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

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.

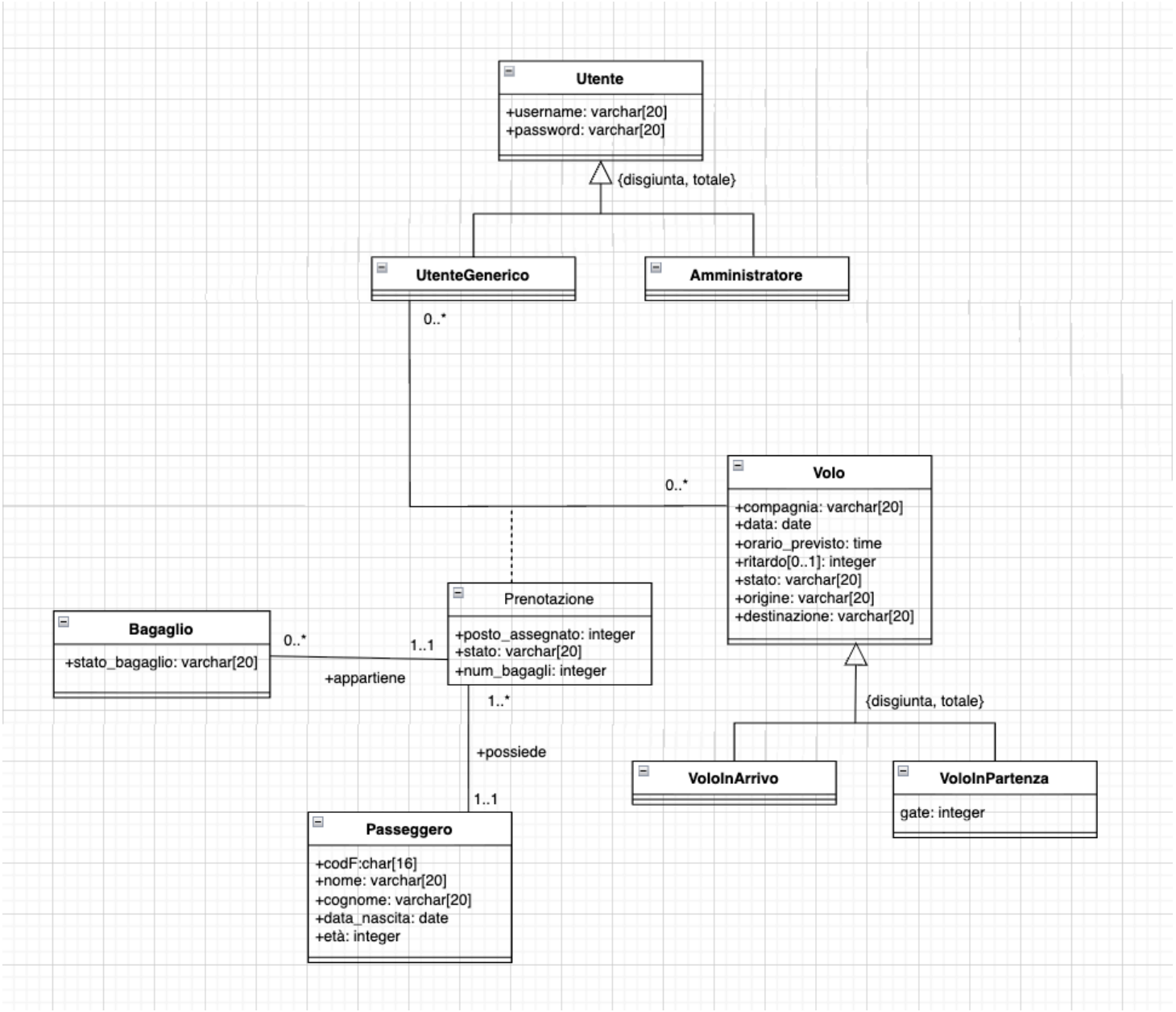
1.1 Descrizione Del Problema

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 entitàin 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:

