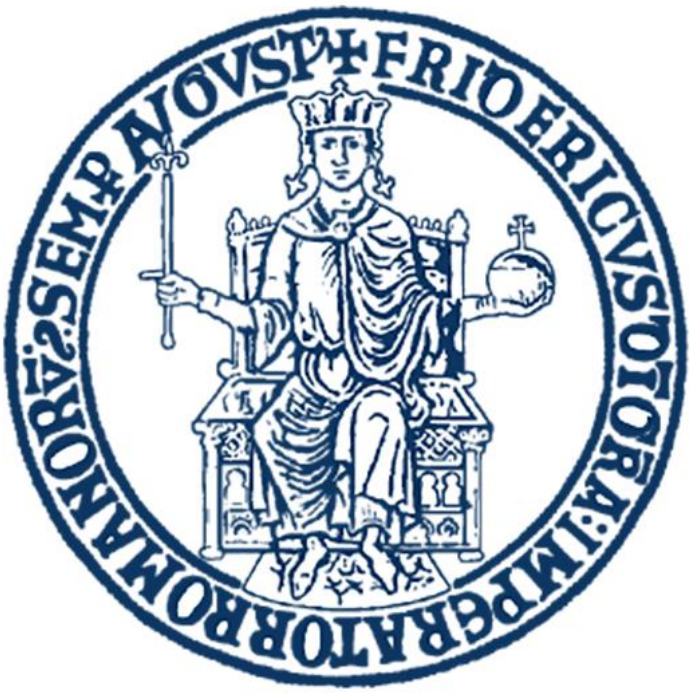


UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

DOCUMENTAZIONE PER PROGETTO

DI BASI DI DATI

CdL Triennale in Informatica
AGOSTINO SORRENTINO
N86005123
MARIATERESA PRINCIPATO
N86005284

ANNO ACCADEMICO: 2024/2025

Sommario

1 Introduzione

Il presente elaborato ha l'obiettivo di documentare l'analisi, la progettazione e lo sviluppo di una base di dati relazionale utilizzando il DBMS PostgreSQL, realizzata dagli studenti Mariateresa Principato e Agostino Sorrentino, iscritti al Corso di Laurea in Informatica presso l'Università degli Studi di Napoli "Federico II". Il progetto è stato svolto nell'ambito dell'insegnamento di Basi di Dati ed è finalizzato alla realizzazione di un sistema informativo per la gestione dell'aeroporto di Napoli, con lo scopo di organizzare e monitorare le operazioni aeroportuali in modo efficiente e strutturato.

