



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

**DOCUMENTAZIONE PER PROGETTO
BASI DI DATI**

CdL Triennale in Informatica

CORSO BASI DI DATI

Andrea Marquez

N86005189

Mirko Maiello

N86005102

Rocco Molinari

N86005071

ANNO ACCADEMICO: 2024/2025

Indice

1	Introduzione	3
1.1	Descrizione Del Problema	3
2	Progettazione Concettuale	4
2.1	Class diagram	4
2.2	Ristrutturazione del Class Diagram	5
2.2.1	Analisi delle chiavi	5
2.2.2	Analisi degli attributi derivati	5
2.2.3	Analisi delle ridondanze	6
2.2.4	Analisi degli attributi strutturati	6
2.2.5	Analisi degli attributi a valore multiplo	6
2.2.6	Analisi delle gerarchie di specializzazione	7
2.3	Class diagram ristrutturato	8
2.4	Dizionario delle Classi	9
2.5	Dizionario delle Associazioni	11
2.6	Dizionario dei Vincoli	12
3	Progettazione Logica	13
3.1	Schema Logico	13
4	Progettazione Fisica	14
4.1	Definizione delle Tabelle	14
4.1.1	Definizione della Tabella AMMINISTRATORE	14
4.1.2	Definizione della Tabella BAGAGLIO	15
4.1.3	Definizione della Tabella PASSEGGERO	16
4.1.4	Definizione della Tabella PRENOTAZIONE	17
4.1.5	Definizione della Tabella UTENTEGENERICO	20
4.1.6	Definizione della Tabella VOLO	21
4.2	Trigger Functions	22
4.2.1	check_posto_occupato E check_volo_stato_per_prenotazione	22
4.2.2	fn_ck_data E fn_ck_gate	23
4.2.3	fn_ck_volo_tipo	24
4.2.4	fn_tg_passeggeronpren	24
4.3	Procedure	24
4.3.1	Cerca bagagli per prenotazione	25
4.3.2	Cerca bagagli per utente	26
4.3.3	Cerca bagaglio per id	27
4.3.4	Cerca passeggero per id prenotazione	28
4.3.5	Cerca volo	29
4.3.6	Elimina prenotazione	30
4.3.7	Modifica gate volo	31
4.3.8	Modifica prenotazione codF e volo	32
4.3.9	Modifica prenotazione codice	32
4.3.10	Prenota volo	33
4.3.11	Segnala smarrimento bagaglio	34
4.3.12	Visualizza prenotazioni per codF	35

4.3.13	Visualizza prenotazioni per volo	36
4.3.14	Visualizza smarrimenti	37
4.3.15	Visualizza voli	38

1 Introduzione

Il seguente elaborato ha lo scopo di documentare la progettazione e lo sviluppo di una base di dati relazionale del DBMS PostgreSQL, ad opera degli studenti Andrea Marquez, Mirko Maiello e Rocco Molinari del CdL in Informatica presso l'Università degli Studi di Napoli "Federico II". Il database nasce come progetto a scopi valutativi per il corso di Basi di Dati, ed implementa un sistema per la gestione dell'aeroporto di Napoli.

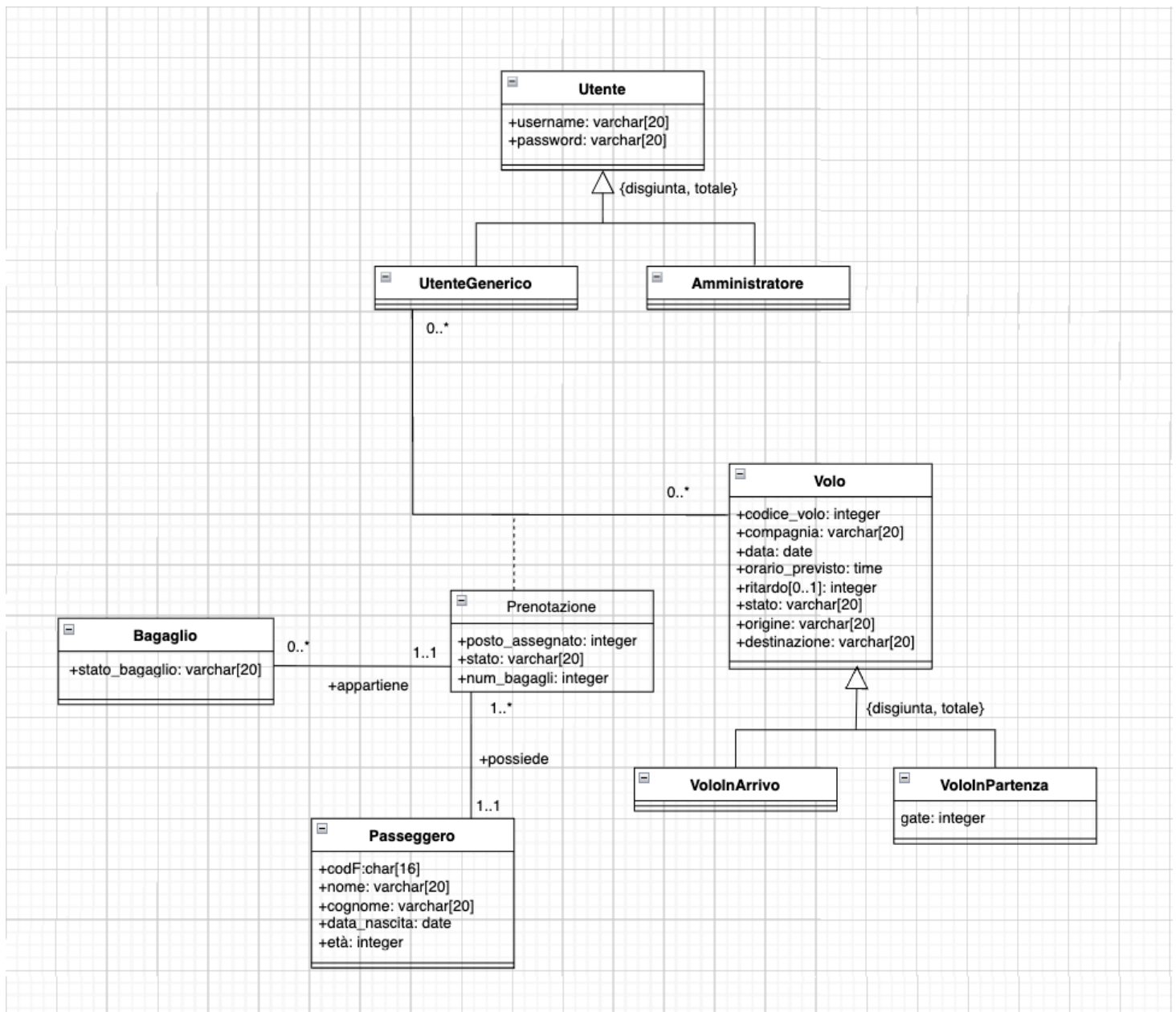
1.1 Descrizione Del Problema

La progettazione prevede lo sviluppo di una base di dati relazionale a supporto della gestione aeroportuale. Il sistema memorizza e organizza informazioni relative ai voli (codice, compagnia aerea, aeroporti di origine e destinazione, data, orario, ritardo, stato), alle prenotazioni (utente, passeggeri, biglietti, stato), ai gate di imbarco e ai bagagli, associati a ciascun volo tramite codice univoco. La base dati consente di gestire inserimento, aggiornamento e ricerca di tali informazioni, facilitando il monitoraggio dei voli in arrivo e in partenza, delle prenotazioni degli utenti e del tracciamento dei bagagli.

2 Progettazione Concettuale

In questo capitolo documentiamo la progettazione del database al suo livello di astrazione più alto. Partendo dall'analisi dei requisiti da soddisfare, si arriverà ad uno schema concettuale indipendente dalla struttura dei dati e dall'implementazione fisica degli stessi, rappresentato con un Class Diagram UML. Quest'ultimo evidenzierà le entità rilevanti nel problema, oltre alle relazioni che intercorrono tra esse e gli eventuali vincoli da imporre.

2.1 Class diagram



2.2 Ristrutturazione del Class Diagram

Si procede alla ristrutturazione del Class Diagram, al fine di rendere quest'ultimo idoneo alla traduzione in schemi relazionali e di migliorarne l'efficienza. La ristrutturazione procederà secondo i seguenti punti:

- Analisi delle chiavi
- Analisi degli attributi derivati
- Analisi delle ridondanze
- Analisi degli attributi strutturati
- Analisi degli attributi a valore multiplo
- Analisi delle gerarchie di specializzazione

2.2.1 Analisi delle chiavi

Ai fini dell'efficienza nella rappresentazione delle varie entità, per Prenotazione e Bagaglio si è reso necessario l'uso di chiavi primarie surrogate, poiché non erano presenti attributi naturali idonei a fungere da chiavi candidate: nessun campo, infatti, era in grado di garantire un'identificazione univoca e stabile delle istanze. Si è quindi optato per identificativi interi generati artificialmente, che consentono di distinguere le tuple in modo semplice e con minore dispendio computazionale.