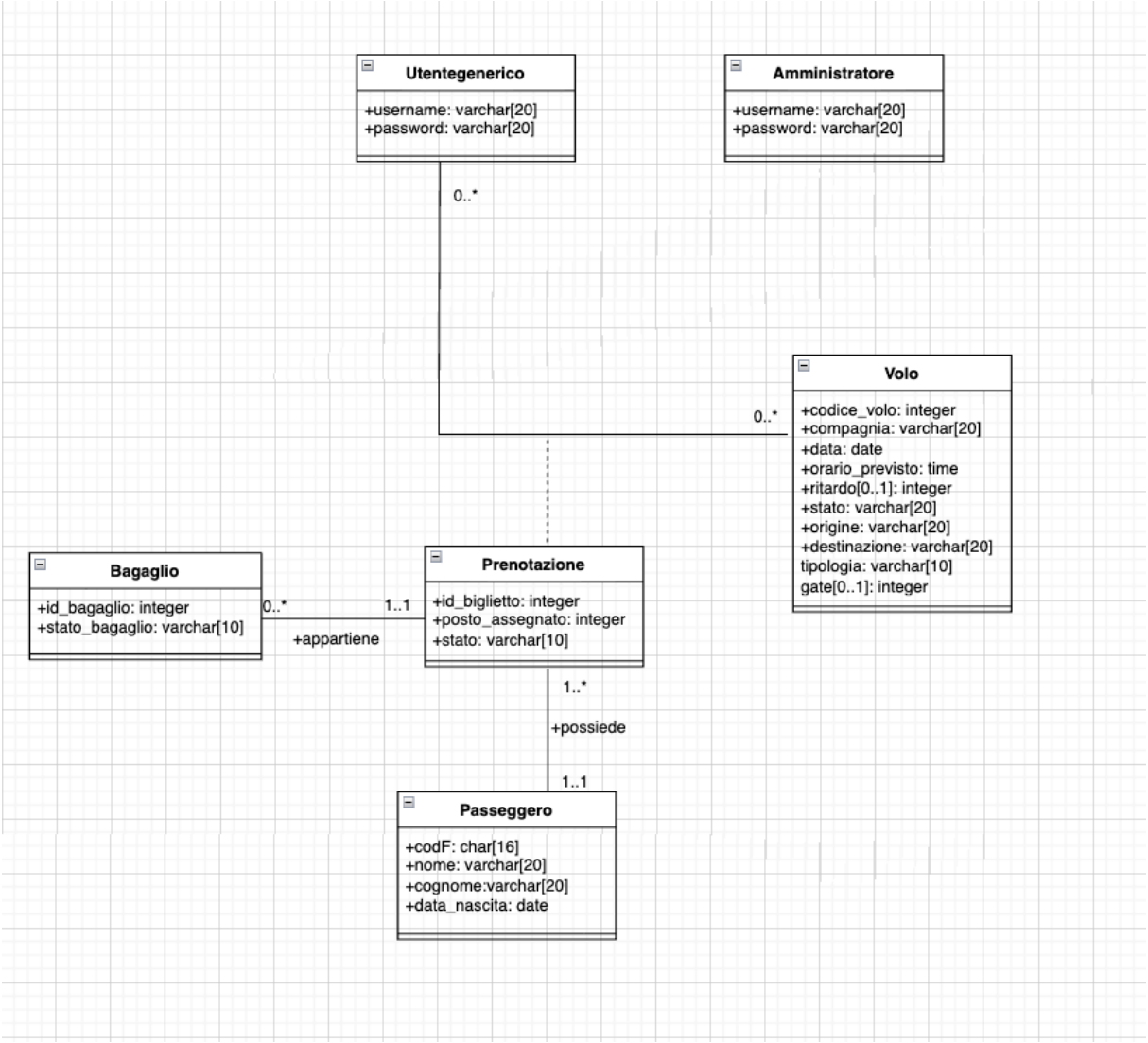
Utente → UtenteGenerico / Amministratore

Volo → VoloInArrivo / VoloInPartenza

Per quanto riguarda la gerarchia Utente → 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 → 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
Utentegenerico	Rappresenta un utente registrato del sistema.	• **username** (varchar[20], PK): Nome utente. • **password** (varchar[20]): Credenziali di accesso.
Amministratore	Rappresenta un utente con privilegi amministrativi.	• **username** (varchar[20], PK): Nome utente. • **password** (varchar[20]): Credenziali di accesso.
Volo	Descrive un volo gestito dal sistema.	• **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.	• **id_bagaglio** (integer, PK): Identificativo univoco del bagaglio. • **statoBagaglio** (varchar[10]): Stato attuale del bagaglio.
Prenotazione	Descrive una singola prenotazione di un biglietto.	• **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.	• **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.	• **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.	• **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.	• **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

```sql
-- ============================================
-- TABELLA: Amministratore
-- ============================================

-- Definizione tabella


CREATE TABLE public."Amministratore" (
    username character varying(20) NOT NULL,
    password character varying(20) NOT NULL
);


ALTER TABLE public."Amministratore" OWNER TO postgres;

-- Vincoli (PK/UNIQUE)

ALTER TABLE ONLY public."Amministratore"
    ADD CONSTRAINT pk_amministratore PRIMARY KEY (username);
```

Figura 1: Codice SQL della tabella `AMMINISTRATORE`

4.1.2 Definizione della Tabella BAGAGLIO

```sql
-- ========================================
-- TABELLA: Bagaglio
-- ========================================

-- Sequenze correlate


CREATE SEQUENCE public.id_bagaglio_seq
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER SEQUENCE public.id_bagaglio_seq OWNER TO postgres;

-- Definizione tabella


CREATE TABLE public."Bagaglio" (
    id_bagaglio integer DEFAULT nextval('public.id_bagaglio_seq'::regclass) NOT NULL,
    stato_bagaglio character varying(10) NOT NULL,
    id_biglietto integer NOT NULL,
    CONSTRAINT ck_stato CHECK ((((stato_bagaglio)::text = 'registrato'::text) OR
    ((stato_bagaglio)::text = 'caricato'::text) OR
    ((stato_bagaglio)::text = 'ritiro'::text) OR
    ((stato_bagaglio)::text = 'smarrito'::text)))
);


ALTER TABLE public."Bagaglio" OWNER TO postgres;

-- Vincoli (PK/UNIQUE)


ALTER TABLE ONLY public."Bagaglio"
    ADD CONSTRAINT "Bagaglio_pkey" PRIMARY KEY (id_bagaglio);


-- Vincoli di chiave esterna


ALTER TABLE ONLY public."Bagaglio"
    ADD CONSTRAINT fk_bagaglio_prenotazione FOREIGN KEY (id_biglietto) REFERENCES
    public."Prenotazione"(id_biglietto) ON DELETE CASCADE;

```

Figura 2: Codice SQL della tabella `BAGAGLIO`

4.1.3 Definizione della Tabella PASSEGGERO

```sql
-- ========================================
-- TABELLA: Passeggero
-- ========================================

-- Definizione tabella


CREATE TABLE public."Passeggero" (
    "codF" character varying(16) NOT NULL,
    nome character varying(20) NOT NULL,
    cognome character varying(20) NOT NULL,
    data_nascita date NOT NULL,
    CONSTRAINT ck_cf CHECK ((("codF")::text = upper(("codF")::text)) AND (length(("codF")::text) = 16))),
    CONSTRAINT ck_data CHECK (((data_nascita > '1900-01-01'::date) AND (data_nascita < CURRENT_DATE)))
);


ALTER TABLE public."Passeggero" OWNER TO postgres;

-- Vincoli (PK/UNIQUE)

ALTER TABLE ONLY public."Passeggero"
    ADD CONSTRAINT "Passeggero_pkey" PRIMARY KEY ("codF");

