\text{DataOra}(d') \wedge \Delta\text{DataOra}(d''))$

$\rightarrow [(d' = d'') \leftrightarrow (d' \leq d'' \wedge d'' \leq d')]$

$\forall d', d'' (\Delta\text{DataOra}(d') \wedge \Delta\text{DataOra}(d''))$

$\rightarrow [(d' < d'') \leftrightarrow (d' \leq d'' \wedge \neg(d' = d''))]$

# DIAGRAMMA UML



## Specifiche Use-Casi: Gestione Prenotazione

prenota ( $c: \text{Cliente}$ ,  $cr: \text{Crociere}$ ,  $n.\text{pass}: \text{int} > 0$ ): Prenotazione  
pre-condizioni:  $\forall \text{in}, cr \quad \text{inizio}(\text{in}, cr) \rightarrow \text{in} < \text{adesso} \wedge$   
 $\text{Prenotazioni}.postiDisponibili(cr) \geq n.\text{pass}$

post-condizioni:

Mod. del liv. estens. dei dati: Il nuovo liv. estens....

Variaz. nel dom. di interpret: nuovo elemento  $\alpha$

Variaz. nelle ennuple di predic:  $\text{Prenotazione}(\alpha)$ ,  
 $\text{preCliente}(\alpha, \text{cl})$ ,  $\text{preCroc}(\alpha, cr)$   
 $\text{istante}(\alpha, \text{adesso})$ ,  $\text{nPass}(\alpha, \text{posti})$

Valore di ritorno:  $\text{result} = \alpha$

postiDisponibili ( $c: \text{Crociere}$ ):  $\text{int} \geq 0$

pre-condizioni: nessuna

post-condizioni:

Mod. del liv. estensionale dei dati: no

Valore di ritorno:

$$Z = \{(z, p) \mid \text{Prenotazione}(z) \wedge \text{nPass}(p, z) \wedge \\ \text{preCroc}(z, cr)\}$$

sia:  $\text{postiOcupati} = \sum_{(z, p) \in Z} p$

sia  $\text{totPosti} = \exists n \ \text{utilizza}(n, cr) \wedge \text{pax}(n, \text{totPosti})$

$\text{result} = \text{totPosti} - \text{postiOcupati}$

## Specifiche Use-Case: GestioneClienti

nuovo (no: str, na: data, co: str, ind: Indirizzo): Cliente  
pre-condizioni: nessuna  
post-condizioni:

Modifica del livello esten: Il livello estens. dei dati.....

Variazione nel dominio di interpre: nuova elemento  $\alpha$

Variazioni ennuple di predicato: Cliente( $\alpha$ ), nome( $\alpha, \text{no}$ ), cognome( $\alpha, \text{co}$ ), indirizzo( $\alpha, \text{ind}$ ), data( $\alpha, \text{na}$ ),

Val. di ritorno: result =  $\alpha$

cerca (no: str(0,1), na: data(0,1), co: str(0,1)): Cliente (0, N)

pre-condizioni: nessuna

post-condizioni:

Modifica del liv. est. dei dati: nessuna

Valore di ritorno: nasuta( $c, \text{na}$ ), nome( $c, \text{no}$ ), cognome( $c, \text{co}$ ),  
(sse. na, no e co sono definiti, altrim. true)

result =  $\{c \mid \text{Cliente}(c) \wedge \varphi_{\text{name}}(c) \wedge \varphi_{\text{cogn}}(c) \wedge \varphi_{\text{nasuta}}(c)\}$

## Specifiche Use-Case: OperazioniCrociere

tipocrociata ( $c: \text{CronacaDiViaggio}$ ): { tradizionali, alternative }

pre-cond: nessuna

post-cond:

Modifica del liv....: nessuna

Valore di ritorno:

$Dr = \{d \mid \exists i, t \quad \text{Destinazione}(d) \wedge \text{Itinerario}(i),$   
 $\text{EappaltoIn}(t, i) \wedge \text{EappaltoEst}(t, d)$   
 $\text{segue}(c, i) \wedge \text{romantica}(d, \text{true})$

$$D_a = \{ d \mid \exists i, t \text{ Destinazione}(d) \wedge \text{Itinerario}(i), \\ \text{Tappaltin}(t, i) \wedge \text{TappaDest}(t, d) \\ \text{segue}(i, c) \wedge \text{alternativa}(d, \text{true}) \}$$

$$\text{result} = \begin{cases} \text{'tradizionale'} \text{ se } |\Delta_r| \geq |\Delta_a| \\ \text{altrimenti 'alternativa'} \end{cases}$$