Per altre entità, invece, sono stati adottati attributi naturali già esistenti e sufficienti a garantire l'univocità, scelti tra le possibili chiavi candidate: l'attributo username per Utentegenerico e Amministratore, codice_volo per Volo e codF(codice fiscale) per Passeggero.

2.2.2 Analisi degli attributi derivati

Per ottimizzare ulteriormente l'utilizzo delle risorse di calcolo, analizziamo gli eventuali attributi derivati, ovvero calcolabili a partire da altri attributi.

Nel nostro caso l'attributo età del Passeggero è derivabile da data_nascita. A seguito di un'analisi attenta, non risulta conveniente la storicizzazione di tale valore come attributo fisico: l'età varia nel tempo e memorizzarla introdurrebbe ridondanza e potenziali inconsistenze (necessità di aggiornare ogni record ad ogni compleanno). Si è quindi scelto di non persistere età, calcolandolo a query time quando necessario, così da garantire coerenza dei dati e miglior normalizzazione del modello.

2.2.3 Analisi delle ridondanze

Analizziamo ora l'eventuale presenza di associazioni ridondanti tra le varie entità in maniera tale da evitare incoerenze nella rappresentazione logica dei dati. E' stata individuata una ridondanza all'interno della classe Prenotazione, relativa all'attributo num_bagagli, tale informazione risulta interamente derivabile dall'associazione tra Prenotazione e Bagaglio.. La presenza di questo attributo avrebbe introdotto il rischio di incoerenze logiche nel modello, in quanto sarebbe stato necessario mantenerlo costantemente allineato con il numero effettivo di bagagli. In caso contrario, si sarebbero generate situazioni di disallineamento tra i dati (ad esempio, un valore di num_bagagli non corrispondente al numero reale di bagagli registrati). Inoltre, la ridondanza avrebbe dato luogo a anomalie di aggiornamento, imponendo aggiornamenti multipli in occasione di inserimenti, cancellazioni o modifiche dei bagagli. Per queste ragioni, l'attributo num_bagagli è stato rimosso dal modello, garantendo così una rappresentazione più pulita, coerente e normalizzata dei dati, senza perdita di informazione, che rimane comunque ricavabile in fase di interrogazione.

2.2.4 Analisi degli attributi strutturati

E' necessario esaminare e opportunamente trasformare gli eventuali attributi strutturati presenti nelle entità. Tali attributi, infatti, non risultano direttamente rappresentabili all'interno di un DBMS relazionale e necessitano quindi di essere eliminati o codificati mediante soluzioni alternative. Nel modello concettuale realizzato, tuttavia, non sono stati individuati attributi di questo tipo.

2.2.5 Analisi degli attributi a valore multiplo

Si procede quindi alla verifica della presenza di attributi a valore multiplo all'interno delle entità. anch'essi non logicamente rappresentabili e quindi da gestire nello schema concettuale ristrutturato. Anche questa volta nel modello concettuale analizzato non si riscontra tuttavia alcun attributo di questo tipo.

2.2.6 Analisi delle gerarchie di specializzazione

Infine è fondamentale gestire eventuali gerarchie di specializzazione, altro elemento non rappresentabile in un DBMS relazionale.

Nel modello concettuale sono presenti due gerarchie di specializzazione, entrambe totali e disgiunte:

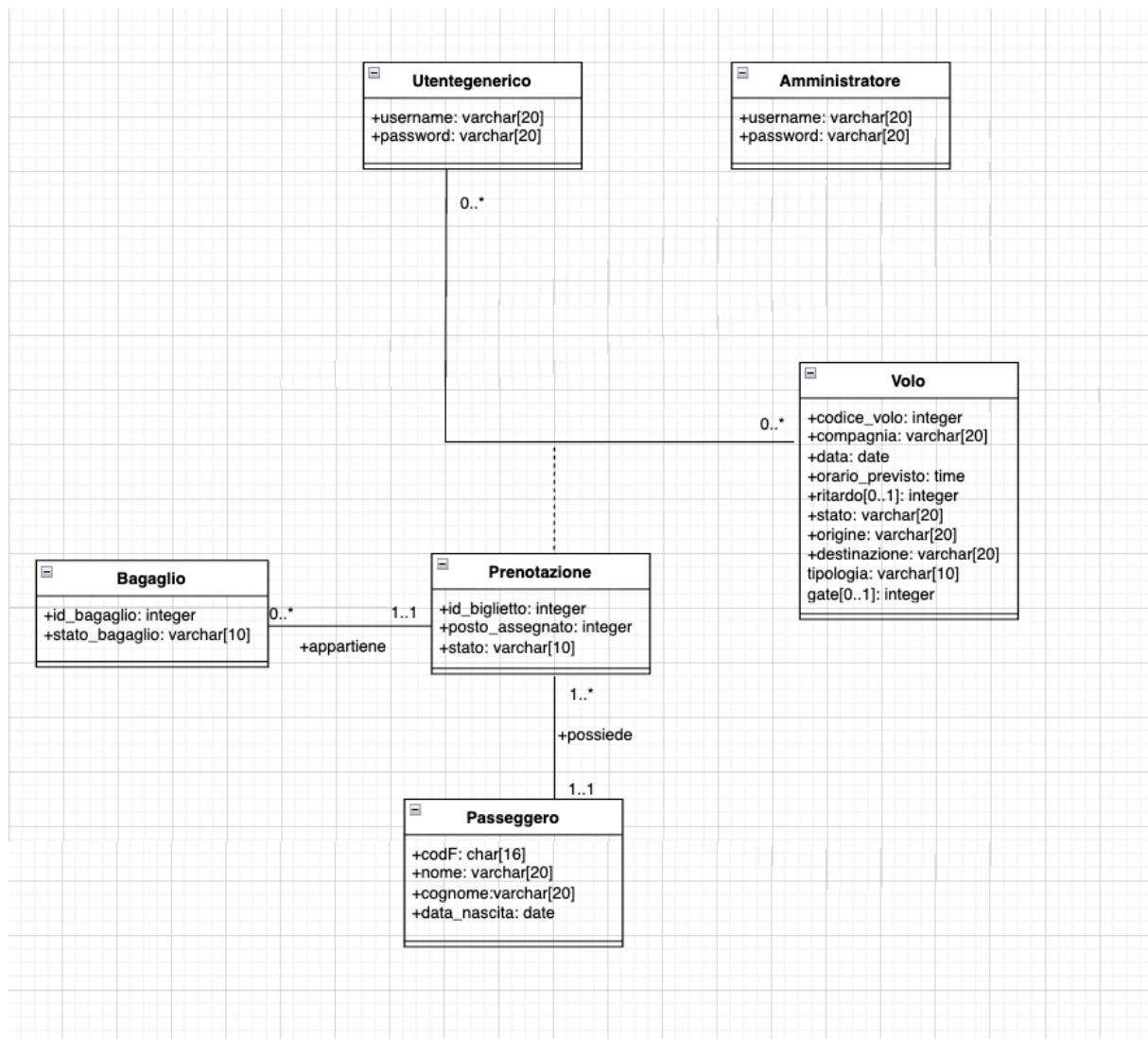
Utente \rightarrow UtenteGenerico / Amministratore

Volo \rightarrow VoloInArrivo / VoloInPartenza

Per quanto riguarda la gerarchia Utente \rightarrow UtenteGenerico / Amministratore, ogni utente deve necessariamente appartenere a una delle due sottoclassi e non può appartenere a entrambe. In fase di ristrutturazione si è scelto di accorpare la superclasse nelle sottoclassi, eliminando quindi la tabella padre e mantenendo soltanto le entità UtenteGenerico e Amministratore. Questa decisione è motivata dal fatto che, nel dominio considerato, non esiste un utente “astratto” a cui fare riferimento: ogni istanza concreta è sempre e solo un utente generico o un amministratore. Inoltre, le due sottoclassi presentano relazioni e responsabilità completamente differenti per questo motivo mantenere entità separate consente di rappresentare in modo più chiaro e coerente la distinzione di ruoli.

Per la gerarchia Volo \rightarrow VoloInArrivo / VoloInPartenza, anch'essa di tipo totale e disgiunto, si è scelto di accorpare le sottoclassi nella superclasse, eliminando quindi le entità specializzate e introducendo in Volo un attributo tipologia che indica se il volo è in arrivo oppure in partenza. È stato inoltre aggiunto l'attributo gate, che può rimanere nullo per i voli in arrivo o contenere un valore nei voli in partenza. Questa decisione è stata guidata non solo dall'obiettivo di ridurre il numero di tabelle e semplificare le interrogazioni, ma anche dal ruolo centrale che l'entità Volo ricopre nel modello: essa è infatti direttamente associata ad UtenteGenerico e Prenotazione. Mantenere sottoclassi separate avrebbe comportato complicazioni nella gestione di tali associazioni e una maggiore frammentazione del modello, mentre la soluzione adottata garantisce una rappresentazione più lineare ed efficace dei legami tra le diverse entità.