```

Figura 3: Codice SQL della tabella `PASSEGERRO`

4.1.4 Definizione della Tabella PRENOTAZIONE

```sql
-- ========================================
-- TABELLA: Prenotazione
-- ========================================

-- Sequenze correlate

CREATE SEQUENCE public."Prenotazione_id_biglietto_seq"
    AS integer
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1;


ALTER SEQUENCE public."Prenotazione_id_biglietto_seq" OWNER TO postgres;

-- Definizione tabella

CREATE TABLE public."Prenotazione" (
    id_biglietto integer NOT NULL,
    posto_assegnato integer NOT NULL,
    stato character varying(10) NOT NULL,
    id_volo integer NOT NULL,
    username character varying(20) NOT NULL,
    "codF_passeggero" character varying(16) NOT NULL,
    CONSTRAINT ck_stato CHECK ((((stato)::text = 'confermata'::text) OR
    ((stato)::text = 'inAttesa'::text) OR ((stato)::text = 'cancellata'::text)))
);


ALTER TABLE public."Prenotazione" OWNER TO postgres;

-- Default/ALTER COLUMN


ALTER TABLE ONLY public."Prenotazione" ALTER COLUMN id_biglietto SET DEFAULT
nextval('public."Prenotazione_id_biglietto_seq"'::regclass);


-- Vincoli (PK/UNIQUE)


ALTER TABLE ONLY public."Prenotazione"
    ADD CONSTRAINT "Prenotazione_pkey" PRIMARY KEY (id_biglietto);

ALTER TABLE ONLY public."Prenotazione"
    ADD CONSTRAINT uq_prenotazione UNIQUE (id_volo, "codF_passeggero");

-- Vincoli di chiave esterna

ALTER TABLE ONLY public."Prenotazione"
    ADD CONSTRAINT fk_prenotazione_passeggero FOREIGN KEY ("codF_passeggero") REFERENCES
    public."Passeggero"("codF") ON DELETE CASCADE;

ALTER TABLE ONLY public."Prenotazione"
    ADD CONSTRAINT fk_prenotazione_username FOREIGN KEY (username) REFERENCES
    public."Utentegenerico"(username) ON DELETE CASCADE;

ALTER TABLE ONLY public."Prenotazione"
    ADD CONSTRAINT fk_prenotazione_volo FOREIGN KEY (id_volo) REFERENCES
    public."Volo"(codice_volo) ON DELETE CASCADE;
```

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

```sql
-- Trigger (raggruppati qui per documentazione; ATTIVATI a fine file)

CREATE TRIGGER trg_check_posto_occupato BEFORE INSERT ON public."Prenotazione" FOR EACH ROW
EXECUTE FUNCTION public.check_posto_occupato();  -- Controlla che il posto non sia già occupato

CREATE TRIGGER trg_check_volo_stato_per_prenotazione BEFORE INSERT ON public."Prenotazione" FOR EACH ROW
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

```sql
-- ========================================
-- TABELLA: Utentegenerico
-- ========================================

-- Definizione tabella

CREATE TABLE public."Utentegenerico" (
    username character varying(20) NOT NULL,
    password character varying(20) NOT NULL
);


ALTER TABLE public."Utentegenerico" OWNER TO postgres;

-- Vincoli (PK/UNIQUE)

ALTER TABLE ONLY public."Utentegenerico"
    ADD CONSTRAINT pk_utentegenerico PRIMARY KEY (username);

```

Figura 7: Codice SQL della tabella `UTENTEGENERICO`

4.1.6 Definizione della Tabella VOLO

```sql
-- ========================================
-- TABELLA: Volo
-- ========================================

-- Definizione tabella


CREATE TABLE public."Volo" (
    codice_volo integer NOT NULL,
    compagnia character varying(20) NOT NULL,
    data date NOT NULL,
    orario_previsto time with time zone NOT NULL,
    ritardo integer DEFAULT 0,
    stato character varying NOT NULL,
    origine character varying(20) NOT NULL,
    destinazione character varying(20) NOT NULL,
    tipologia character varying(10) NOT NULL,
    gate integer,
    CONSTRAINT ck_ritardo CHECK ((ritardo >= 0)),

    CONSTRAINT ck_stato CHECK ((((stato)::text = 'programmato'::text) OR

    ((stato)::text = 'decollato'::text) OR ((stato)::text = 'inRitardo'::text)
    OR ((stato)::text = 'atterrato'::text) OR ((stato)::text = 'cancellato'::text))),

    CONSTRAINT ck_tipo CHECK ((((tipologia)::text = 'inPartenza'::text) OR
    ((tipologia)::text = 'inArrivo'::text)))
);


ALTER TABLE public."Volo" OWNER TO postgres;

-- Vincoli (PK/UNIQUE)


ALTER TABLE ONLY public."Volo"
    ADD CONSTRAINT "Volo_pkey" PRIMARY KEY (codice_volo);

-- Trigger (raggruppati qui per documentazione)

CREATE TRIGGER tg_ck_data BEFORE INSERT OR UPDATE OF tipologia, data ON public."Volo" FOR EACH ROW
EXECUTE FUNCTION public.fn_ck_data();  -- Controlla coerenza tra stato e data volo

CREATE TRIGGER tg_ck_gate BEFORE INSERT OR UPDATE OF gate, data, orario_previsto ON public."Volo" FOR EACH ROW
EXECUTE FUNCTION public.fn_ck_gate();  -- Verifica correttezza e unicità del gate