Specifiche Use-Case: StatisticheClienti

eta-MediaEsotica ( $c: \text{Clienti}(O, N)$ ): reale  $\geq 0$

pre-condizioni:  $C_e = \{c \in C \mid \exists p, cr, i, t, d$   
 $\text{Cliente}(c) \wedge \text{Destinazione}(d) \wedge \text{Prenotazione}(p)$   
 $\wedge \text{prencliente}(p, c) \wedge \text{premcroc}(p, cr)$   
 $\text{Crociere}(cr) \wedge \text{segue}(cr, i) \wedge$   
 $\text{Tappaltin}(i, t) \wedge \text{Tappa}(t) \wedge \text{TappaDest}(t, d)$   
 $\wedge \text{StatisticheDestinazioni.esotica}(d)\}$

Con  $|C_e| \geq 1$

post-condizioni:

Modifica del liv. estensionale dei dati: nessuna

Valore di ritorno:

$$\text{result} = \frac{\sum_{c \in C_e} \text{statclient.eta}(c)}{|C_e|}$$

eta( $c: \text{Clienti}$ ): int  $\geq 0$

pre-condizioni:  $\exists c \text{ Cliente} \wedge \text{eta}(e, c)$

post-condizioni:

Modifica del liv. ... dei dati: nessuna

Valore di ritorno:

Sia:

$naseita(c, na) \wedge \text{diff} = \text{adesso} - na \wedge \text{anno}(\text{diff}, \text{anni})$   
result = anni

Specifico Use-Case: Statistiche Destinazioni

gettonata(d: Destinazione): boolean

pre-condizioni:

post-condizioni:

Modifica del liv. estens. dei dati: nessuna

Valore di ritorno:

Sia:

$C_{LdM} = \{ cr \mid \exists cr, in, diff, anni, i, it$   
 $\text{CrocieraLdM}(cr) \wedge \text{inizio}(in, cr)$   
 $\wedge \text{diff} = \text{adesso} - in \wedge \text{anno}(\text{diff}, \text{anni}) \wedge$   
 $\wedge \text{anni} \leq 2 \wedge \text{Itinerario}(i) \wedge \text{segue}(cr, i)$   
 $\wedge \text{tappalIn}(it, E) \wedge \text{tappaDest}(E, d) \}$

$C_{FaM} = \{ cr \mid \exists cr, in, diff, anni, i, it$   
 $\text{CrocieraFaM}(cr) \wedge \text{inizio}(in, cr)$   
 $\wedge \text{diff} = \text{adesso} - in \wedge \text{anno}(\text{diff}, \text{anni}) \wedge$   
 $\wedge \text{anni} \leq 2 \wedge \text{Itinerario}(i) \wedge \text{segue}(cr, i)$   
 $\wedge \text{tappalIn}(it, E) \wedge \text{tappaDest}(E, d) \}$

result =  $\begin{cases} \text{'True'} \text{ se } |C_{LdM}| \geq 10 \vee |C_{FaM}| \geq 15, \\ \text{'False' altrimenti} \end{cases}$

percentualeGettonate (d: Destinazione(1, N)): reale in  $[0, 1]$

pre-condizioni:

post-condizioni:

Modifica del liv. estens. dei dati: nessuna

Valore di ritorno:

$$D' = \{ d \in D \mid \text{statDestinazioni.gettonata}(d) = \text{true} \}$$

$$\text{Result} = \frac{|D'|}{|D|}$$

esotica (d: Destinazione): boolean

pre-condizioni: nessuna

post-condizioni:

Modif. del liv. esten: nessuna

Valore di ritorno:

$$\text{result} = \exists \text{cont} \text{ continent}(d, \text{cont}) \wedge \text{cont} \neq \text{'EU'}$$

# PARTE P

## Progettazione della BD e delle funzionalità

### 1) Domini

```
create domain StringS as varchar(50);
```

```
create domain StringM as varchar(100);
```

```
create domain StringL as varchar(1000);
```

```
create domain TipoConfort as integer  
check(value >= 3 and value <= 5);
```

```
create domain PosInteger as integer  
check(value > 0);
```

```
create type CONTINENTE as  
enum {'E', 'AS', 'AMN', 'AMS', 'AF', 'O'}
```

```
create type DettalDataOra as (  
giorni integerGZ,  
ora time  
);
```

```
create type FasciaOraria as (  
da time,  
a time  
);
```

create domain IndirizzoVia as StringS  
check (value is not null);

create domain IndirizzoCivico as integer  
check (value > 0);

create domain IndirizzoCAP as char(5)  
check (value is not null and value ~ '^ [0-9]\*\\$');