2.3 Class diagram ristrutturato



2.4 Dizionario delle Classi

Classe	Descrizione	Attributi
Utente generico	Rappresenta un utente registrato del sistema.	<ul style="list-style-type: none"> • username (varchar[20], PK): Nome utente. • password (varchar[20]): Credenziali di accesso.
Amministratore	Rappresenta un utente con privilegi amministrativi.	<ul style="list-style-type: none"> • username (varchar[20], PK): Nome utente. • password (varchar[20]): Credenziali di accesso.
Volo	Descrive un volo gestito dal sistema.	<ul style="list-style-type: none"> • codiceVolo (integer, PK): Codice identificativo del volo. • compagnia (varchar[20]): Nome della compagnia aerea. • data (date): Data del volo. • orarioPrevisto (time): Orario previsto. • ritardo (integer, opzionale): Minuti di ritardo. • stato (varchar[20]): Stato del volo. • origine (varchar[20]): Aeroporto di partenza. • destinazione (varchar[20]): Aeroporto di arrivo. • tipologia (varchar[10]): Tipologia del volo (es. arrivo, partenza). • gate (integer, opzionale): Gate assegnato.
Bagaglio	Descrive un bagaglio associato a una prenotazione.	<ul style="list-style-type: none"> • id_bagaglio (integer, PK): Identificativo univoco del bagaglio. • statoBagaglio (varchar[10]): Stato attuale del bagaglio.
Prenotazione	Descrive una singola prenotazione di un biglietto.	<ul style="list-style-type: none"> • id_biglietto (integer, PK): Identificativo univoco della prenotazione. • postoAssegnato (integer): Posto assegnato per il volo. • stato (varchar[10]): Stato della prenotazione.

Classe	Descrizione	Attributi
Passeggero	Descrive un passeggero.	<ul style="list-style-type: none">• codF (char[16], PK): Codice Fiscale del passeggero.• nome (varchar[20]): Nome del passeggero.• cognome (varchar[20]): Cognome del passeggero.• dataNascita (date): Data di nascita.

2.5 Dizionario delle Associazioni

Nome	Descrizione	Classi coinvolte
possiede	Associazione tra un passeggero e le sue prenotazioni.	<ul style="list-style-type: none"> • Passeggero (0..*): Un passeggero possiede almeno una prenotazione. • Prenotazione (1..1): Una prenotazione è posseduta da un solo passeggero.
appartiene	Associazione tra una prenotazione e i suoi bagagli.	<ul style="list-style-type: none"> • Prenotazione (0..*): Una prenotazione può avere zero o più bagagli. • Bagaglio (1..1): Un bagaglio è associato a una sola prenotazione.
Prenotazione (Utente-Volo)	Associazione, realizzata tramite la classe associativa "Prenotazione", tra un utente e i voli che quest'ultimo può prenotare.	<ul style="list-style-type: none"> • Utentegenerico (0..*): Un utente può prenotare zero o più voli. • Volo (0..*): Un volo può essere prenotato da zero o più utenti.

2.6 Dizionario dei Vincoli

Nome	Descrizione
Codice volo univoco	Ogni volo deve avere un codice univoco nel sistema.
Gate disponibile	Un gate può essere assegnato a un solo volo in partenza nello stesso orario, considerando anche i tempi di preparazione.
Bagaglio unico	Ogni bagaglio appartiene a una sola prenotazione.
Prenotazione valida	Ogni prenotazione deve essere effettuata da un utente registrato nel sistema.
Ritardo non negativo	L'attributo "ritardo" di un volo deve essere maggiore o uguale a zero.
Email univoca	Ogni utente (generico o amministratore) deve avere un indirizzo email univoco nel sistema.
Codice Fiscale Valido	Il Codice Fiscale di un passeggero deve essere alfanumerico, scritto con lettere maiuscole e di lunghezza: max 16 caratteri.
Data Nascita Valida	La data di nascita del passeggero deve essere precedente alla data odierna e successiva al 1900.
Orario Volo Valido	L'orario del volo deve essere in formato HH:MM e compreso tra 00:00 e 23:59.
Stato Volo Consistente	Lo stato del volo deve essere uno tra: "in orario", "in ritardo", "cancellato", "atterrato", "decollato".
Stato Prenotazione Consistente	Lo stato della prenotazione deve essere uno tra: "confermata", "in attesa", "cancellata".
Stato Bagaglio Consistente	Lo stato del bagaglio deve essere uno tra: "registrato", "caricato", "ritiro", "smarrito".
Tipologia Volo Valida	La tipologia del volo deve essere "Arrivo" o "Partenza".
Gate Solo Partenze	L'attributo gate può essere valorizzato solo per voli di tipologia "Partenza".
Data Volo Futura	La data del volo non può essere nel passato (eccetto per voli storici già completati).
Obbligo prenotazione passeggero	Nel database non può esistere un passeggero senza prenotazioni associate
Consistenza Prenotazione-Volo	Una prenotazione può essere associata solo a voli la cui data non sia già trascorsa al momento della prenotazione.

3 Progettazione Logica

In questo capitolo continuiamo la fase di progettazione, scendendo ad un livello di astrazione più basso rispetto al precedente. Lo schema concettuale verrà tradotto in uno schema logico.

3.1 Schema Logico

A seguire è riportato lo schema logico della base di dati. Al suo interno, le chiavi primarie sono indicate con una sottolineatura singola mentre le chiavi esterne con una sottolineatura doppia.

- Utentegenerico (username, password)
- Amministratore (username, password)
- Volo (codice_volo, compagnia, data, orario_previsto, ritardo, stato, origine, destinazione tipologia, gate)
- Prenotazione (id_biglietto, posto_assegnato, stato, username, codice_volo, codF)
 - codice_volo → Volo.codice_volo
 - codF → Passeggero.codF
 - username → Utentegenerico.username
- Passeggero (codF, nome, cognome, data_nascita)
- Bagaglio (id_bagaglio, stato_bagaglio, id_biglietto)
 - id_biglietto → Prenotazione.id_biglietto

4 Progettazione Fisica

In questo capitolo verrà riportata l'implementazione dello schema logico sopra descritto.

4.1 Definizione delle Tabelle

Di seguito sono riportate le definizioni delle tabelle, dei loro vincoli intrarelazionali e di eventuali semplici strutture per la loro gestione.

Disclaimer: nella seguente sezione, insieme alle istruzioni `CREATE TABLE`, sono riportati anche i comandi `CREATE TRIGGER` associati, al fine di documentare in un unico punto la struttura delle tabelle e le regole a esse collegate. Si precisa che l'ordine qui presentato ha valore puramente descrittivo: in un contesto reale di implementazione su PostgreSQL, le `TRIGGER FUNCTION` devono essere definite prima, e solo successivamente possono essere creati i relativi `TRIGGER`.

4.1.1 Definizione della Tabella AMMINISTRATORE

```
1  -- =====
2  -- TABELLA: Amministratore
3  -- =====
4
5  -- Definizione tabella
6
7
8  CREATE TABLE public."Amministratore" (
9      username character varying(20) NOT NULL,
10     password character varying(20) NOT NULL
11 );
12
13
14 ALTER TABLE public."Amministratore" OWNER TO postgres;
15
16 -- Vincoli (PK/UNIQUE)
17
18 ALTER TABLE ONLY public."Amministratore"
19     ADD CONSTRAINT pk_amministratore PRIMARY KEY (username);
```

Figura 1: Codice SQL della tabella AMMINISTRATORE

4.1.2 Definizione della Tabella BAGAGLIO

```

1  -- =====
2  -- TABELLA: Bagaglio
3  -- =====
4
5  -- Sequenze correlate
6
7
8  CREATE SEQUENCE public.id_bagaglio_seq
9      START WITH 1
10     INCREMENT BY 1
11     NO MINVALUE
12     NO MAXVALUE
13     CACHE 1;
14
15
16  ALTER SEQUENCE public.id_bagaglio_seq OWNER TO postgres;
17
18  -- Definizione tabella
19
20
21  CREATE TABLE public."Bagaglio" (
22      id_bagaglio integer DEFAULT nextval('public.id_bagaglio_seq'::regclass) NOT NULL,
23      stato_bagaglio character varying(10) NOT NULL,
24      id_biglietto integer NOT NULL,
25      CONSTRAINT ck_stato CHECK (((stato_bagaglio)::text = 'registrato'::text) OR
26      ((stato_bagaglio)::text = 'caricato'::text) OR
27      ((stato_bagaglio)::text = 'ritiro'::text) OR
28      ((stato_bagaglio)::text = 'smarrito'::text)))
29  );
30
31
32  ALTER TABLE public."Bagaglio" OWNER TO postgres;
33
34  -- Vincoli (PK/UNIQUE)
35
36
37  ALTER TABLE ONLY public."Bagaglio"
38      ADD CONSTRAINT "Bagaglio_pkey" PRIMARY KEY (id_bagaglio);
39
40
41  -- Vincoli di chiave esterna
42
43
44  ALTER TABLE ONLY public."Bagaglio"
45      ADD CONSTRAINT fk_bagaglio_prenotazione FOREIGN KEY (id_biglietto) REFERENCES
46      public."Prenotazione"(id_biglietto) ON DELETE CASCADE;
47