CREATE TRIGGER tg_ck_volo_tipo BEFORE INSERT ON public."Volo" FOR EACH ROW
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

```sql
-- ========================================
-- SEZIONE TRIGGER FUNCTION
-- ========================================

-- Trigger Function per controllare posto duplicato
CREATE FUNCTION public.check_posto_occupato() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM public."Prenotazione"
        WHERE id_volo = NEW.id_volo AND posto_assegnato = NEW.posto_assegnato
    ) THEN
        RAISE EXCEPTION 'Il posto % sul volo % è già prenotato', NEW.posto_assegnato, NEW.id_volo;
    END IF;

    RETURN NEW;
END;
$$;

ALTER FUNCTION public.check_posto_occupato() OWNER TO postgres;

-- Trigger function per controllare stato volo che si sta prenotando
CREATE FUNCTION public.check_volo_stato_per_prenotazione() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM public."Volo"
        WHERE codice_volo = NEW.id_volo
        AND (stato = 'inRitardo' OR stato = 'programmato')
    ) THEN
        RAISE EXCEPTION 'Non puoi prenotare un volo che non è in partenza o è stato cancellato o decollato';
    END IF;

    RETURN NEW;
END;
$$;

```

Figura 9: Trigger function "check_posto_occupato"

4.2.2 check_volo_stato_per_prenotazione

```
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 function "check_volo_stato_per_prenotazione"

4.2.3 fn_ck_data

```
ALTER FUNCTION public.check_volo_stato_per_prenotazione() OWNER TO postgres;

CREATE FUNCTION public.fn_ck_data() RETURNS trigger  -- Trigger function che controlla coerenza stato/data
    LANGUAGE plpgsql
    AS $$
BEGIN
  IF NEW.stato IS NULL OR NEW.data IS NULL THEN
    RAISE EXCEPTION 'stato e data non possono essere NULL';
  END IF;

  IF NEW.stato IN ('cancellato','decollato') THEN
    IF NEW.data::date > CURRENT_DATE THEN
      RAISE EXCEPTION 'Volo con stato % deve avere data <= oggi (trovato: %)',
                      NEW.stato, NEW.data::date;
    END IF;
  ELSE
    IF NEW.data::date < CURRENT_DATE THEN
      RAISE EXCEPTION 'Volo con stato % deve avere data >= oggi (trovato: %)',
                      NEW.stato, NEW.data::date;
    END IF;
  END IF;

  RETURN NEW;
END;
$$;


ALTER FUNCTION public.fn_ck_data() OWNER TO postgres;

CREATE FUNCTION public.fn_ck_gate() RETURNS trigger  -- Trigger function che controlla disponibilità del gate
    LANGUAGE plpgsql
    AS $$
DECLARE
  altro_volo integer;
BEGIN
  -- gate considerato assegnato solo se > 0
  IF NEW.gate IS NOT NULL AND NEW.gate > 0 THEN
    IF NEW.tipologia <> 'inPartenza' THEN
      RAISE EXCEPTION 'Il gate può essere assegnato solo ai voli in partenza';
    END IF;

    -- stesso gate, stessa data e stesso orario non possono avere due voli diversi
    SELECT v.codice_volo
      INTO altro_volo
      FROM public."Volo" v
     WHERE v.gate = NEW.gate
       AND v.data = NEW.data
       AND v.orario_previsto = NEW.orario_previsto
       AND v.codice_volo <> NEW.codice_volo
     LIMIT 1;

    IF altro_volo IS NOT NULL THEN
      RAISE EXCEPTION 'Gate % già assegnato per % alle % al volo %',
        NEW.gate, NEW.data, NEW.orario_previsto, altro_volo;
    END IF;
  END IF;

  RETURN NEW;
END;
$$;
```

Figura 11: Trigger function "fn_ck_data"

4.2.4 fn_ck_gate

```
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 12: Trigger function "fn_ck_gate"

4.2.5 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 13: 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

```sql
-- ========================================
-- SEZIONE PROCEDURE
-- ========================================

CREATE PROCEDURE public.cerca_bagagli_per_prenotazione(IN p_codice_prenotazione integer,
IN p_username character varying)  -- Cerca i bagagli legati a una prenotazione
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_id_biglietto INTEGER;
    v_codF_passeggero VARCHAR;
    v_is_admin BOOLEAN;
    result RECORD;
BEGIN
    SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
    INTO v_is_admin;
    IF NOT v_is_admin THEN
        SELECT p.id_biglietto, p."codF_passeggero"
        INTO v_id_biglietto, v_codF_passeggero
        FROM public."Prenotazione" p
        WHERE p.id_biglietto = p_codice_prenotazione AND p.username = p_username;

        IF v_id_biglietto IS NULL THEN
            RAISE EXCEPTION 'Prenotazione con codice % non trovata per l''utente %', p_codice_prenotazione,
            p_username;
        END IF;

        IF v_codF_passeggero IS NULL THEN
            RAISE EXCEPTION 'Il codice fiscale del passeggero non è associato alla prenotazione.';
        END IF;
    ELSE
        SELECT p.id_biglietto, p."codF_passeggero"
        INTO v_id_biglietto, v_codF_passeggero
        FROM public."Prenotazione" p
        WHERE p.id_biglietto = p_codice_prenotazione;
    END IF;
    FOR result IN
        SELECT
            b.id_bagaglio, b.stato_bagaglio, b.id_biglietto,
            p.posto_assegnato,
            v.codice_volo,
            v.compagnia,
            v.orario_previsto,
            v.origine,
            v.destinazione
        FROM
            public."Bagaglio" b
        JOIN
            public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
        JOIN
            public."Volo" v ON p.id_volo = v.codice_volo
        WHERE
            b.id_biglietto = v_id_biglietto
    LOOP
        RAISE NOTICE 'ID Bagaglio: %, Stato: %, Posto Assegnato: %, Codice Volo: %, Compagnia: %,
        Orario: %, Origine: %, Destinazione: %',
            result.id_bagaglio, result.stato_bagaglio, result.posto_assegnato, result.codice_volo,
            result.compagnia, result.orario_previsto, result.origine, result.destinazione;
    END LOOP;
END;
$$;
```