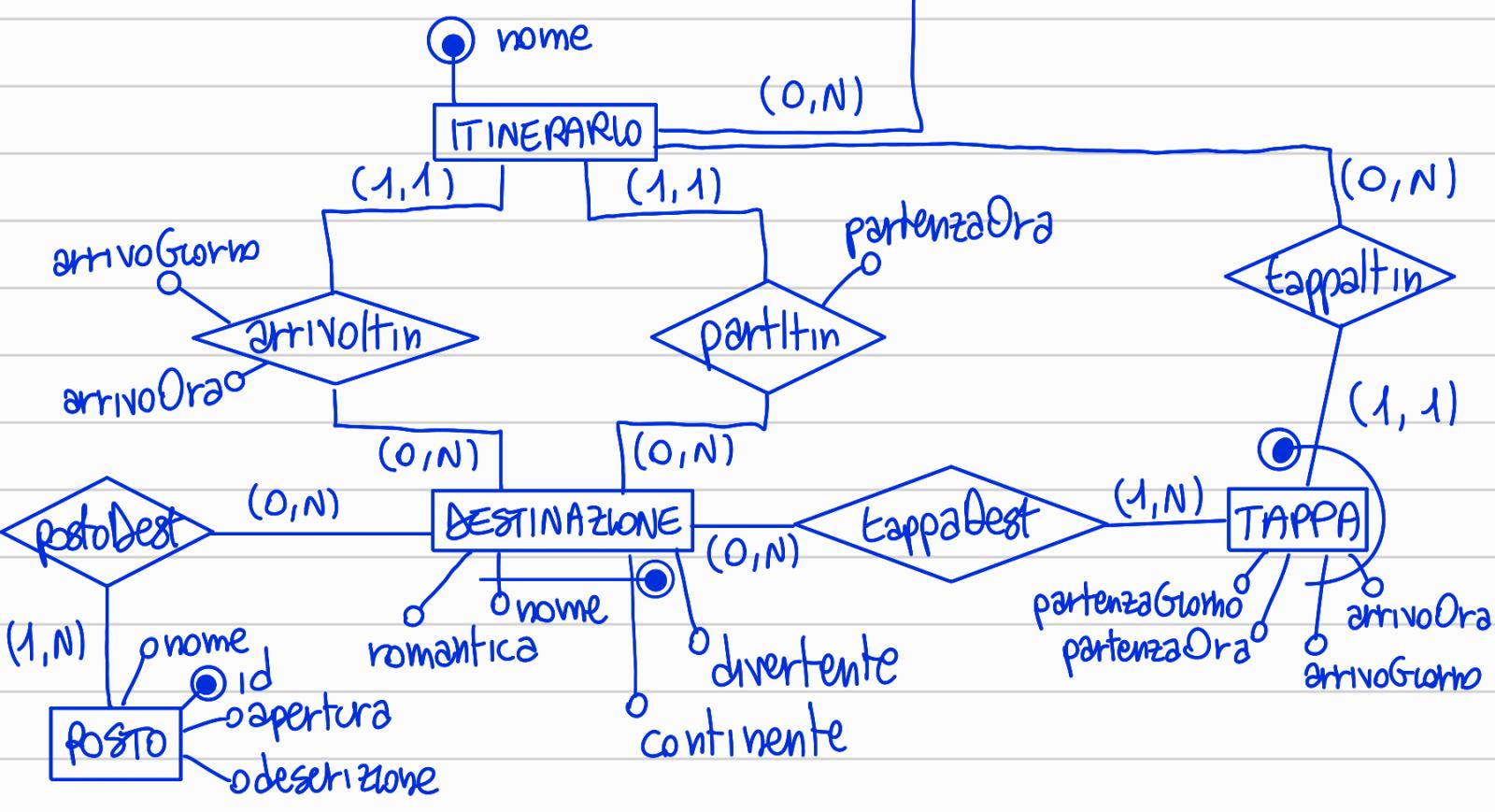
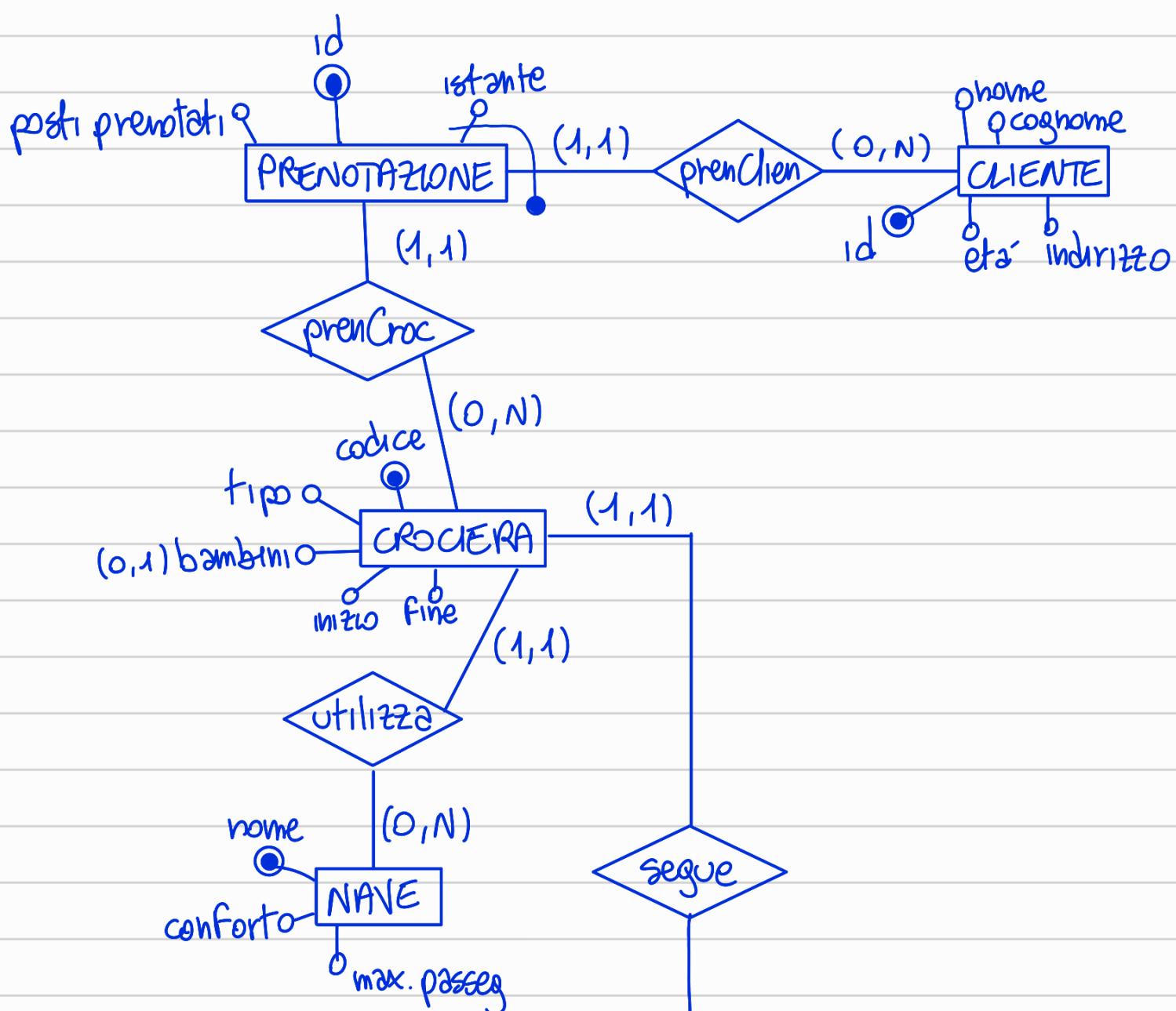
create domain IndirizzoCittà as StringM  
check (value is not null)