```

Figura 2: Codice SQL della tabella BAGAGLIO

4.1.3 Definizione della Tabella PASSEGGERO

```
1  -- =====
2  -- TABELLA: Passeggero
3  -- =====
4
5  -- Definizione tabella
6
7
8  CREATE TABLE public."Passeggero" (
9      "codF" character varying(16) NOT NULL,
10     nome character varying(20) NOT NULL,
11     cognome character varying(20) NOT NULL,
12     data_nascita date NOT NULL,
13     CONSTRAINT ck_cf CHECK (((("codF")::text = upper(("codF")::text)) AND (length(("codF")::text) = 16))),
14     CONSTRAINT ck_data CHECK (((data_nascita > '1900-01-01'::date) AND (data_nascita < CURRENT_DATE)))
15 );
16
17
18 ALTER TABLE public."Passeggero" OWNER TO postgres;
19
20 -- Vincoli (PK/UNIQUE)
21 |
22
23 ALTER TABLE ONLY public."Passeggero"
24     ADD CONSTRAINT "Passeggero_pkey" PRIMARY KEY ("codF");
25
```

Figura 3: Codice SQL della tabella PASSEGERO

4.1.4 Definizione della Tabella PRENOTAZIONE

```

1  -- =====
2  -- TABELLA: Prenotazione
3  -- =====
4
5  -- Sequenze correlate
6
7  CREATE SEQUENCE public."Prenotazione_id_biglietto_seq"
8      AS integer
9      START WITH 1
10     INCREMENT BY 1
11     NO MINVALUE
12     NO MAXVALUE
13     CACHE 1;
14
15
16 ALTER SEQUENCE public."Prenotazione_id_biglietto_seq" OWNER TO postgres;
17
18 -- Definizione tabella
19
20 CREATE TABLE public."Prenotazione" (
21     id_biglietto integer NOT NULL,
22     posto_assegnato integer NOT NULL,
23     stato character varying(10) NOT NULL,
24     id_volo integer NOT NULL,
25     username character varying(20) NOT NULL,
26     "codF_passeggero" character varying(16) NOT NULL,
27     CONSTRAINT ck_stato CHECK (((stato)::text = 'confermata'::text) OR
28     ((stato)::text = 'inAttesa'::text) OR ((stato)::text = 'cancellata'::text)))
29 );
30
31
32 ALTER TABLE public."Prenotazione" OWNER TO postgres;
33
34 -- Default/ALTER COLUMN
35
36
37 ALTER TABLE ONLY public."Prenotazione" ALTER COLUMN id_biglietto SET DEFAULT
38 nextval('public."Prenotazione_id_biglietto_seq"::regclass');
39
40
41 -- Vincoli (PK/UNIQUE)
42
43
44 ALTER TABLE ONLY public."Prenotazione"
45     ADD CONSTRAINT "Prenotazione_pkey" PRIMARY KEY (id_biglietto);
46
47 ALTER TABLE ONLY public."Prenotazione"
48     ADD CONSTRAINT uq_prenotazione UNIQUE (id_volo, "codF_passeggero");
49
50 -- Vincoli di chiave esterna
51
52 ALTER TABLE ONLY public."Prenotazione"
53     ADD CONSTRAINT fk_prenotazione_passeggero FOREIGN KEY ("codF_passeggero") REFERENCES
54     public."Passeggero"("codF") ON DELETE CASCADE;
55
56 ALTER TABLE ONLY public."Prenotazione"
57     ADD CONSTRAINT fk_prenotazione_username FOREIGN KEY (username) REFERENCES
58     public."Utentegenerico"(username) ON DELETE CASCADE;
59
60 ALTER TABLE ONLY public."Prenotazione"
61     ADD CONSTRAINT fk_prenotazione_volo FOREIGN KEY (id_volo) REFERENCES
62     public."Volo"(codice_volo) ON DELETE CASCADE;

```

Figura 4: Codice SQL della tabella PRENOTAZIONE (parte 1)

```
1  -- Trigger (raggruppati qui per documentazione; ATTIVATI a fine file)
2
3  ✓ CREATE TRIGGER trg_check_posto_occupato BEFORE INSERT ON public."Prenotazione" FOR EACH ROW
4  EXECUTE FUNCTION public.check_posto_occupato(); -- Controlla che il posto non sia già occupato
5
6  ✓ CREATE TRIGGER trg_check_volo_stato_per_prenotazione BEFORE INSERT ON public."Prenotazione" FOR EACH ROW
7  EXECUTE FUNCTION public.check_volo_stato_per_prenotazione(); -- Impedisce prenotazioni su voli non validi
```

Figura 5: Codice SQL della tabella PRENOTAZIONE (parte 2)

```
17 CREATE TRIGGER tg_passeggeronpren
18 AFTER DELETE ON "Prenotazione"
19 FOR EACH ROW
20 EXECUTE FUNCTION fn_tg_passeggeronpren();
21
```

Figura 6: Codice SQL della tabella PRENOTAZIONE (parte 3)

4.1.5 Definizione della Tabella UTENTEGENERICO

```
1  -- =====
2  -- TABELLA: Utentegenerico
3  -- =====
4
5  -- Definizione tabella
6
7  CREATE TABLE public."Utentegenerico" (
8      username character varying(20) NOT NULL,
9      password character varying(20) NOT NULL
10 );
11
12
13 ALTER TABLE public."Utentegenerico" OWNER TO postgres;
14
15 -- Vincoli (PK/UNIQUE)
16
17 ALTER TABLE ONLY public."Utentegenerico"
18     ADD CONSTRAINT pk_utentegenerico PRIMARY KEY (username);
19
```

Figura 7: Codice SQL della tabella UTENTEGENERICO

4.1.6 Definizione della Tabella VOLO

```

1  -- =====
2  -- TABELLA: Volo
3  -- =====
4
5  -- Definizione tabella
6
7
8  CREATE TABLE public."Volo" (
9      codice_volo integer NOT NULL,
10     compagnia character varying(20) NOT NULL,
11     data date NOT NULL,
12     orario_previsto time with time zone NOT NULL,
13     ritardo integer DEFAULT 0,
14     stato character varying NOT NULL,
15     origine character varying(20) NOT NULL,
16     destinazione character varying(20) NOT NULL,
17     tipologia character varying(10) NOT NULL,
18     gate integer,
19     CONSTRAINT ck_ritardo CHECK ((ritardo >= 0)),
20
21     CONSTRAINT ck_stato CHECK (((stato)::text = 'programmato'::text) OR
22
23     ((stato)::text = 'decollato'::text) OR ((stato)::text = 'inRitardo'::text)
24     OR ((stato)::text = 'atterrato'::text) OR ((stato)::text = 'cancellato'::text))),
25
26     CONSTRAINT ck_tipo CHECK (((tipologia)::text = 'inPartenza'::text) OR
27     ((tipologia)::text = 'inArrivo'::text)))
28 );
29
30
31 ALTER TABLE public."Volo" OWNER TO postgres;
32
33 -- Vincoli (PK/UNIQUE)
34
35
36 ALTER TABLE ONLY public."Volo"
37     ADD CONSTRAINT "Volo_pkey" PRIMARY KEY (codice_volo);
38
39 -- Trigger (raggruppati qui per documentazione)
40
41 CREATE TRIGGER tg_ck_data BEFORE INSERT OR UPDATE OF tipologia, data ON public."Volo" FOR EACH ROW
42 EXECUTE FUNCTION public.fn_ck_data(); -- Controlla coerenza tra stato e data volo
43
44 CREATE TRIGGER tg_ck_gate BEFORE INSERT OR UPDATE OF gate, data, orario_previsto ON public."Volo" FOR EACH ROW
45 EXECUTE FUNCTION public.fn_ck_gate(); -- Verifica correttezza e unicit  del gate
46
47 CREATE TRIGGER tg_ck_volo_tipo BEFORE INSERT ON public."Volo" FOR EACH ROW
48 EXECUTE FUNCTION public.fn_ck_volo_tipo(); -- Imposta tipologia volo e impedisce origine=destinazione