Figura 14: Procedure "Cerca bagagli per prenotazione"

4.3.2 Cerca bagagli per utente

```
-- Mostra bagagli per utente (o tutti se admin)
CREATE PROCEDURE public.cerca_bagagli_per_utente(IN username_input character varying)
    LANGUAGE plpgsql
    AS $$DECLARE
    result RECORD;
    bagaglio_cursor REFCURSOR;
    v_is_admin BOOLEAN;
    v_username varchar := username_input;
BEGIN

    SELECT EXISTS(
        SELECT 1
        FROM public."Amministratore" a
        WHERE a.username = v_username
    )
    INTO v_is_admin;

    IF v_is_admin THEN

        OPEN bagaglio_cursor FOR
            SELECT
                b.id_bagaglio,
                p.id_biglietto AS numero_prenotazione,
                ps.nome AS nome_passeggero,
                ps.cognome AS cognome_passeggero,
                p.username
            FROM public."Bagaglio" b
            JOIN public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
            JOIN public."Passeggero" ps ON p."codF_passeggero" = ps."codF"
            ORDER BY b.id_bagaglio;
    ELSE

        OPEN bagaglio_cursor FOR
            SELECT
                b.id_bagaglio,
                p.id_biglietto AS numero_prenotazione,
                ps.nome AS nome_passeggero,
                ps.cognome AS cognome_passeggero,
                p.username
            FROM public."Bagaglio" b
            JOIN public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
            JOIN public."Passeggero" ps ON p."codF_passeggero" = ps."codF"
            WHERE p.username = v_username
            ORDER BY b.id_bagaglio;
    END IF;
    LOOP
        FETCH bagaglio_cursor INTO result;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE
            'ID Bagaglio: %, Prenotazione: %, Nome: %, Cognome: %, Username: %',
            result.id_bagaglio,
            result.numero_prenotazione,
            result.nome_passeggero,
            result.cognome_passeggero,
            result.username;
    END LOOP;

    CLOSE bagaglio_cursor;
END;
$$;
```

Figura 15: Procedure "Cerca bagagli per utente"

4.3.3 Cerca bagaglio per id

```
-- Cerca dettagli di un bagaglio tramite ID
CREATE PROCEDURE public.cerca_bagaglio_per_id(IN p_id_bagaglio integer, IN p_username character varying)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_id_biglietto INTEGER;
    v_is_admin BOOLEAN;
    result RECORD;
BEGIN
    SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
    INTO v_is_admin;

    IF NOT v_is_admin THEN
        SELECT p.id_biglietto
        INTO v_id_biglietto
        FROM public."Prenotazione" p
        WHERE p.id_biglietto = (SELECT id_biglietto FROM public."Bagaglio" WHERE id_bagaglio = p_id_bagaglio)
          AND p.username = p_username;

        IF v_id_biglietto IS NULL THEN
            RAISE EXCEPTION 'L''utente % non è associato a nessun bagaglio con ID %', p_username,
            p_id_bagaglio;
        END IF;
    ELSE
        SELECT p.id_biglietto
        INTO v_id_biglietto
        FROM public."Prenotazione" p
        WHERE p.id_biglietto = (SELECT id_biglietto FROM public."Bagaglio" WHERE id_bagaglio = p_id_bagaglio);
    END IF;

    FOR result IN
        SELECT
            b.id_bagaglio,
            b.stato_bagaglio,
            b.id_biglietto,
            p.posto_assegnato,
            v.codice_volo,
            v.compagnia,
            v.orario_previsto,
            v.origine,
            v.destinazione
        FROM
            public."Bagaglio" b
        JOIN
            public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
        JOIN
            public."Volo" v ON p.id_volo = v.codice_volo
        WHERE
            b.id_bagaglio = p_id_bagaglio
    LOOP
        RAISE NOTICE 'ID Bagaglio: %, Stato: %, Posto Assegnato: %, Codice Volo: %, Compagnia: %,
        Orario: %, Origine: %, Destinazione: %',
            result.id_bagaglio, result.stato_bagaglio, result.posto_assegnato, result.codice_volo,
            result.compagnia, result.orario_previsto, result.origine, result.destinazione;
    END LOOP;
END;
$$;
```

Figura 16: Procedure "Cerca bagaglio per id"

4.3.4 Cerca passeggero per id prenotazione

```sql
-- Visualizza i dati passeggero legato a prenotazione
CREATE PROCEDURE public.cerca_passeggero_per_id_prenotazione(IN p_id_prenotazione integer,
IN p_username character varying)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_codF_passeggero VARCHAR;
    v_is_admin BOOLEAN;
    result RECORD;
BEGIN
    SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
    INTO v_is_admin;

    IF v_is_admin THEN
        SELECT p."codF_passeggero"
        INTO v_codF_passeggero
        FROM public."Prenotazione" p
        WHERE p.id_biglietto = p_id_prenotazione;
    ELSE
        SELECT p."codF_passeggero"
        INTO v_codF_passeggero
        FROM public."Prenotazione" p
        WHERE p.id_biglietto = p_id_prenotazione AND p.username = p_username;

        IF v_codF_passeggero IS NULL THEN
            RAISE EXCEPTION 'Prenotazione con ID % non trovata per l''utente %',
            p_id_prenotazione, p_username;
        END IF;
    END IF;

    FOR result IN
        SELECT
            ps."codF",
            ps.nome,
            ps.cognome,
            ps.data_nascita
        FROM
            public."Passeggero" ps
        WHERE
            ps."codF" = v_codF_passeggero
    LOOP
        RAISE NOTICE 'Codice Fiscale: %, Nome: %, Cognome: %, Data Nascita: %',
            result."codF", result.nome, result.cognome, result.data_nascita;
    END LOOP;
END;
$$;
```