create domain IndirizzoNazione as StringM  
check (value is not null)

create type Indirizzo as (  
 via IndirizzoVia,  
 civico IndirizzoCivico,  
 CAP IndirizzoCAP,  
 città IndirizzoCittà,  
 nazione IndirizzoNazione  
);

create type TIPOCROCIERA as  
enum {'LunaDiMiele', 'PerFamiglia'}

2) Ristrutturazione E-R



## Ciente

indirizzo INDIRIZZO

nascita date

nome String M

cognome String M

## Posto

nome String M

descrizione String L

aperturaDa time

aperturaA time

id integer

## Destinazione

diventante boolean

romantica boolean

continentente CONTINENTE

nome String M

## Crociata

inizio date

codice String S

tipo Tipo Crociata

cFamBambini boolean

<\ V. Crociata . isz . bambini >

$\forall c \text{ Crociata}(c) \rightarrow [\text{tipo}(c, \text{PerFamiglia}) \leftrightarrow \exists b \text{ cFamBambini}(c, b)]$

<\ V. Crociata . posti >

$\forall c \text{ Crociata}(c) \rightarrow \text{GestionePrenotazioni}.\text{postiDisponibili}(c) \geq 0$

## Prenotazione

istante timestamp  
 num. passeggeri Integer GZ  
 id integer

< \forall. Prenotazione. partenza >

$\forall p, i, \text{in}, c$   
 $\text{Prenotazione}(p) \wedge \text{istante}(p, i) \wedge \text{prenCroc}(p, c) \wedge$   
 $\wedge \text{inizio}(c, \text{in}) \rightarrow i < \text{in}$