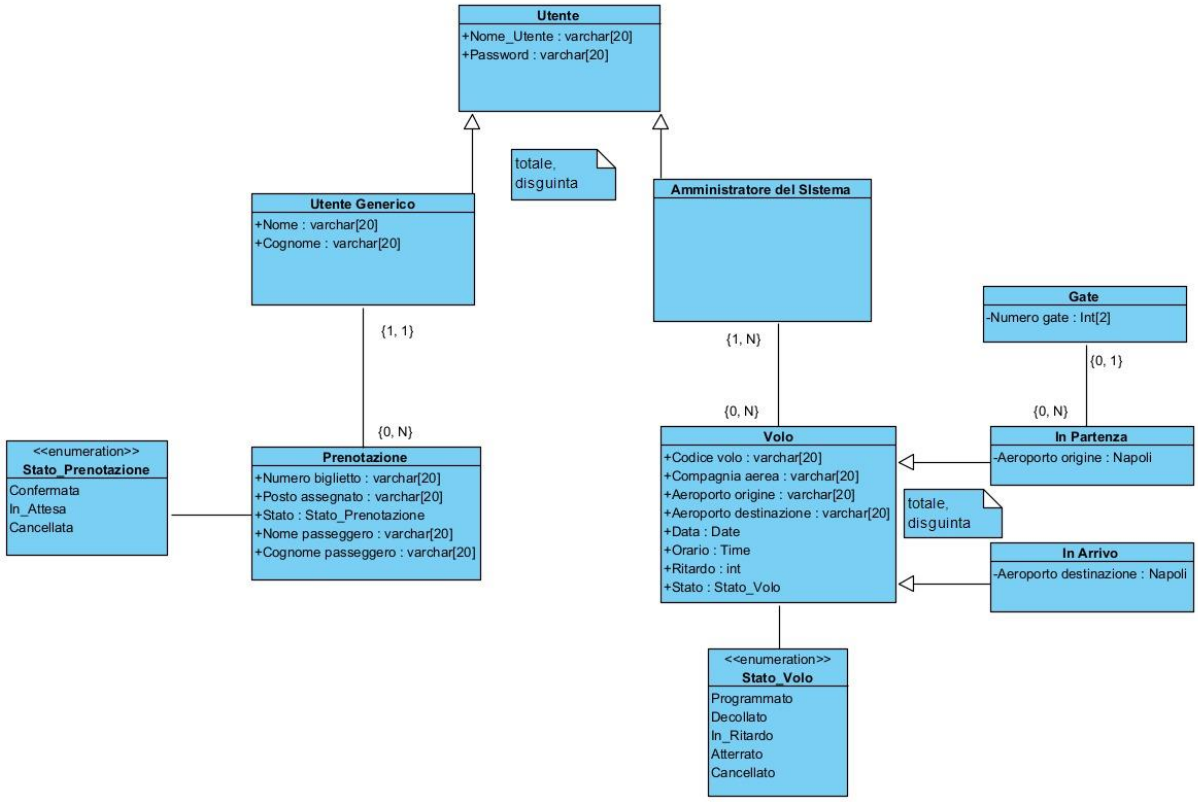
1.1 Descrizione del problema

Verranno presentati la progettazione e lo sviluppo di una base di dati relazionale che supporti un sistema informativo per la gestione delle attività aeroportuali presso l'aeroporto di Napoli. Il sistema ha l'obiettivo di organizzare, monitorare e aggiornare in modo strutturato ed efficiente le operazioni relative a voli, prenotazioni, gate e utenti. Il sistema gestisce sia i voli in arrivo che quelli in partenza, conservando per ciascuno informazioni come compagnia aerea, aeroporto di origine e destinazione, data e orario previsti, stato attuale (programmato, decollato, in ritardo, atterrato, cancellato) ed eventuali ritardi. È prevista inoltre la possibilità di associare i voli a gate specifici, con funzionalità per la modifica delle assegnazioni da parte degli amministratori. Gli utenti autenticati si distinguono in due categorie: utenti generici, che possono effettuare prenotazioni, e amministratori, che possono inserire e aggiornare i voli, gestire i gate e monitorare le operazioni. Le prenotazioni sono collegate a un determinato volo e comprendono informazioni sul passeggero, sul posto assegnato e sullo stato della prenotazione (in attesa, confermata o cancellata).

2 Progettazione Concettuale

In questo capitolo viene descritta la progettazione del database a un livello di astrazione elevato, svincolato da qualsiasi dettaglio implementativo. A partire dall'analisi dei requisiti funzionali e informativi relativi alla gestione dell'aeroporto di Napoli, si è giunti alla definizione di uno schema concettuale, rappresentato attraverso un diagramma UML delle classi. Questo schema evidenzia le principali entità coinvolte, come voli, utenti, prenotazioni e gate, insieme alle relazioni che le legano e ai vincoli logici che regolano il corretto funzionamento del sistema informativo.

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

2.2.1Analisi delle chiavi

Ai fini dell'efficienza nella rappresentazione delle varie entità, nello specifico Utente, Volo, Prenotazione e Gate, risulta conveniente l'introduzione di chiavi primarie (per esempio, id per utente. Questa soluzione, adottata nel modello ristrutturato, consente di identificare in maniera univoca e computazionalmente più efficiente ciascuna istanza, facilitando la gestione delle relazioni e delle operazioni. Al contrario, nel modello base, alcune entità utilizzano come chiavi attributi descrittivi, che potrebbero non essere univoci e sono maggiormente soggetti a errori o duplicazioni. Questo approccio può compromettere l'integrità dei dati e rendere più complessa la manutenzione.