```

Figura 8: Codice SQL della tabella VOLO

4.2 Trigger Functions

4.2.1 check_posto_occupato E check_volo_stato_per_prenotazione

```

1  -- =====
2  -- SEZIONE TRIGGER FUNCTION
3  -- =====
4
5  -- Trigger Function per controllare posto duplicato
6  CREATE FUNCTION public.check_posto_occupato() RETURNS trigger
7  LANGUAGE plpgsql
8  AS $$
9  BEGIN
10     IF EXISTS (
11         SELECT 1
12         FROM public."Prenotazione"
13         WHERE id_volo = NEW.id_volo AND posto_assegnato = NEW.posto_assegnato
14     ) THEN
15         RAISE EXCEPTION 'Il posto % sul volo % è già prenotato', NEW.posto_assegnato, NEW.id_volo;
16     END IF;
17
18     RETURN NEW;
19 END;
20 $$;
21
22 ALTER FUNCTION public.check_posto_occupato() OWNER TO postgres;
23
24 -- Trigger function per controllare stato volo che si sta prenotando
25 CREATE FUNCTION public.check_volo_stato_per_prenotazione() RETURNS trigger
26 LANGUAGE plpgsql
27 AS $$
28 BEGIN
29     IF NOT EXISTS (
30         SELECT 1
31         FROM public."Volo"
32         WHERE codice_volo = NEW.id_volo
33         AND (stato = 'inRitardo' OR stato = 'programmato')
34     ) THEN
35         RAISE EXCEPTION 'Non puoi prenotare un volo che non è in partenza o è stato cancellato o decollato';
36     END IF;
37
38     RETURN NEW;
39 END;
40 $$;
41

```

Figura 9: Trigger functions "check_posto_occupato" E "check_volo_stato_per_prenotazione"

4.2.2 fn_ck_data E fn_ck_gate

```

42 ALTER FUNCTION public.check_volo_stato_per_prenotazione() OWNER TO postgres;
43
44 CREATE FUNCTION public.fn_ck_data() RETURNS trigger -- Trigger function che controlla coerenza stato/data
45 LANGUAGE plpgsql
46 AS $$
47 BEGIN
48 IF NEW.stato IS NULL OR NEW.data IS NULL THEN
49 RAISE EXCEPTION 'stato e data non possono essere NULL';
50 END IF;
51
52 IF NEW.stato IN ('cancellato','decollato') THEN
53 IF NEW.data::date > CURRENT_DATE THEN
54 RAISE EXCEPTION 'Volo con stato % deve avere data <= oggi (trovato: %)',
55 NEW.stato, NEW.data::date;
56 END IF;
57 ELSE
58 IF NEW.data::date < CURRENT_DATE THEN
59 RAISE EXCEPTION 'Volo con stato % deve avere data >= oggi (trovato: %)',
60 NEW.stato, NEW.data::date;
61 END IF;
62 END IF;
63
64 RETURN NEW;
65 END;
66 $$;
67
68
69 ALTER FUNCTION public.fn_ck_data() OWNER TO postgres;
70
71 CREATE FUNCTION public.fn_ck_gate() RETURNS trigger -- Trigger function che controlla disponibilità del gate
72 LANGUAGE plpgsql
73 AS $$
74 DECLARE
75 altro_volo integer;
76 BEGIN
77 -- gate considerato assegnato solo se > 0
78 IF NEW.gate IS NOT NULL AND NEW.gate > 0 THEN
79 IF NEW.tipologia <> 'inPartenza' THEN
80 RAISE EXCEPTION 'Il gate può essere assegnato solo ai voli in partenza';
81 END IF;
82
83 -- stesso gate, stessa data e stesso orario non possono avere due voli diversi
84 SELECT v.codice_volo
85 INTO altro_volo
86 FROM public."Volo" v
87 WHERE v.gate = NEW.gate
88 AND v.data = NEW.data
89 AND v.orario_previsto = NEW.orario_previsto
90 AND v.codice_volo <> NEW.codice_volo
91 LIMIT 1;
92
93 IF altro_volo IS NOT NULL THEN
94 RAISE EXCEPTION 'Gate % già assegnato per % alle % al volo %',
95 NEW.gate, NEW.data, NEW.orario_previsto, altro_volo;
96 END IF;
97 END IF;
98
99 RETURN NEW;
100 END;
101 $$;
102
103

```

Figura 10: Trigger functions "fn_ck_data" E "fn_ck_gate"

4.2.3 fn_ck_volo_tipo

```

104 ALTER FUNCTION public.fn_ck_gate() OWNER TO postgres;
105
106
107 CREATE FUNCTION public.fn_ck_volo_tipo() RETURNS trigger -- Trigger function che determina tipologia del volo
108 LANGUAGE plpgsql
109 AS $$
110 BEGIN
111     IF NEW.origine = NEW.destinazione THEN
112         RAISE EXCEPTION 'Origine e destinazione non possono essere uguali';
113     END IF;
114     IF NEW.origine = 'NAP' THEN
115         NEW.tipologia := 'inPartenza';
116     ELSE
117         NEW.tipologia := 'inArrivo';
118     END IF;
119
120     RETURN NEW;
121 END;
122 $$;
123
124
125 ALTER FUNCTION public.fn_ck_volo_tipo() OWNER TO postgres;
126

```

Figura 11: Trigger function "fn_ck_volo_tipo"

4.2.4 fn_tg_passeggeronpren

```

1 CREATE OR REPLACE FUNCTION fn_tg_passeggeronpren()
2 RETURNS trigger AS $$
3 BEGIN
4     IF NOT EXISTS (
5         SELECT 1
6         FROM "Prenotazione" p
7         WHERE p.codF_passeggero = OLD.codF_passeggero
8     ) THEN
9         DELETE FROM "Passeggero"
10        WHERE codF = codF_passeggero;
11    END IF;
12    RETURN OLD;
13 END;
14 $$
15 LANGUAGE plpgsql;

```

Figura 12: Trigger function "fn_tg_passeggeronpren"

4.3 Procedure

Disclaimer: le procedure riportate di seguito sono, nella maggior parte dei casi, eseguibili solo dagli utenti che risultano titolari dell'oggetto su cui l'operazione agisce (ad esempio prenotazioni o bagagli). Gli amministratori, invece, possono eseguire qualsiasi procedura senza limitazioni.

4.3.1 Cerca bagagli per prenotazione

```

1  -- =====
2  -- SEZIONE PROCEDURE
3  -- =====
4
5  CREATE PROCEDURE public.cerca_bagagli_per_prenotazione(IN p_codice_prenotazione integer,
6  IN p_username character varying) -- Cerca i bagagli legati a una prenotazione
7      LANGUAGE plpgsql
8      AS $$
9  DECLARE
10     v_id_biglietto INTEGER;
11     v_codF_passeggero VARCHAR;
12     v_is_admin BOOLEAN;
13     result RECORD;
14  BEGIN
15     SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
16     INTO v_is_admin;
17     IF NOT v_is_admin THEN
18         SELECT p.id_biglietto, p."codF_passeggero"
19         INTO v_id_biglietto, v_codF_passeggero
20         FROM public."Prenotazione" p
21         WHERE p.id_biglietto = p_codice_prenotazione AND p.username = p_username;
22
23     IF v_id_biglietto IS NULL THEN
24         RAISE EXCEPTION 'Prenotazione con codice % non trovata per l''utente %', p_codice_prenotazione,
25         p_username;
26     END IF;
27
28     IF v_codF_passeggero IS NULL THEN
29         RAISE EXCEPTION 'Il codice fiscale del passeggero non è associato alla prenotazione.';
30     END IF;
31
32     ELSE
33         SELECT p.id_biglietto, p."codF_passeggero"
34         INTO v_id_biglietto, v_codF_passeggero
35         FROM public."Prenotazione" p
36         WHERE p.id_biglietto = p_codice_prenotazione;
37     END IF;
38     FOR result IN
39         SELECT
40             b.id_bagaglio, b.stato_bagaglio, b.id_biglietto,
41             p.posto_assegnato,
42             v.codice_volo,
43             v.compagnia,
44             v.orario_previsto,
45             v.origine,
46             v.destinazione
47         FROM
48             public."Bagaglio" b
49         JOIN
50             public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
51         JOIN
52             public."Volo" v ON p.id_volo = v.codice_volo
53         WHERE
54             b.id_biglietto = v_id_biglietto
55     LOOP
56         RAISE NOTICE 'ID Bagaglio: %, Stato: %, Posto Assegnato: %, Codice Volo: %, Compagnia: %,
57         Orario: %, Origine: %, Destinazione: %',
58         result.id_bagaglio, result.stato_bagaglio, result.posto_assegnato, result.codice_volo,
59         result.compagnia, result.orario_previsto, result.origine, result.destinazione;
60     END LOOP;
61 END;
62 $$;

```

Figura 13: Procedure "Cerca bagagli per prenotazione"