## Nave

max. posti PosInteger  
 confort TipoComfort  
 nome StringM

< \forall. Nave. crociera. disj >

$\forall c, c', n, c\_in, c'\_in, \text{dest\_arr\_c}, \text{dest\_arr\_c}', i\_c, i\_c'$   
 $\text{arr\_c}, \text{arr\_c}', \text{arr\_g\_c}, \text{arr\_g\_c}'$   
 $[\text{Crociera}(c) \wedge \text{Crociera}(c') \wedge c \neq c' \wedge \text{utilizza}(n, c) \wedge$   
 $\wedge \text{utilizza}(n, c') \wedge \text{inizio}(c, c\_in) \wedge \text{inizio}(c', c'\_in) \wedge$   
 $\text{segue}(c, i\_c) \wedge \text{segue}(c, i\_c') \wedge \text{arrivoltin}(\text{dest\_arr\_c}, i\_c) \wedge$   
 $\wedge \text{arrivoltin}(\text{dest\_arr\_c}', i\_c') \wedge \text{arrivo}(\text{dest\_arr\_c}, i\_c, \text{arr\_c}) \wedge$   
 $\wedge \text{arrivo}(\text{dest\_arr\_c}', i\_c', \text{arr\_c}') \wedge \text{giorni}(\text{arr\_c}, \text{arr\_g\_c}) \wedge$   
 $\text{giorni}(\text{arr\_c}', \text{arr\_g\_c}')] \rightarrow [c\_in' > c\_in + \text{arr\_g\_c} \vee$   
 $c\_in' > c\_in + \text{arr\_g\_c}']$

## Itinerario

nome StringM

## Tappa

arrivoGiorno	PosInteger
arrivoOra	time
partenzaGiorno	PosInteger
partenzaOra	time

< V.Tappa.date >

$$\forall t, p\_h, p\_g, a\_g, a\_h$$

$$[Tappa(t) \wedge partenzaGiorno(t, p\_g) \wedge partenzaOra(t, p\_h)$$

$$\wedge arrivoGiorno(t, a\_g) \wedge arrivoOra(t, a\_h)]$$

$$\rightarrow ((a\_g < p\_g) \vee a\_g = p\_g \wedge a\_h < p\_h))$$

< V.Tappa.succPart >

$$\forall i, p, t, \alpha_t, \sigma_t, \sigma_p$$

$$itinerario(i) \wedge partitin(p, i) \wedge partenzaOra(p, i, \sigma_p) \wedge$$

$$Tappa(t) \wedge tappaltin(t, i) \wedge arrivo(t, \alpha_t) \wedge giorno(\alpha_t, 1) \wedge$$

$$\wedge giorno(\alpha_t, \sigma_t) \rightarrow \sigma_p < \sigma_t$$

< V.Tappa.primaArrivo >

$$\forall i, dest\_arr, dest\_arr_g, dest\_arr_h, t, t\_part_g, t\_part_h$$

$$itinerario(i) \wedge arrivoltin(i, dest\_arr) \wedge$$

$$arrivoGiorno(dest\_arr, i, dest\_arr_g) \wedge$$

$$arrivoOra(dest\_arr, i, dest\_arr_h) \wedge$$

$$Tappa(t) \wedge tappaltin(t, i) \wedge partenzaGiorno(t, t\_part_g) \wedge$$

$$partenzaOra(t, t\_part_h) \rightarrow$$

$$[(t\_part\_g < dest\_arr\_g) \vee$$

$$(t\_part\_g = dest\_arr\_g) \wedge t\_part\_h < dest\_arr\_h)]$$

partitin  
partenzaOra time

arrivoltin  
arrivoGiorno PosInteger  
arrivoOra time

## Tappaltn

<V. tappaltn. soste. disj >  
 $\forall i, t, t', t_{\text{-part}}, t'_{\text{-part}}, t_{\text{-arr}}, t'_{\text{-arr}}$

Itinerario(i)  $\wedge$  Tappa(t)  $\wedge$  tappaltn(t, i)  $\wedge$  arrivo(t, t\_arr)  $\wedge$   
partenza(t, t\_part)  $\wedge$  Tappa(t')  $\wedge$  tappaltn(t', i)  $\wedge$   
 $\wedge$  arrivo(t', t'\_arr)  $\wedge$  partenza(t, t'\_part)  $\wedge$   $t \neq t'$   
 $\rightarrow (t_{\text{-arr}} > t'_{\text{-arr}}) \vee (t'_{\text{-part}} > t_{\text{-part}})$

Itinerario(i)  $\wedge$  Tappa(t)  $\wedge$  tappaltn(t, i)  $\wedge$   
partenzaGiorno(t, t\_part\_g)  $\wedge$  partenzaOra(t, t\_part\_h)  
 $\wedge$  arrivoGiorno(t, t\_arr\_g)  $\wedge$  arrivoOra(t, t\_arr\_h)  $\wedge$   
Tappa(t', i)  $\wedge$  partenzaGiorno(t', t'\_part\_g)  $\wedge$   
partenzaOra(t', t'\_arr\_h)  $\wedge$  arrivoGiorno(t', t'\_arr\_g)  $\wedge$   
arrivoOra(t', t'\_arr\_h)  $\wedge$  tappaltn(t', i)  $\wedge$   $t = t'$   $\rightarrow$   
 $\left[ \begin{array}{l} ((t'_arr_g > t_{\text{-part}_g}) \vee \\ ((t'_arr_g = t_{\text{-part}_g}) \wedge t'_arr_h > t_{\text{-part}_h})) \end{array} \right]$   
 $\vee$   
 $\left[ \begin{array}{l} ((t_{\text{-arr}_g} > t'_{\text{-part}_g}) \vee \\ ((t_{\text{-arr}_g} = t'_{\text{-part}_g}) \wedge t_{\text{-arr}_h} > t'_{\text{-part}_h})) \end{array} \right]$