2.2.2Analisi degli attributi derivati

In entrambi i modelli non sono presenti esplicitamente attributi derivati; tuttavia, il modello ristrutturato presenta una struttura più modulare e normalizzata che agevola il calcolo di informazioni derivabili tramite query. Nel modello base, l'assenza di un'organizzazione chiara delle relazioni rende meno immediato l'utilizzo di attributi derivati e riduce la flessibilità nell'interrogazione dei dati.

2.2.3Analisi delle ridondanze

La fase di analisi delle ridondanze ha permesso di identificare e risolvere una delle principali inefficienze presenti nel diagramma originale. Voli: le classi Volo in Partenza e Volo in Arrivo, presenti nel modello iniziale, risultavano ridondanti, in quanto caratterizzate da una struttura informativa pressoché identica. Entrambe possedevano attributi comuni come: compagnia aerea, aeroporto di origine, aeroporto di destinazione, data, orario, ritardo e stato del volo.

Questa duplicazione avrebbe comportato una gestione inefficiente e potenzialmente incoerente dei dati. Soluzione: nel diagramma ristrutturato, la ridondanza è stata eliminata attraverso l'unificazione delle due entità in una singola classe Volo. A questa è stato aggiunto l'attributo tipo, che distingue se il volo è in arrivo o in partenza. Questa ristrutturazione consente di:

- Eliminare la duplicazione informativa
- Semplificare la logica di gestione dei voli, unificandola in una struttura più efficiente e manutenibile

2.2.4Analisi degli attributi strutturati

Nessun attributo strutturato (ovvero attributi composti da più sotto-attributi come un indirizzo completo o un nome completo con titolo, nome, cognome) è presente né nel diagramma originale né in quello ristrutturato. Tutti gli attributi sono stati modellati come atomici.

2.2.5Analisi degli attributi a valore multiplo

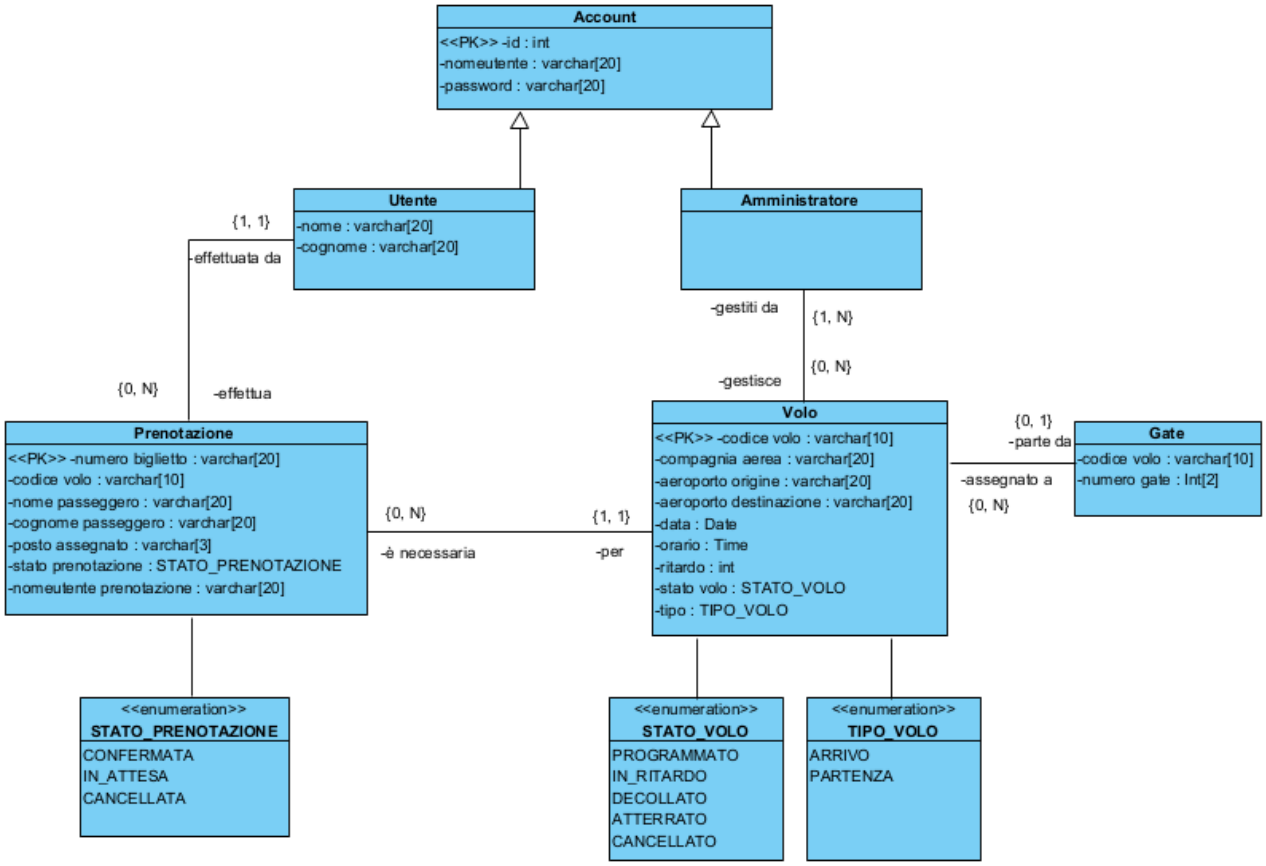
In entrambi i modelli – sia il diagramma base che quello ristrutturato – non si riscontrano attributi a valore multiplo, ovvero attributi che possono assumere più valori per una stessa istanza di entità. Tutti gli attributi presenti nei due modelli sono atomici, cioè assumono un singolo valore per ogni istanza. Esempi evidenti sono attributi come posto assegnato, stato prenotazione, compagnia aerea, nome, cognome, che assumono esattamente uno e un solo valore per ogni record di prenotazione, volo o utente. Questa scelta evita la necessità di strutture complesse e garantisce una maggiore semplicità nella gestione e nell'interrogazione dei dati da parte di un DBMS.