Figura 17: Procedure "Cerca passeggero per id prenotazione"

4.3.5 Cerca volo

```
CREATE PROCEDURE public.cerca_volo(IN p_codice_volo integer)  -- Visualizza le info di un volo specifico
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_codice_volo INTEGER;
    v_compagnia VARCHAR;
    v_data DATE;
    v_orario_previsto TIME WITH TIME ZONE;
    v_ritardo INTEGER;
    v_stato VARCHAR;
    v_origine VARCHAR;
    v_destinazione VARCHAR;
    v_tipologia VARCHAR;
    v_gate INTEGER;
BEGIN

    SELECT
        codice_volo,
        compagnia,
        data,
        orario_previsto,
        ritardo,
        stato,
        origine,
        destinazione,
        tipologia,
        gate
    INTO
        v_codice_volo, v_compagnia, v_data, v_orario_previsto, v_ritardo, v_stato,
        v_origine, v_destinazione, v_tipologia, v_gate
    FROM
        public."Volo"
    WHERE
        codice_volo = p_codice_volo;
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Volo con codice % non trovato.', p_codice_volo;
    END IF;
    RAISE NOTICE 'Codice Volo: %, Compagnia: %, Data: %, Orario: %, Ritardo: %, Stato: %, Origine: %,
    Destinazione: %, Tipologia: %, Gate: %',
        v_codice_volo, v_compagnia, v_data, v_orario_previsto, v_ritardo, v_stato,
        v_origine, v_destinazione, v_tipologia, v_gate;

END;
$$;
```

Figura 18: Procedure "Cerca volo"

4.3.6 Elimina prenotazione

```
-- Procedura che elimina una prenotazione (bagagli rimossi in cascata)
CREATE PROCEDURE public.elimina_prenotazione(IN p_id_prenotazione integer, IN p_username character varying)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_is_admin   boolean;
    v_owner_ok   boolean;
BEGIN
    -- 1) Admin?
    SELECT EXISTS(
        SELECT 1
        FROM public."Amministratore"
        WHERE username = p_username
    ) INTO v_is_admin;

    -- 2) Esiste la prenotazione?
    IF NOT EXISTS (
        SELECT 1
        FROM public."Prenotazione"
        WHERE id_biglietto = p_id_prenotazione
    ) THEN
        RAISE EXCEPTION 'Prenotazione con ID % non esiste.', p_id_prenotazione;
    END IF;

    -- 3) Se non admin, verifico che appartenga all'utente
    IF NOT v_is_admin THEN
        SELECT EXISTS(
            SELECT 1
            FROM public."Prenotazione"
            WHERE id_biglietto = p_id_prenotazione
              AND username = p_username
        ) INTO v_owner_ok;

        IF NOT v_owner_ok THEN
            RAISE EXCEPTION
                'Operazione negata: la prenotazione % non appartiene all''utente %.',
                p_id_prenotazione, p_username;
        END IF;
    END IF;

    -- 4) Elimino (i bagagli collegati vengono rimossi via ON DELETE CASCADE)
    DELETE FROM public."Prenotazione"
    WHERE id_biglietto = p_id_prenotazione;

    IF NOT FOUND THEN
        -- teoricamente non ci arrivi perché abbiamo già verificato l'esistenza
        RAISE EXCEPTION 'Eliminazione fallita per la prenotazione %.', p_id_prenotazione;
    END IF;

    RAISE NOTICE
        'Prenotazione % eliminata con successo da % (admin=%).',
        p_id_prenotazione, p_username, v_is_admin;
END;
$$;
```

Figura 19: Procedure "Elimina prenotazione"

4.3.7 Modifica gate volo

```
CREATE PROCEDURE public.modifica_gate_volo(IN p_username character varying, IN p_codice_volo integer,
IN p_nuovo_gate integer)  -- Modifica il gate (solo admin, voli in partenza)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_is_admin BOOLEAN;
    v_stato_volo VARCHAR;
BEGIN

    SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
    INTO v_is_admin;

    IF NOT v_is_admin THEN
        RAISE EXCEPTION 'L''utente % non è un amministratore e non può eseguire questa operazione.', p_username
    END IF;

    SELECT tipologia INTO v_stato_volo
    FROM public."Volo"
    WHERE codice_volo = p_codice_volo;

    IF v_stato_volo IS NULL THEN
        RAISE EXCEPTION 'Il volo con codice % non esiste.', p_codice_volo;
    ELSIF v_stato_volo != 'inPartenza' THEN
        RAISE EXCEPTION 'Il volo % non è in partenza, quindi non può essere modificato.', p_codice_volo;
    END IF;
    IF EXISTS (SELECT 1 FROM public."Volo" WHERE gate = p_nuovo_gate AND codice_volo != p_codice_volo) THEN
        RAISE EXCEPTION 'Il gate % è già occupato da un altro volo.', p_nuovo_gate;
    END IF;
    UPDATE public."Volo"
    SET gate = p_nuovo_gate
    WHERE codice_volo = p_codice_volo;
    RAISE NOTICE 'Gate del volo % modificato con successo a %.', p_codice_volo, p_nuovo_gate;

END;
$$;
```

Figura 20: Procedure "Modifica gate volo"