## 4) Creazioni Tabelle

accorda segue e utilizza

Crociera (codice: StringS, inizio: date, tipo: TipoCROCIERA,  
cFamBambini\*: boolean, nave: StringM,  
itinerario: StringM)

Vincolo DB Foreign Key: nave references Nave(nome)

Vincolo DB Foreign Key: itinerario references Itinerario(nome)

Vincolo DB ennupla: tipo = PerFamiglia  $\leftrightarrow$  cFamBamb ≠ NULL

accorda tappaltn, tappaBest

Tappa (arrivoGiorno: PosInteger, arrivoOra: time,  
partenzaGiorno: PosInteger, partenzaOra: time,  
itinerario: StringM, destinazioneNome: StringM,  
destinazioneContinente: CONTINENTE)

Vincolo DB Foreign Key: itinerario references Itinerario(nome)

Vincolo DB Foreign Key: (destinazioneNome, destinazioneContinente)  
references Destinazione(nome, continente)

Vincolo DB ennupla: arrivoGiorno < partenzaGiorno  $\checkmark$

(arrivoGiorno = partenzaGiorno  $\wedge$  arrivoOra < partenzaOra)

accorda prenCliente, prenCroc

Prenotazione (id: integer, pax: PosInteger, istante: timestamp,  
cliente: integer, crociera: StringS)

Vincolo DB unique: (cliente, istante)

Vincolo DB Foreign Key: cliente references Cliente(id)

Vincolo DB Foreign Key: crociera references Crociera(codice)

Vincolo DB seriale: i valori della colonna id sono generati  
automaticamente dal DBMS

Cliente (id: integer, nascita: date, nome: StringM,  
cognome: StringM, indirizzo: INDIRIZZO )

Vincolo DB seriale: i valori della colonna id sono generati automaticamente dal DBMS

accoppa arrivoltin, partitin

Itinerario (nome: StringM, arrivoGiorno: PosInteger,  
arrivoOra: time, partenzaOra: time,  
Itin NomePart: StringM, Itin ContPart: CONTINENTE,  
Itin NomeArr: StringM, Itin ContArr: CONTINENTE)

Vincolo DB Foreign Key: (Itin NomePart, Itin ContPart) references Destinazione (nome, continente)

Vincolo DB Foreign Key: (Itin NomeArr, Itin ContArr) references Destinazione (nome, continente)

Destinazione (nome: StringM, continente: CONTINENTE,  
romantica: boolean, divertente: boolean)

Posto (id: integer, aperturaA: time, aperturaDa: time,  
nome: StringM)

Vincolo DB inclusione: id ⊆ postoDest (posto)

Vincolo DB seriale: i valori della colonna id sono generati automaticamente dal DBMS

Nave (nome: StringM, confort: TIPO CONFORT, pax: PosInt)

postoDest (posto: integer, destinazioneNome: stringM,  
destinazioneCont: CONTINENTE)

Vincolo DB foreign key: posto references Posto(id)

Vincolo DB foreign key: (destNome, destCont) references  
Destinazione(name, continente)

## 4) TRIGGER

Vincolo <V.Crociera.posti>

Il numero di posti prenotati per ogni crociera non può mai eccedere il numero di posti liberi nella nave utilizzata

Si decide di progettare una funzione:

DB. postiDisponibili (cr: stringS): integer

algoritmo

1 Q  $\leftarrow$  risultato della query seguente sostituendo a 'cr' il valore attuale di 'cr'

select h.pax - x.postiPrenotati as PostiLibri

from Crociera c, Nave h,

(select sum(z.pax) as postiPrenotati

From Prenotazione z

where z.crociera = cr) x

where c.codice = cr and c.nave = h.nome

2 if Q == NULL then

3 genera errore 'Crociera non trovata';

4 else

5 return il valore della colonna postiLibri dell'unica  
ennupla in Q;

## Trigger

Operazione: modifica di una tupla nella relazione Nave

Istante di invocazione: dopo l'operazione intercettata

Funzione:

1 old  $\leftarrow$  l'entità prima della modifica

2 new  $\leftarrow$  l'entità dopo la modifica

3 isError  $\leftarrow$  false;

4 if new.pax < old.pax then

5 isError  $\leftarrow$

exists(

select \*

from Crociera cr

where nave = new.nome

and DB.postiDisponibili(cr) < 0

6 if isError then blocca l'operazione

7 else permetti l'operazione

## Trigger

Operazione: modifica di una tupla della relazione Crociera

Istante di invocazione: dopo l'operazione intercettata

Funzione:

1 old  $\leftarrow$  l'entità del risultato prima della modifica

2 new  $\leftarrow$  l'entità del risultato dopo la modifica

3 isError  $\leftarrow$  false;

4 if new.nave  $\neq$  old.nave then

/\* Si sta tentando di sostituire la nave associata alla  
crociera new.codice \*/

5 isError  $\leftarrow$  DB.postiDisponibili(new.codice) < 0;

6 if isError then blocca l'operazione;

7 else permetti l'operazione;

## Trigger

Operazione: inserimento o modifica di una tupla nella rel. Prenotazione  
Istante di invocazione: dopo l'operazione intercettata

Funzione:

- 1 new  $\leftarrow$  l'entità inserita o risultato della modifica
- 2 isError  $\leftarrow$  DB.postiDisponibili(new.codice) < 0;
- 3 if isError then blocca l'operazione;
- 4 else permetti l'operazione;

Vincoli  $\leftarrow$  V. Nave.crociere.disj

Le crociere che utilizzano la stessa nave devono avere periodi tutti disgiunti

## Trigger

Operazioni: inserimento o modifica di una entità nella relazione Crociera

Istante di invocazione: prima dell'operazione intercettata

Funzione:

- 1 new  $\leftarrow$  l'entità che si sta inserendo opp l'entità risultato della modifica
- 2 isError  $\leftarrow$   $\exists$  una crociera (diversa da new) che si sovrappone a new  
`exists (`  
`select *`  
`from Crociera cr, Itinerario cr_i, Itinerario new_i`  
`where cr.nave = new.nave`  
`and cr.codice <> new.codice`  
`and cr.itinerario <> cr_i.nome`  
`and new.itinerario = new_i.nome`  
`and cr.inizio <= new.inizio + new_i.arrivoGiorno`

and cr.inizio + cr.i.arrivoGiorno >= new.inizio

3 if isError then blocca l'operazione

4 else permetti l'operazione

## Trigger

Operazione: inserimento o modifica di una tupla nella rel. Tappa  
Istante di invocazione: prima dell'operazione intercettata

### Funzione:

1 new  $\leftarrow$  l'entità del risultato della modifica

2 isError  $\leftarrow$  //  $\exists$  una crociera cr associata all'itin della tappa new?  
exist(

select \*

from Crociera cr

where new.itinerario = cr.itinerario

3 if isError then blocca l'operazione

4 else permetti l'operazione

## Use-case SQL

Specifico use-case: Verbalizzazione

prenota(d: integer, cr: stringS, posti: posinteger): integer

algoritmo:

1 if GestionePrenotazioni.postiDisponibili(cr) restituisce errore:

2 then molla l'errore

3 else if GestionePrenotazioni.postiDisponibili(cr) < 0:

4 then ritorna l'errore 'Posti disp. finiti'

5 I  $\leftarrow$  il risultato del comando SQL ottenuto sostituendo a d, cr, i i rispettivi valori attuali:

```
insert into Prenotazione (cliente, crociera, pax, istante)
values (:cl, :cr, :posti, current_timestamp)
returning id
```

6 if I rappresenta un errore then: moltra errore  
else: return il valore di id

postiDisponibili (cr: String): integer

algoritmo:

- 1 Q  $\leftarrow$  risultato della query  

```
select DB.postiDisponibili(cr) as postiLiberi
```
- 2 if Q rappresenta un errore then: moltra l'errore
- 3 else: return il valore della colonna 'postiLiberi'

Specifico UseCase: GestioneClienti

algoritmo:

nuovo (no: StringM, c: StringM, na: date, i: Indirizzo): integer

- 1 Q  $\leftarrow$  il risultato del comando SQL  

```
Insert into Cliente (nome, cognome, nascita, indirizzo)
values (:no, :c, :na, :i)
```
- 2 returning id
- 3 return il valore della colonna id;

cerca (no\*: StringM, co\*: StringM, na\*: date): Collezione(<id, nome,  
cognome, nascita, indirizzo>)

- 1 PHI-Nome  $\leftarrow$  'true' se no='NULL' e 'nome = no' altrimenti
- 2 PHI-Cognome  $\leftarrow$  'true' se co='NULL' e cognome=co "
- 3 PHI-Nascita  $\leftarrow$  'true' se na='NULL' e nascita=na "

4  $Q \leftarrow$  risultato della query SQL

```
select id, nome, cognome, nascita, indirizzo  
from Cliente
```

where PHI-Nome and PHI-Cognome and PHI-Nascita

5 return  $Q_i$

Specifica Use-Case: OperazioniCrociere

tipiCrociere( $cr : StringS$ ): {'tradizionali', 'alternative'}

algoritmo:

1  $Q \leftarrow$  risultato della query SQL

```
select sum(case when d.romantica then 1 else 0 end) -  
sum(case when d.romantica then 1 else 0 end) as delta
```