2.2.6Analisi delle gerarchie di specializzazione

La gerarchia tra Utente, Utente Generico e Amministrare del modello base e Account, Utente e Amministratore del modello ristrutturato non cambia molto. Oltre a un ovvio cambio di nomi di tabella per ridurre confusione, è stato introdotta una chiave primaria per semplificare operazioni e relazioni.

Per quanto riguarda i voli, la struttura di Volo, in Partenza e in Arrivo del modello base è stata accorpata in una sola classe Volo nel modello ristrutturato per semplificare e per ridurre ripetizioni.

2.3 Class Diagram Ristrutturato



2.4 Dizionario delle Classi

Classe	Descrizione	Attributi
Account	Rappresenta un account del sistema, base per Utente e Amministratore.	• **IdUtente** (int) : Identificativo univoco dell'Account (chiave primaria). • **nomeutente** (varchar[20]) : Nome utente per l'accesso al sistema. • **password** (varchar[20]): Password dell'account.
Utente	Rappresenta un utente comune del sistema, che può effettuare prenotazioni di voli.	• **nome** (varchar[20]): Nome dell'Utente. • **cognome** (varchar[20]): Cognome dell'Utente
Amministratore	Rappresenta un amministratore del sistema, con accesso a funzionalità gestionali.	
Volo	Rappresenta i voli registrati dal sistema.	• **codice volo** (varchar[10]) : Identificativo univoco del volo (chiave primaria). • **compagnia aerea** (varchar[20]) : Nome della compagnia aerea. • **aeroporto origine** (varchar[20]) : Aeroporto di partenza (Napoli). • **aeroporto destinazione** (varchar[20]) : Aeroporto di arrivo. • **data** (Date) : Data del volo. • **orario** (Time) : Orario di partenza previsto. • **ritardo** (Time) : Eventuale ritardo. • **tipo**(TIPO_VOLO) :Tipo di volo, se in partenza o in arrivo. • **stato volo** (STATO_VOLO) : Stato corrente del volo.