4.3.8 Modifica prenotazione codF e volo

```
-- Aggiorna bagagli tramite CF passeggero e volo
CREATE PROCEDURE public.modifica_prenotazione_codf_e_volo(IN p_codf_passeggero character varying,
IN p_codice_volo integer, IN p_username character varying, IN p_numero_bagagli integer)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_id_biglietto INTEGER;
BEGIN
    SELECT p.id_biglietto
    INTO v_id_biglietto
    FROM public."Prenotazione" p
    WHERE p."codF_passeggero" = p_codF_passeggero
      AND p.id_volo = p_codice_volo
      AND p.username = p_username;

    IF v_id_biglietto IS NULL THEN
        RAISE EXCEPTION 'Prenotazione non trovata per il passeggero con CF % sul volo % per l''utente %',
            p_codF_passeggero, p_codice_volo, p_username;
    END IF;

    DELETE FROM public."Bagaglio"
    WHERE id_biglietto = v_id_biglietto;

    FOR i IN 1..p_numero_bagagli LOOP
        INSERT INTO public."Bagaglio" (id_biglietto, stato_bagaglio)
        VALUES (v_id_biglietto, 'registrato');
    END LOOP;
    RAISE NOTICE 'Prenotazione aggiornata per il passeggero con CF % sul volo %: % nuovi bagagli inseriti.',
                 p_codF_passeggero, p_codice_volo, p_numero_bagagli;
END;
$$;
```

Figura 21: Procedure "Modifica prenotazione codF e volo"

4.3.9 Modifica prenotazione codice

```
-- Aggiorna bagagli di una prenotazione tramite codice
CREATE PROCEDURE public.modifica_prenotazione_codice(IN p_codice_prenotazione integer,
IN p_username character varying, IN p_numero_bagagli integer)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_id_biglietto INTEGER;
BEGIN
    SELECT p.id_biglietto
    INTO v_id_biglietto
    FROM public."Prenotazione" p
    WHERE p.id_biglietto = p_codice_prenotazione AND p.username = p_username;
    IF v_id_biglietto IS NULL THEN
        RAISE EXCEPTION 'Prenotazione non trovata con codice % per l''utente %', p_codice_prenotazione,
        p_username;
    END IF;
    DELETE FROM public."Bagaglio"
    WHERE id_biglietto = v_id_biglietto;
    FOR i IN 1..p_numero_bagagli LOOP
        INSERT INTO public."Bagaglio" (id_biglietto, stato_bagaglio)
        VALUES (v_id_biglietto, 'registrato');
    END LOOP;
    RAISE NOTICE 'Prenotazione con codice % aggiornata per l''utente %: % nuovi bagagli inseriti.',
                 p_codice_prenotazione, p_username, p_numero_bagagli;
END;
$$;
```

Figura 22: Procedure "Modifica prenotazione codice"

4.3.10 Prenota volo

```
-- Procedura per effettuare una prenotazione con passeggero e bagagli
CREATE PROCEDURE public.prenota_volo(IN p_username character varying, IN p_codice_volo integer,
IN p_posto_assegnato integer, IN p_codf_passeggero character varying,
IN p_nome character varying, IN p_cognome character varying, IN p_data_nascita date,
IN p_num_bagagli integer)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_id_biglietto INTEGER;
    i INTEGER;
BEGIN

    IF NOT EXISTS (
        SELECT 1
        FROM public."Utentegenerico"
        WHERE username = p_username
    ) THEN
        RAISE EXCEPTION 'Username % non esistente', p_username;
    END IF;

    PERFORM 1
    FROM public."Volo" v
    WHERE v.codice_volo = p_codice_volo
      AND v.tipologia = 'inPartenza';
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Impossibile prenotare: il volo % non è "inPartenza" o non esiste.', p_codice_volo;
    END IF;

    INSERT INTO public."Passeggero" ("codF", nome, cognome, data_nascita)
    VALUES (p_codf_passeggero, p_nome, p_cognome, p_data_nascita)
    ON CONFLICT ("codF") DO NOTHING;

    INSERT INTO public."Prenotazione" (posto_assegnato, stato, id_volo, username, "codF_passeggero")
    VALUES (p_posto_assegnato, 'confermata', p_codice_volo, p_username, p_codf_passeggero)
    RETURNING id_biglietto INTO v_id_biglietto;

    FOR i IN 1..p_num_bagagli LOOP
        INSERT INTO public."Bagaglio" (stato_bagaglio, id_biglietto)
        VALUES ('registrato', v_id_biglietto);
    END LOOP;

    RAISE NOTICE
        'Prenotazione OK: utente %, volo %, posto %, id %, bagagli %.',
        p_username, p_codice_volo, p_posto_assegnato, v_id_biglietto, p_num_bagagli;
END;
$$;
```

Figura 23: Procedure "Prenota volo"

4.3.11 Segnala smarrimento bagaglio

```
-- Segnala un bagaglio come smarrito, utente solo se possiede quel bagaglio,
--admin puo' segnalare tutti i bagagli
CREATE PROCEDURE public.segnala_smarrimento_bagaglio(IN p_id_bagaglio integer,
IN p_username character varying)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_id_biglietto INTEGER;
    v_codF_passeggero VARCHAR;
    v_is_admin BOOLEAN;
BEGIN
    SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
    INTO v_is_admin;

    IF NOT v_is_admin THEN
        SELECT p.id_biglietto, p."codF_passeggero"
        INTO v_id_biglietto, v_codF_passeggero
        FROM public."Prenotazione" p
        WHERE p.id_biglietto = (SELECT id_biglietto FROM public."Bagaglio" WHERE id_bagaglio = p_id_bagaglio)
          AND p.username = p_username;

        IF v_id_biglietto IS NULL THEN
            RAISE EXCEPTION 'L''utente % non è associato al bagaglio con ID %', p_username, p_id_bagaglio;
        END IF;
    ELSE
        SELECT p.id_biglietto, p."codF_passeggero"
        INTO v_id_biglietto, v_codF_passeggero
        FROM public."Prenotazione" p
        WHERE p.id_biglietto = (SELECT id_biglietto FROM public."Bagaglio" WHERE id_bagaglio = p_id_bagaglio);
    END IF;

    UPDATE public."Bagaglio"
    SET stato_bagaglio = 'smarrito'
    WHERE id_bagaglio = p_id_bagaglio;

    RAISE NOTICE 'Il bagaglio con ID % è stato segnato come smarrito.', p_id_bagaglio;
END;
$$;
```