From Crociera c, Tappa t, Destinazione d

Where c.codice =: cod

and c.tipo = 'LunaDiMiele'

and c.itinerario = t.itinerario

and t.destinazioneNome = d.nome

and t.destinazioneContinente = d.continente

2 If  $Q = \text{NULL}$  Then: genera errore

3 else:  $t \leftarrow$  unica ennupla di  $Q$

If  $t.\delta \geq 0$  Then: return tradizionale;

else return alternativa;

Specifica usecase: StatisticheClienti

et2-MediaEsotica( $C : \text{Insieme(Integer)}$ ): real

algoritmo:

1 Crea tab. temporanea TmpClienti(cliente)

Foreign Key: cliente ref Cliente(id)

2  $I \leftarrow$  risultato del seguente comando SQL

insert into TmpClienti(cliente)  
values (-...)

3 if  $I$  rapp. un errore di FK per un certo  $c \in C$  then:  
genera errore e return;

4  $Q \leftarrow$

select avg(floor((Current-date - cl.nascita)/365)) as etatMedia  
from (select distinct pr.diente as id  
from Prenotazione pr, Crociera cr, Tappa t, TmpClienti tmp  
where pr.diente = tmp.diente  
and pr.crociera = cr.codice  
and cr.itinerario = t.itinerario  
and DB.isDestinazioneEsotica(t.destinazioneNome,  
t.destinazioneContinente)) codice\_diente,

Cliente d

where codice\_diente.id = d.id

eta(cl:integer): real

algoritmo:

1  $Q \leftarrow$  risultato della query

select floor((Current-Date - nascita)/365) as eta

from Cliente

where id = :cl

2 IF  $Q$  insieme vuoto then: genera errore

3 else: return il valore di  $Q$

Specifico usecase : StatisticheDestinazioni  
esotica ( nome : StringM, continente : Continente ) : boolean  
algoritmo :

- 1 Q  $\leftarrow$  risultato della query  
select DB.isDestinazioneEsotica( : nome, : continente )
- 2 if Q è vuoto then : genera errore
- 3 else : return il valore booleano

gettonata ( nome : StringM, continente : Continente ) : boolean  
algoritmo :

- 1 Q  $\leftarrow$  risultato della query  
select DB.isDestinazioneGettonata( : nome, : continente )
- 2 if Q rappresenta un errore then : invola errore
- 3 else: return il valore di Q

percentualeGettonate ( D : Insieme(< nome : StringM, cont : Continente >) ) : real

algoritmo :

- 1 if  $|D| = 0$  then : genera errore
- 2 Crea tab temporanea  
TmpDest ( nome , continente )  
PK : ( nome, continente ) refer. Destinazione ( nome, continente )
- 3 I  $\leftarrow$  risultato della query  
Insert into TmpDest ( nome, continente )  
values (....)
- 4 if I rappresenta un errore di PK per un certo d=(n,c)  $\in$  D then :  
genera errore

- 5  $Q \leftarrow$  risultato della query  
 $\text{select sum[when } DB.\text{isDestinazioneGettonata(t.nome, t.continente)} \\ \text{Then 1 else 0 end] / count(*) \text{ as result}$   
 $\text{from TmpDest t}$
- 6 elimina TmpDest
- 7 return il valore di Q;

Si definiscono le seguenti funzioni

- $DB.\text{isDestinazioneEsotica}(\text{name: String}, \text{continente: Continente}): \text{boolean}$
- 1  $Q \leftarrow$  risultato della query  
2  $\text{select (continente} \neq \text{'E'}) \text{ as esotica}$   
 $\text{from Continente}$   
 $\text{where name} = : \text{name} \text{ and continente} = : \text{continente}$   
3 if Q vuoto then: genera errore  
4 else: return il valore di Q

- $DB.\text{isDestinazioneGettonata}(\text{name: String}, \text{continente: Continente}): \text{boolean}$
- algoritmo
- 1  $Q \leftarrow$  risultato della query  
 $\text{select cr.tipo as tipoCrociere, count(*) as numero}$   
 $\text{from Crociera cr, Tappa t}$   
 $\text{where cr.itinerario} = t.itinerario$   
 $\text{and t.destinazioneNome} = : \text{name}$   
 $\text{and t.destinazioneContinente} = : \text{continente}$   
 $\text{and (current_date - cr.inizio)} \leq 365 * 2$   
 $\text{group by cr.tipo}$
- 2 IF  $Q$  contiene due ennuplie then:

- 3       $ldm \leftarrow$  il valore per la colonna numero della tupla con  
          valore 'LunaBimba'
- 4       $fam \leftarrow$  il valore per la colonna numero della tupla con  
          valore 'PerFamiglie'
- 5      return  $ldm \geq 10$  or  $fam \geq 15$
- 6      else: return false;