4.3.2 Cerca bagagli per utente

```

65  -- Mostra bagagli per utente (o tutti se admin)
66  CREATE PROCEDURE public.cerca_bagagli_per_utente(IN username_input character varying)
67  LANGUAGE plpgsql
68  AS $$DECLARE
69  result RECORD;
70  bagaglio_cursor REFCURSOR;
71  v_is_admin BOOLEAN;
72  v_username varchar := username_input;
73  BEGIN
74
75      SELECT EXISTS(
76          SELECT 1
77          FROM public."Amministratore" a
78          WHERE a.username = v_username
79      )
80      INTO v_is_admin;
81
82  IF v_is_admin THEN
83
84      OPEN bagaglio_cursor FOR
85      SELECT
86          b.id_bagaglio,
87          p.id_biglietto AS numero_prenotazione,
88          ps.nome AS nome_passeggero,
89          ps.cognome AS cognome_passeggero,
90          p.username
91      FROM public."Bagaglio" b
92      JOIN public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
93      JOIN public."Passeggero" ps ON p."codF_passeggero" = ps."codF"
94      ORDER BY b.id_bagaglio;
95  ELSE
96
97      OPEN bagaglio_cursor FOR
98      SELECT
99          b.id_bagaglio,
100         p.id_biglietto AS numero_prenotazione,
101         ps.nome AS nome_passeggero,
102         ps.cognome AS cognome_passeggero,
103         p.username
104      FROM public."Bagaglio" b
105      JOIN public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
106      JOIN public."Passeggero" ps ON p."codF_passeggero" = ps."codF"
107      WHERE p.username = v_username
108      ORDER BY b.id_bagaglio;
109  END IF;
110  LOOP
111      FETCH bagaglio_cursor INTO result;
112      EXIT WHEN NOT FOUND;
113      RAISE NOTICE
114          'ID Bagaglio: %, Prenotazione: %, Nome: %, Cognome: %, Username: %',
115          result.id_bagaglio,
116          result.numero_prenotazione,
117          result.nome_passeggero,
118          result.cognome_passeggero,
119          result.username;
120  END LOOP;
121
122  CLOSE bagaglio_cursor;
123  END;
124  $$;

```

Figura 14: Procedure "Cerca bagagli per utente"

4.3.3 Cerca bagaglio per id

```

129 -- Cerca dettagli di un bagaglio tramite ID
130 CREATE PROCEDURE public.cerca_bagaglio_per_id(IN p_id_bagaglio integer, IN p_username character varying)
131 LANGUAGE plpgsql
132 AS $$
133 DECLARE
134     v_id_biglietto INTEGER;
135     v_is_admin BOOLEAN;
136     result RECORD;
137 BEGIN
138     SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
139     INTO v_is_admin;
140
141     IF NOT v_is_admin THEN
142         SELECT p.id_biglietto
143         INTO v_id_biglietto
144         FROM public."Prenotazione" p
145         WHERE p.id_biglietto = (SELECT id_biglietto FROM public."Bagaglio" WHERE id_bagaglio = p_id_bagaglio)
146         AND p.username = p_username;
147
148         IF v_id_biglietto IS NULL THEN
149             RAISE EXCEPTION 'L'utente % non è associato a nessun bagaglio con ID %', p_username,
150             p_id_bagaglio;
151         END IF;
152     ELSE
153         SELECT p.id_biglietto
154         INTO v_id_biglietto
155         FROM public."Prenotazione" p
156         WHERE p.id_biglietto = (SELECT id_biglietto FROM public."Bagaglio" WHERE id_bagaglio = p_id_bagaglio);
157     END IF;
158
159     FOR result IN
160         SELECT
161             b.id_bagaglio,
162             b.stato_bagaglio,
163             b.id_biglietto,
164             p.posto_assegnato,
165             v.codice_volo,
166             v.compagnia,
167             v.orario_previsto,
168             v.origine,
169             v.destinazione
170         FROM
171             public."Bagaglio" b
172         JOIN
173             public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
174         JOIN
175             public."Volo" v ON p.id_volo = v.codice_volo
176         WHERE
177             b.id_bagaglio = p_id_bagaglio
178     LOOP
179         RAISE NOTICE 'ID Bagaglio: %, Stato: %, Posto Assegnato: %, Codice Volo: %, Compagnia: %,
180         |orario: %, Origine: %, Destinazione: %',
181         result.id_bagaglio, result.stato_bagaglio, result.posto_assegnato, result.codice_volo,
182         result.compagnia, result.orario_previsto, result.origine, result.destinazione;
183     END LOOP;
184 END;
185 $$;

```

Figura 15: Procedure "Cerca bagaglio per id"

4.3.4 Cerca passeggero per id prenotazione

```

189 -- Visualizza i dati passeggero legato a prenotazione
190 CREATE PROCEDURE public.cerca_passeggero_per_id_prenotazione(IN p_id_prenotazione integer,
191 IN p_username character varying)
192 LANGUAGE plpgsql
193 AS $$
194 DECLARE
195     v_codF_passeggero VARCHAR;
196     v_is_admin BOOLEAN;
197     result RECORD;
198 BEGIN
199     SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
200     INTO v_is_admin;
201
202     IF v_is_admin THEN
203         SELECT p."codF_passeggero"
204         INTO v_codF_passeggero
205         FROM public."Prenotazione" p
206         WHERE p.id_biglietto = p_id_prenotazione;
207     ELSE
208         SELECT p."codF_passeggero"
209         INTO v_codF_passeggero
210         FROM public."Prenotazione" p
211         WHERE p.id_biglietto = p_id_prenotazione AND p.username = p_username;
212
213     IF v_codF_passeggero IS NULL THEN
214         RAISE EXCEPTION 'Prenotazione con ID % non trovata per l''utente %',
215             p_id_prenotazione, p_username;
216     END IF;
217 END IF;
218
219 FOR result IN
220     SELECT
221         ps."codF",
222         ps.nome,
223         ps.cognome,
224         ps.data_nascita
225     FROM
226         public."Passeggero" ps
227     WHERE
228         ps."codF" = v_codF_passeggero
229 LOOP
230     RAISE NOTICE 'Codice Fiscale: %, Nome: %, Cognome: %, Data Nascita: %',
231         result."codF", result.nome, result.cognome, result.data_nascita;
232 END LOOP;
233 END;
234 $$;

```

Figura 16: Procedure "Cerca passeggero per id prenotazione"

4.3.5 Cerca volo

```

240 CREATE PROCEDURE public.cerca_volo(IN p_codice_volo integer) -- Visualizza le info di un volo specifico
241 LANGUAGE plpgsql
242 AS $$
243 DECLARE
244     v_codice_volo INTEGER;
245     v_compagnia VARCHAR;
246     v_data DATE;
247     v_orario_previsto TIME WITH TIME ZONE;
248     v_ritardo INTEGER;
249     v_stato VARCHAR;
250     v_origine VARCHAR;
251     v_destinazione VARCHAR;
252     v_tipologia VARCHAR;
253     v_gate INTEGER;
254 BEGIN
255
256     SELECT
257         codice_volo,
258         compagnia,
259         data,
260         orario_previsto,
261         ritardo,
262         stato,
263         origine,
264         destinazione,
265         tipologia,
266         gate
267     INTO
268         v_codice_volo, v_compagnia, v_data, v_orario_previsto, v_ritardo, v_stato,
269         v_origine, v_destinazione, v_tipologia, v_gate
270     FROM
271         public."Volo"
272     WHERE
273         codice_volo = p_codice_volo;
274 IF NOT FOUND THEN
275     RAISE EXCEPTION 'Volo con codice % non trovato.', p_codice_volo;
276 END IF;
277 RAISE NOTICE 'Codice Volo: %, Compagnia: %, Data: %, Orario: %, Ritardo: %, Stato: %, Origine: %,
278 Destinazione: %, Tipologia: %, Gate: %',
279     v_codice_volo, v_compagnia, v_data, v_orario_previsto, v_ritardo, v_stato,
280     v_origine, v_destinazione, v_tipologia, v_gate;
281
282 END;
283 $$;

```

Figura 17: Procedure "Cerca volo"