Figura 24: Procedure "Segnala smarrimento bagaglio"

4.3.12 Visualizza prenotazioni per codF

```
-- Visualizza prenotazioni legate a un codice fiscale
CREATE PROCEDURE public.visualizza_prenotazioni_codf(IN p_codf_passeggero character varying)
    LANGUAGE plpgsql
    AS $$DECLARE
    r               RECORD;
    num_bagagli     INTEGER;
    v_out           TEXT := '';
    v_count_rows    INTEGER := 0;
BEGIN
    FOR r IN
        SELECT
            p.id_biglietto,
            p.posto_assegnato,
            p.stato,
            p.id_volo,
            p.username,
            p."codF_passeggero"
        FROM public."Prenotazione" p
        JOIN public."Passeggero" ps
          ON p."codF_passeggero" = ps."codF"
        WHERE ps."codF" = p_codf_passeggero
        ORDER BY p.id_biglietto
    LOOP
        SELECT COUNT(*)
          INTO num_bagagli
          FROM public."Bagaglio" b
         WHERE b.id_biglietto = r.id_biglietto;

        v_out := v_out ||
                 format(
                   'ID Biglietto: %s | Posto: %s | Stato: %s | ID Volo: %s | Username: %s | CodF: %s |
                   Bagagli: %s',
                   r.id_biglietto, r.posto_assegnato, r.stato, r.id_volo, r.username, r."codF_passeggero",
                   num_bagagli
                 ) || E'\n';

        v_count_rows := v_count_rows + 1;
    END LOOP;

    IF v_count_rows = 0 THEN
        RAISE NOTICE 'Nessuna prenotazione trovata per il Codice Fiscale: %', p_codf_passeggero;
    ELSE
        RAISE NOTICE '%', v_out;
    END IF;
END;
$$;
```

Figura 25: Procedure "Visualizza prenotazioni per codF"

4.3.13 Visualizza prenotazioni per volo

```
-- Mostra tutte le prenotazioni relative a un volo
CREATE PROCEDURE public.visualizza_prenotazioni_per_volo(IN p_id_volo integer,
IN p_username character varying)
    LANGUAGE plpgsql
    AS $$
DECLARE
    result RECORD;
    num_bagagli INTEGER;
    v_is_admin BOOLEAN;
BEGIN
    SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
    INTO v_is_admin;

    IF NOT v_is_admin THEN
        FOR result IN
            SELECT
                p.id_biglietto,
                p.posto_assegnato,
                p.stato,
                p.id_volo,
                p.username,
                p."codF_passeggero"
            FROM
                public."Prenotazione" p
            WHERE
                p.id_volo = p_id_volo AND p.username = p_username
        LOOP
            SELECT COUNT(*) INTO num_bagagli
            FROM public."Bagaglio" b
            WHERE b.id_biglietto = result.id_biglietto;
            RAISE NOTICE 'ID Biglietto: %, Posto Assegnato: %, Stato: %, ID Volo: %, Username: %,
            Codice Fiscale Passeggero: %, Numero Bagagli: %',
            result.id_biglietto, result.posto_assegnato, result.stato,
            result.id_volo, result.username, result."codF_passeggero", num_bagagli;
        END LOOP;
    ELSE
        FOR result IN
            SELECT
                p.id_biglietto,
                p.posto_assegnato,
                p.stato,
                p.id_volo,
                p.username,
                p."codF_passeggero"
            FROM
                public."Prenotazione" p
            WHERE
                p.id_volo = p_id_volo
        LOOP
            SELECT COUNT(*) INTO num_bagagli
            FROM public."Bagaglio" b
            WHERE b.id_biglietto = result.id_biglietto;
            RAISE NOTICE 'ID Biglietto: %, Posto Assegnato: %, Stato: %, ID Volo: %, Username: %,
            Codice Fiscale Passeggero: %, Numero Bagagli: %',
            result.id_biglietto, result.posto_assegnato, result.stato, result.id_volo,
            result.username, result."codF_passeggero", num_bagagli;
        END LOOP;
    END IF;
END;
$$;
```

Figura 26: Procedure "Visualizza prenotazioni per volo"

4.3.14 Visualizza smarrimenti

```
-- Elenco bagagli smarriti (solo admin)
CREATE PROCEDURE public.visualizza_smarrimenti(IN p_username character varying)
    LANGUAGE plpgsql
    AS $$
DECLARE
    v_is_admin BOOLEAN;
    result RECORD;
BEGIN
    SELECT EXISTS(SELECT 1 FROM public."Amministratore" WHERE username = p_username)
    INTO v_is_admin;

    IF NOT v_is_admin THEN
        RAISE EXCEPTION 'L''utente % non è un amministratore.', p_username;
    END IF;

    FOR result IN
        SELECT
            b.id_bagaglio,
            b.stato_bagaglio,
            b.id_biglietto,
            p.posto_assegnato,
            v.codice_volo,
            v.compagnia,
            v.orario_previsto,
            v.origine,
            v.destinazione
        FROM
            public."Bagaglio" b
        JOIN
            public."Prenotazione" p ON b.id_biglietto = p.id_biglietto
        JOIN
            public."Volo" v ON p.id_volo = v.codice_volo
        WHERE
            b.stato_bagaglio = 'smarrito'
    LOOP
        RAISE NOTICE 'ID Bagaglio: %, Stato: %, Posto Assegnato: %, Codice Volo: %, Compagnia: %,
        Orario: %, Origine: %, Destinazione: %',
            result.id_bagaglio, result.stato_bagaglio, result.posto_assegnato, result.codice_volo,
            result.compagnia, result.orario_previsto, result.origine, result.destinazione;
    END LOOP;
END;
$$;
```

Figura 27: 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 28: Procedure "Visualizza voli"