Prenotazione	Descrive una prenotazione effettuata da un utente per un determinato volo.	• **numero biglietto** (varchar[20]): Identificativo univoco della prenotazione (chiave primaria). • **codice volo** (varchar[10]) : Codice del volo prenotato. • **nome passeggero** (varchar[20]) : Nome del passeggero. • **cognome passeggero** (varchar[20]) : Cognome del passeggero. • **posto assegnato** (varchar[3]) : Posto assegnato sul volo. • **stato prenotazione** (STATO_PRENOTAZIONE) : Stato della prenotazione. • **nomeutente prenotazione** (varchar[20]) : Nome utente che ha effettuato la prenotazione.
GATE	Rappresenta un gate dell'aeroporto associato a un volo in partenza.	• **codice volo** (varchar[10]) : Identificativo univoco del gate (chiave primaria). • **numero gate** (int[2]) : Numero del gate.
STATO_PRENOTAZIONE	Enumerazione degli stati possibili di una prenotazione.	• CONFERMATA • IN_ATTESA • CANCELLATA
STATO_VOLO	Enumerazione degli stati possibili per un volo.	• PROGRAMMATO • IN_RITARDO • DECOLLATO • ATTERRATO • CANCELLATO
TIPO_VOLO	Enumerazione dei tipi possibili per un volo	• PARTENZA • ARRIVO

2.5 Dizionario delle Associazioni

Nome	Descrizione	Classi coinvolte
Effettua	Esprime l'azione di prenotazione effettuata da un Utente.	**Utente** [1,1] *effettuante*: indica un Utente che effettua una o più Prenotazioni. **Prenotazione** [0,N] *prenotazione effettuata da*: indica una Prenotazione effettuata da un Utente.
Gestisce	Esprime la relazione di gestione tra un Amministratore e uno o più Voli.	**Amministratore** [1,N] *gestore*: indica un Amministratore che gestisce uno o più Voli. **Volo** [0,N] *gestito*: indica un Volo gestito da un Amministratore.
E' necessaria per	Esprime che una Prenotazione è necessaria per un Volo in Partenza.	**Prenotazione** [0,N] *necessaria*: indica una Prenotazione necessaria per uno specifico Volo. **Volo** [1,1] *prenotazione associata*: indica il Volo per cui serve la Prenotazione.
Parte da	Esprime da quale Gate parte un Volo	**Gate** [0,1] *di origine*: indica un Gate da cui parte un Volo. **Volo** [0,N] *volo* : indica un Volo associato a un Gate.

2.6 Dizionario dei Vincoli

Nome	Descrizione
unique_passeggero_volo	Uno stesso passeggero non può essere prenotato più di una volta sullo stesso volo.
check_aeroporti_diversi	L'aeroporto di origine deve differire da quello di destinazione.
unique_posto_volo	Ogni posto su un volo specifico può essere assegnato una sola volta.
gate_volo_partenza_check	Un gate può essere assegnato solo a voli in partenza.

2.7 Dizionario di Funzioni, Procedure e View

Nome	Descrizione
VIEW voli_in_arrivo	Permette di visualizzare una tabella con tutti i voli in arrivo.
VIEW voli_in_partenza	Permette di visualizzare una tabella con tutti i voli in partenza.
VIEW tutti_voli	Permette di visualizzare una tabella con tutti i voli.
VIEW prenotazioni	Permette di visualizzare una tabella con tutte le prenotazioni effettuate da utenti
VIEW tutti_utenti	Permette di visualizzare una tabella con tutti gli account utenti e i suoi attributi
VIEW tutti_admin	Permette di visualizzare una tabella con tutti gli account amministratori e i suoi attributi
FUNCTION genera_numero_biglietto	Utilizzata nella procedure crea_prenotazione, genera un numero unico per il biglietto.
FUNCTION genera_posto_casuale	Utilizzata nella procedure crea_prenotazione, genera una sequenza di due numeri e una lettera per simulare l'assegnazione dei posti. Prima di restituire il valore, si assicura che quel posto non sia già assegnato nel volo
FUNCTION verifica_utente	Utilizzato per ogni function o procedure dove è necessario simulare un login alla piattaforma, controlla le credenziali dell'utente
FUNCTION verifica_admin	Utilizzato per ogni function o procedure dove è necessario simulare un login alla piattaforma, controlla le credenziali dell'amministratore
FUNCTION i_miei_voli	Restituisce una tabella con tutte le prenotazioni effettuate dal proprio account
FUNCTION cerca_prenotazioni_passeggero	Restituisce una tabella con tutte le prenotazioni effettuate a nome e cognome di una persona