4.3.6 Elimina prenotazione

```

288 -- Procedura che elimina una prenotazione (bagagli rimossi in cascata)
289 CREATE PROCEDURE public.elimina_prenotazione(IN p_id_prenotazione integer, IN p_username character varying)
290 LANGUAGE plpgsql
291 AS $$
292 DECLARE
293     v_is_admin    boolean;
294     v_owner_ok    boolean;
295 BEGIN
296     -- 1) Admin?
297     SELECT EXISTS(
298         SELECT 1
299         FROM public."Amministratore"
300         WHERE username = p_username
301     ) INTO v_is_admin;
302
303     -- 2) Esiste la prenotazione?
304     IF NOT EXISTS (
305         SELECT 1
306         FROM public."Prenotazione"
307         WHERE id_biglietto = p_id_prenotazione
308     ) THEN
309         RAISE EXCEPTION 'Prenotazione con ID % non esiste.', p_id_prenotazione;
310     END IF;
311
312     -- 3) Se non admin, verifico che appartenga all'utente
313     IF NOT v_is_admin THEN
314         SELECT EXISTS(
315             SELECT 1
316             FROM public."Prenotazione"
317             WHERE id_biglietto = p_id_prenotazione
318             AND username = p_username
319         ) INTO v_owner_ok;
320
321         IF NOT v_owner_ok THEN
322             RAISE EXCEPTION
323             'Operazione negata: la prenotazione % non appartiene all''utente %.',
324             p_id_prenotazione, p_username;
325         END IF;
326     END IF;
327
328     -- 4) Elimino (i bagagli collegati vengono rimossi via ON DELETE CASCADE)
329     DELETE FROM public."Prenotazione"
330     WHERE id_biglietto = p_id_prenotazione;
331
332     IF NOT FOUND THEN
333         -- teoricamente non ci arrivi perché abbiamo già verificato l'esistenza
334         RAISE EXCEPTION 'Eliminazione fallita per la prenotazione %.', p_id_prenotazione;
335     END IF;
336
337     RAISE NOTICE
338     'Prenotazione % eliminata con successo da % (admin=%).',
339     p_id_prenotazione, p_username, v_is_admin;
340 END;
341 $$;

```

Figura 18: Procedure "Elimina prenotazione"

4.3.7 Modifica gate volo

```

347 CREATE PROCEDURE public.modifica_gate_volo(IN p_username character varying, IN p_codice_volo integer,
348 IN p_nuovo_gate integer) -- Modifica il gate (solo admin, voli in partenza)
349     LANGUAGE plpgsql
350     AS $$
351 DECLARE
352     v_is_admin BOOLEAN;
353     v_stato_volo VARCHAR;
354 BEGIN
355
356     SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
357     INTO v_is_admin;
358
359     IF NOT v_is_admin THEN
360         RAISE EXCEPTION 'L'utente % non è un amministratore e non può eseguire questa operazione.', p_username;
361     END IF;
362
363     SELECT tipologia INTO v_stato_volo
364     FROM public."Volo"
365     WHERE codice_volo = p_codice_volo;
366
367     IF v_stato_volo IS NULL THEN
368         RAISE EXCEPTION 'Il volo con codice % non esiste.', p_codice_volo;
369     ELSIF v_stato_volo != 'inPartenza' THEN
370         RAISE EXCEPTION 'Il volo % non è in partenza, quindi non può essere modificato.', p_codice_volo;
371     END IF;
372     IF EXISTS (SELECT 1 FROM public."Volo" WHERE gate = p_nuovo_gate AND codice_volo != p_codice_volo) THEN
373         RAISE EXCEPTION 'Il gate % è già occupato da un altro volo.', p_nuovo_gate;
374     END IF;
375     UPDATE public."Volo"
376     SET gate = p_nuovo_gate
377     WHERE codice_volo = p_codice_volo;
378     RAISE NOTICE 'Gate del volo % modificato con successo a %.', p_codice_volo, p_nuovo_gate;
379
380 END;
381 $$;

```

Figura 19: Procedure "Modifica gate volo"

4.3.8 Modifica prenotazione codF e volo

```

386 -- Aggiorna bagagli tramite CF passeggero e volo
387 CREATE PROCEDURE public.modifica_prenotazione_codf_e_volo(IN p_codf_passeggero character varying,
388 IN p_codice_volo integer, IN p_username character varying, IN p_numero_bagagli integer)
389 LANGUAGE plpgsql
390 AS $$
391 DECLARE
392     v_id_biglietto INTEGER;
393 BEGIN
394     SELECT p.id_biglietto
395     INTO v_id_biglietto
396     FROM public."Prenotazione" p
397     WHERE p."codF_passeggero" = p_codf_passeggero
398           AND p.id_volo = p_codice_volo
399           AND p.username = p_username;
400
401     IF v_id_biglietto IS NULL THEN
402         RAISE EXCEPTION 'Prenotazione non trovata per il passeggero con CF % sul volo % per l''utente %',
403             p_codf_passeggero, p_codice_volo, p_username;
404     END IF;
405
406     DELETE FROM public."Bagaglio"
407     WHERE id_biglietto = v_id_biglietto;
408
409     FOR i IN 1..p_numero_bagagli LOOP
410         INSERT INTO public."Bagaglio" (id_biglietto, stato_bagaglio)
411         VALUES (v_id_biglietto, 'registrato');
412     END LOOP;
413     RAISE NOTICE 'Prenotazione aggiornata per il passeggero con CF % sul volo %: % nuovi bagagli inseriti.',
414         p_codf_passeggero, p_codice_volo, p_numero_bagagli;
415 END;
416 $$;

```

Figura 20: Procedure "Modifica prenotazione codF e volo"

4.3.9 Modifica prenotazione codice

```

427 -- Aggiorna bagagli di una prenotazione tramite codice
428 CREATE PROCEDURE public.modifica_prenotazione_codice(IN p_codice_prenotazione integer,
429 IN p_username character varying, IN p_numero_bagagli integer)
430 LANGUAGE plpgsql
431 AS $$
432 DECLARE
433     v_id_biglietto INTEGER;
434 BEGIN
435     SELECT p.id_biglietto
436     INTO v_id_biglietto
437     FROM public."Prenotazione" p
438     WHERE p.id_biglietto = p_codice_prenotazione AND p.username = p_username;
439     IF v_id_biglietto IS NULL THEN
440         RAISE EXCEPTION 'Prenotazione non trovata con codice % per l''utente %', p_codice_prenotazione,
441             p_username;
442     END IF;
443     DELETE FROM public."Bagaglio"
444     WHERE id_biglietto = v_id_biglietto;
445     FOR i IN 1..p_numero_bagagli LOOP
446         INSERT INTO public."Bagaglio" (id_biglietto, stato_bagaglio)
447         VALUES (v_id_biglietto, 'registrato');
448     END LOOP;
449     RAISE NOTICE 'Prenotazione con codice % aggiornata per l''utente %: % nuovi bagagli inseriti.',
450         p_codice_prenotazione, p_username, p_numero_bagagli;
451 END;
452 $$;

```

Figura 21: Procedure "Modifica prenotazione codice"

4.3.10 Prenota volo

```

458 -- Procedura per effettuare una prenotazione con passeggero e bagagli
459 CREATE PROCEDURE public.prenota_volo(IN p_username character varying, IN p_codice_volo integer,
460 IN p_posto_assegnato integer, IN p_codf_passeggero character varying,
461 IN p_nome character varying, IN p_cognome character varying, IN p_data_nascita date,
462 IN p_num_bagagli integer)
463 LANGUAGE plpgsql
464 AS $$
465 DECLARE
466     v_id_biglietto INTEGER;
467     i INTEGER;
468 BEGIN
469     IF NOT EXISTS (
470         SELECT 1
471         FROM public."Utentegenerico"
472         WHERE username = p_username
473     ) THEN
474         RAISE EXCEPTION 'Username % non esistente', p_username;
475     END IF;
476
477     PERFORM 1
478     FROM public."Volo" v
479     WHERE v.codice_volo = p_codice_volo
480           AND v.tipologia = 'inPartenza';
481
482     IF NOT FOUND THEN
483         RAISE EXCEPTION 'Impossibile prenotare: il volo % non è "inPartenza" o non esiste.', p_codice_volo;
484     END IF;
485
486     INSERT INTO public."Passeggero" ("codF", nome, cognome, data_nascita)
487     VALUES (p_codf_passeggero, p_nome, p_cognome, p_data_nascita)
488     ON CONFLICT ("codF") DO NOTHING;
489
490     INSERT INTO public."Prenotazione" (posto_assegnato, stato, id_volo, username, "codF_passeggero")
491     VALUES (p_posto_assegnato, 'confermata', p_codice_volo, p_username, p_codf_passeggero)
492     RETURNING id_biglietto INTO v_id_biglietto;
493
494     FOR i IN 1..p_num_bagagli LOOP
495         INSERT INTO public."Bagaglio" (stato_bagaglio, id_biglietto)
496         VALUES ('registrato', v_id_biglietto);
497     END LOOP;
498
499     RAISE NOTICE
500     'Prenotazione OK: utente %, volo %, posto %, id %, bagagli %.',
501     p_username, p_codice_volo, p_posto_assegnato, v_id_biglietto, p_num_bagagli;
502 END;
503 $$;

```

Figura 22: Procedure "Prenota volo"

4.3.11 Segnala smarrimento bagaglio

```

514 -- Segnala un bagaglio come smarrito, utente solo se possiede quel bagaglio,
515 --admin puo' segnalare tutti i bagagli
516 CREATE PROCEDURE public.segnala_smarrimento_bagaglio(IN p_id_bagaglio integer,
517 IN p_username character varying)
518     LANGUAGE plpgsql
519     AS $$
520 DECLARE
521     v_id_biglietto INTEGER;
522     v_codF_passeggero VARCHAR;
523     v_is_admin BOOLEAN;
524 BEGIN
525     SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
526     INTO v_is_admin;
527
528     IF NOT v_is_admin THEN
529         SELECT p.id_biglietto, p."codF_passeggero"
530         INTO v_id_biglietto, v_codF_passeggero
531         FROM public."Prenotazione" p
532         WHERE p.id_biglietto = (SELECT id_biglietto FROM public."Bagaglio" WHERE id_bagaglio = p_id_bagaglio)
533         AND p.username = p_username;
534
535         IF v_id_biglietto IS NULL THEN
536             RAISE EXCEPTION 'L'utente % non è associato al bagaglio con ID %', p_username, p_id_bagaglio;
537         END IF;
538     ELSE
539         SELECT p.id_biglietto, p."codF_passeggero"
540         INTO v_id_biglietto, v_codF_passeggero
541         FROM public."Prenotazione" p
542         WHERE p.id_biglietto = (SELECT id_biglietto FROM public."Bagaglio" WHERE id_bagaglio = p_id_bagaglio);
543     END IF;
544
545     UPDATE public."Bagaglio"
546     SET stato_bagaglio = 'smarrito'
547     WHERE id_bagaglio = p_id_bagaglio;
548
549     RAISE NOTICE 'Il bagaglio con ID % è stato segnato come smarrito.', p_id_bagaglio;
550 END;
551 $$;

```

Figura 23: Procedure "Segnala smarrimento bagaglio"

4.3.12 Visualizza prenotazioni per codF

```

555 -- Visualizza prenotazioni legate a un codice fiscale
556 CREATE PROCEDURE public.visualizza_prenotazioni_codf(IN p_codf_passeggero character varying)
557 LANGUAGE plpgsql
558 AS $$DECLARE
559     r RECORD;
560     num_bagagli INTEGER;
561     v_out TEXT := '';
562     v_count_rows INTEGER := 0;
563 BEGIN
564     FOR r IN
565         SELECT
566             p.id_biglietto,
567             p.posto_assegnato,
568             p.stato,
569             p.id_volo,
570             p.username,
571             p."codF_passeggero"
572         FROM public."Prenotazione" p
573         JOIN public."Passeggero" ps
574             ON p."codF_passeggero" = ps."codF"
575         WHERE ps."codF" = p_codf_passeggero
576         ORDER BY p.id_biglietto
577     LOOP
578         SELECT COUNT(*)
579             INTO num_bagagli
580         FROM public."Bagaglio" b
581         WHERE b.id_biglietto = r.id_biglietto;
582
583         v_out := v_out ||
584             format(
585                 'ID Biglietto: %s | Posto: %s | Stato: %s | ID Volo: %s | Username: %s | CodF: %s |
586                 Bagagli: %s',
587                 r.id_biglietto, r.posto_assegnato, r.stato, r.id_volo, r.username, r."codF_passeggero",
588                 num_bagagli
589             ) || E'\n';
590
591         v_count_rows := v_count_rows + 1;
592     END LOOP;
593
594     IF v_count_rows = 0 THEN
595         RAISE NOTICE 'Nessuna prenotazione trovata per il Codice Fiscale: %', p_codf_passeggero;
596     ELSE
597         RAISE NOTICE '%', v_out;
598     END IF;
599 END;
600 $$;

```

Figura 24: Procedure "Visualizza prenotazioni per codF"

4.3.13 Visualizza prenotazioni per volo

```

606 -- Mostra tutte le prenotazioni relative a un volo
607 CREATE PROCEDURE public.visualizza_prenotazioni_per_volo(IN p_id_volo integer,
608 IN p_username character varying)
609     LANGUAGE plpgsql
610     AS $$
611 DECLARE
612     result RECORD;
613     num_bagagli INTEGER;
614     v_is_admin BOOLEAN;
615 BEGIN
616     SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
617     INTO v_is_admin;
618
619     IF NOT v_is_admin THEN
620         FOR result IN
621             SELECT
622                 p.id_biglietto,
623                 p.posto_assegnato,
624                 p.stato,
625                 p.id_volo,
626                 p.username,
627                 p."codF_passeggero"
628             FROM
629                 public."Prenotazione" p
630             WHERE
631                 p.id_volo = p_id_volo AND p.username = p_username
632         LOOP
633             SELECT COUNT(*) INTO num_bagagli
634             FROM public."Bagaglio" b
635             WHERE b.id_biglietto = result.id_biglietto;
636             RAISE NOTICE 'ID Biglietto: %, Posto Assegnato: %, Stato: %, ID Volo: %, Username: %,
637             Codice Fiscale Passeggero: %, Numero Bagagli: %',
638             result.id_biglietto, result.posto_assegnato, result.stato,
639             result.id_volo, result.username, result."codF_passeggero", num_bagagli;
640         END LOOP;
641     ELSE
642         FOR result IN
643             SELECT
644                 p.id_biglietto,
645                 p.posto_assegnato,
646                 p.stato,
647                 p.id_volo,
648                 p.username,
649                 p."codF_passeggero"
650             FROM
651                 public."Prenotazione" p
652             WHERE
653                 p.id_volo = p_id_volo
654         LOOP
655             SELECT COUNT(*) INTO num_bagagli
656             FROM public."Bagaglio" b
657             WHERE b.id_biglietto = result.id_biglietto;
658             RAISE NOTICE 'ID Biglietto: %, Posto Assegnato: %, Stato: %, ID Volo: %, Username: %,
659             Codice Fiscale Passeggero: %, Numero Bagagli: %',
660             result.id_biglietto, result.posto_assegnato, result.stato, result.id_volo,
661             result.username, result."codF_passeggero", num_bagagli;
662         END LOOP;
663     END IF;
664 END;
665 $$;

```

Figura 25: Procedure "Visualizza prenotazioni per volo"

4.3.14 Visualizza smarrimenti

```

670  -- Elenco bagagli smarriti (solo admin)
671  CREATE PROCEDURE public.visualizza_smarrimenti(IN p_username character varying)
672  LANGUAGE plpgsql
673  AS $$
674  DECLARE
675  v_is_admin BOOLEAN;
676  result RECORD;
677  BEGIN
678  SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
679  INTO v_is_admin;
680
681  IF NOT v_is_admin THEN
682  RAISE EXCEPTION 'L'utente % non è un amministratore.', p_username;
683  END IF;
684
685  FOR result IN
686  SELECT
687  b.id_bagaglio,
688  b.stato_bagaglio,
689  b.id_biglietto,
690  p.posto_assegnato,
691  v.codice_volo,
692  v.compagnia,
693  v.orario_previsto,
694  v.origine,
695  v.destinazione
696  FROM
697  public."Bagaglio" b
698  JOIN
699  public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
700  JOIN
701  public."Volo" v ON p.id_volo = v.codice_volo
702  WHERE
703  b.stato_bagaglio = 'smarrito'
704  LOOP
705  RAISE NOTICE 'ID Bagaglio: %, Stato: %, Posto Assegnato: %, Codice Volo: %, Compagnia: %,
706  Orario: %, Origine: %, Destinazione: %',
707  result.id_bagaglio, result.stato_bagaglio, result.posto_assegnato, result.codice_volo,
708  result.compagnia, result.orario_previsto, result.origine, result.destinazione;
709  END LOOP;
710  END;
711  $$;

```

Figura 26: Procedure "Visualizza smarrimenti"

4.3.15 Visualizza voli

```

716 CREATE PROCEDURE public.visualizza_voli() -- Elenco di tutti i voli
717 LANGUAGE plpgsql
718 AS $$
719 DECLARE
720     result RECORD;
721 BEGIN
722     FOR result IN
723         SELECT
724             v.codice_volo,
725             v.compagnia,
726             v.data,
727             v.orario_previsto,
728             v.ritardo,
729             v.stato,
730             v.origine,
731             v.destinazione,
732             v.tipologia,
733             v.gate
734         FROM public."Volo" v
735         ORDER BY v.codice_volo
736     LOOP
737         RAISE NOTICE
738             'Codice Volo: %, Compagnia: %, Data: %, Orario: %, Ritardo: %, Stato: %, Origine: %,
739             Destinazione: %, Tipologia: %, Gate: %',
740             result.codice_volo,
741             result.compagnia,
742             result.data,
743             result.orario_previsto,
744             result.ritardo,
745             result.stato,
746             result.origine,
747             result.destinazione,
748             result.tipologia,
749             result.gate;
750     END LOOP;
751 END;
752 $$;

```

Figura 27: Procedure "Visualizza voli"