FUNCTION cerca_prenotazioni_volo	Restituisce una tabella con tutte le prenotazioni effettuate per quel volo
PROCEDURE crea_account_utente	Crea un account utente inserendo i dati all'interno delle tabelle account e utente
PROCEDURE elimina_account_utente	Elimina un account utente rimuovendone i dati e le prenotazioni associate.
PROCEDURE crea_prenotazione	Crea una prenotazione aggiungendo i dati inseriti dall'utente nelle tabelle, generando codice_prenotazione e posto_assegnato
PROCEDURE modifica_prenotazione	Consente all'utente di modificare la propria prenotazione, cambiando nome e cognome del passeggero
PROCEDURE elimina_prenotazione	Elimina la prenotazione che corrisponde col codice_prenotazione fornito dall'utente.
PROCEDURE crea_account_admin	Crea un account utente inserendo i dati all'interno delle tabelle account e amministratore.
PROCEDURE elimina_account_admin	Elimina un account amministratore
PROCEDURE aggiungi_volo	Crea un volo a seconda dei dati inseriti dall'amministratore
PROCEDURE elimina_volo	Elimina un volo e tutte le prenotazioni per esso
PROCEDURE aggiorna_stato_volo	Permette di modificare lo stato e il ritardo del volo. Nel caso in cui il ritardo sia >0, lo stato sarà automaticamente imposto a 'IN_RITARDO'
PROCEDURE aggiorna_stato_prenotazione	Permette di modificare lo stato di prenotazioni di utenti ai voli
PROCEDURE assegna_gate	Assegna un gate a un volo. Non è possibile assegnare gate ai voli in arrivo, o a voli il cui stato è 'DECOLLATO', 'ATTERRATO' o 'CANCELLATO'

3 Progettazione Logica

In questo capitolo tratteremo la seconda fase della progettazione, scendendo ad un livello di astrazione più basso rispetto al precedente.

3.1 Schema Logico

Al suo interno, le chiavi primarie sono indicate con una sottolineatura mentre le chiavi esterne in **grassetto**.

- Account (id, nomeutente, password)
- Utente (**id**, nome, cognome)

  id -> Account.id
- Amministratore (**id**)

  id -> Account.id
- Volo (codice_volo, compagnia_aerea, aeroporto_origine, aeroporto_destinazione, data, orario, ritardo, stato, tipo)

  stato -> STATO_VOLO

  tipo -> TIPO_VOLO
- Prenotazione (numero_biglietto, **codice_volo**, nome_passeggero, cognome_passeggero, posto_assegnato, stato_prenotazione, nomeutente_prenotazione)

  stato_prenotazione -> STATO_PRENOTAZIONE

  codice_volo -> Volo.codice_volo

  nomeutente_prenotazione -> Account.nomeutente
- Gate (numero_gate, **codice_volo**)

  Codice_volo -> Volo.codice_volo
- STATO_PRENOTAZIONE = {CONFERMATA, IN_ATTESA, CANCELLATA}
- STATO_VOLO = {PROGRAMMATO, IN_RITARDO, DECOLLATO, ATTERRATO, CANCELLATO}
- TIPO_VOLO = {PARTENZA, ARRIVO}

4 Progettazione Fisica

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

4.1 Definizione Tabelle ed Enumerazioni

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

4.1.1 Enumerazione stato_prenotazione

```
CREATE TYPE stato_prenotazione AS ENUM (
    'CONFERMATA',
    'IN_ATTESA',
    'CANCELLATA'
);
```

4.1.2 Enumerazione stato_volo

```
CREATE TYPE stato_volo AS ENUM (
    'PROGRAMMATO',
    'IN_RITARDO',
    'DECOLLATO',
    'ATTERRATO',
    'CANCELLATO'
);
```

4.1.3 Enumerazione tipo_volo

```
CREATE TYPE tipo_volo AS ENUM (
    'ARRIVO',
    'PARTENZA'
);
```

4.1.4 Tabella account

```
-- Tabella Account

CREATE TABLE account
(
    id SERIAL PRIMARY KEY,
    nomeutente VARCHAR(20) UNIQUE NOT NULL,
    password VARCHAR(20) NOT NULL
```

4.1.5 Tabella utente

```
-- Tabella Utente
CREATE TABLE utente
(
    id INTEGER PRIMARY KEY REFERENCES account(id),
    nome VARCHAR(20) NOT NULL,
    cognome VARCHAR(20) NOT NULL
);
```

4.1.6 Tabella amministrat

```
-- Tabella Amministratore
CREATE TABLE amministratore (

    id INTEGER PRIMARY KEY REFERENCES account(id)
);
```

4.1.7 Tabella volo

```
-- Tabella Volo
CREATE TABLE volo
(
    codice VARCHAR(10) PRIMARY KEY,
    compagnia_aerea VARCHAR(20) NOT NULL,
    aeroporto_origine VARCHAR(20) NOT NULL,
    aeroporto_destinazione VARCHAR(20) NOT NULL,
    data_partenza DATE NOT NULL,
    orario TIME NOT NULL,
    ritardo INTEGER DEFAULT 0,
    stato stato_volo DEFAULT 'PROGRAMMATO',
    tipo tipo_volo
);
```

4.1.8 Tabella prenotazione

```
-- Tabella Prenotazione
CREATE TABLE prenotazione
(
    numero_biglietto VARCHAR(20) PRIMARY KEY,
    posto_assegnato VARCHAR(4) NOT NULL,
    stato stato_prenotazione DEFAULT 'CONFERMATA',
    nome_passeggero VARCHAR(20) NOT NULL,
    cognome_passeggero VARCHAR(20) NOT NULL,
    codice_volo VARCHAR(10) REFERENCES volo(codice),
    username_prenotazione VARCHAR(20) REFERENCES account(nomeutente)
);
```

4.1.9 Tabella gate

```
-- Tabella Gate
CREATE TABLE gate
(
    numero_gate INTEGER NOT NULL,
    codice_volo VARCHAR(10) UNIQUE,
    FOREIGN KEY (codice_volo) REFERENCES volo(codice),
    CONSTRAINT check_positive_gate CHECK (numero_gate > 0),
    PRIMARY KEY (numero_gate, codice_volo)
);
```

4.2 Definizione Vincoli

4.2.1 Vincolo gate_volo_partenza_check

```
-- Trigger per assicurarsi che unn gate non possa essere assegnato a un volo in
arrivo, ma solo a quelli in partenza

CREATE OR REPLACE FUNCTION check_volo_partenza_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.codice_volo IS NOT NULL THEN
        IF NOT EXISTS (
            SELECT 1
            FROM volo v
            WHERE v.codice = NEW.codice_volo
            AND v.tipo = 'PARTENZA'
        ) THEN
            RAISE EXCEPTION 'Il gate può essere assegnato solo a voli in
partenza';
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER gate_volo_partenza_check
BEFORE INSERT OR UPDATE ON gate
FOR EACH ROW
EXECUTE FUNCTION check_volo_partenza_trigger();
```

4.2.2 Vincolo unique_passeggero_volo

```
-- Vincoli di unicità per prenotazioni
ALTER TABLE prenotazione
ADD CONSTRAINT unique_passeggero_volo
UNIQUE (nome_passeggero, cognome_passeggero, codice_volo);
```

4.2.3 Vincolo unique_posto_volo

```
ALTER TABLE prenotazione
ADD CONSTRAINT unique_posto_volo
UNIQUE (posto_assegnato, codice_volo);
```

4.2.4 Vincolo check_aeroporti_diversi

```
-- Vincolo per garantire che aeroporto_origine e aeroporto_destinazione siano
diversi
ALTER TABLE volo
ADD CONSTRAINT check_aeroporti_diversi
CHECK (aeroporto_origine <> aeroporto_destinazione);
```

4.3 Definizione Views

4.3.1 View voli_in_partenza

```
-- Vista per voli in partenza
CREATE OR REPLACE VIEW voli_in_partenza AS
SELECT
    v.*,
    g.numero_gate
FROM volo v
LEFT JOIN gate g ON v.codice = g.codice_volo
WHERE v.tipo = 'PARTENZA'
AND v.aeroporto_origine = 'Napoli';
```

4.3.2 View voli_in_arrivo

```
-- Vista per voli in arrivo
CREATE OR REPLACE VIEW voli_in_arrivo AS
SELECT *
FROM volo
WHERE tipo = 'ARRIVO'
AND aeroporto_destinazione = 'Napoli';
```

4.3.3 View prenotazioni

```
-- Vista prenotazioni complete
CREATE OR REPLACE VIEW prenotazioni AS
SELECT
    p.numero_biglietto,
    p.posto_assegnato,
    p.stato AS stato_prenotazione,
    p.nome_passeggero,
    p.cognome_passeggero,
    p.codice_volo,
    p.username_prenotazione,
    v.compagnia_aerea,
    v.aeroporto_origine,
    v.aeroporto_destinazione,
    v.data_partenza,
    v.orario,
    v.ritardo,
    v.stato AS stato_volo,
    v.tipo
FROM prenotazione p
JOIN volo v ON p.codice_volo = v.codice
JOIN account a ON p.username_prenotazione = a.nomeutente;
```

4.3.4 View tutti_utenti

```
-- Vista per tutti gli utenti
CREATE OR REPLACE VIEW tutti_utenti AS
SELECT
    a.nomeutente,
    a.password,
    u.nome,
    u.cognome
FROM account a
JOIN utente u ON a.id = u.id
ORDER BY u.cognome, u.nome;
```

4.3.5 View tutti_admin

```
-- Vista per amministratori
CREATE OR REPLACE VIEW tutti_admin AS
SELECT
    a.nomeutente,
    a.password
FROM account a
JOIN amministratore am ON a.id = am.id
ORDER BY a.nomeutente;
```

4.3.6 View tutti_voli

```
-- Vista per tutti i voli

CREATE OR REPLACE VIEW tutti_voli AS
SELECT
    v.codice,
    v.compagnia_aerea,
    v.aeroporto_origine,
    v.aeroporto_destinazione,
    v.data_partenza,
    v.orario,
    v.ritardo,
    v.stato,
    v.tipo,
    g.numero_gate
FROM volo v
LEFT JOIN gate g ON v.codice = g.codice_volo
ORDER BY v.data_partenza, v.orario;
```

4.4 Definizione Functions

4.4.1 Function genera_numero_biglietto

```
-- Funzione per generare numero del biglietto
CREATE OR REPLACE FUNCTION genera_numero_biglietto()
RETURNS VARCHAR AS $$
DECLARE
    ultimo_numero INTEGER;
BEGIN

    SELECT MAX(CAST(SUBSTRING(numero_biglietto FROM 4) AS INTEGER))
    INTO ultimo_numero
    FROM prenotazione
    WHERE numero_biglietto LIKE 'PRE%';

    IF ultimo_numero IS NULL THEN
        ultimo_numero := 0;
    END IF;

    ultimo_numero := ultimo_numero + 1;

    RETURN 'PRE' || LPAD(ultimo_numero::TEXT, 6, '0');
END;
$$ LANGUAGE plpgsql;

-- Funzione per generare posto casuale
CREATE OR REPLACE FUNCTION genera_posto_casuale(p_codice_volo VARCHAR)
RETURNS VARCHAR AS $$
DECLARE
    posto VARCHAR(4);
    occupato BOOLEAN;
BEGIN
    LOOP
        -- Genera un posto casuale (fila 1-30, lettera A-F)
        posto := LPAD(FLOOR(RANDOM() * 30 + 1)::TEXT, 2, '0') ||
                 CHR(FLOOR(RANDOM() * 6 + 65)::INTEGER);

        -- Verifica se il posto è già occupato
        SELECT EXISTS(
            SELECT 1
            FROM prenotazione
            WHERE codice_volo = p_codice_volo
            AND posto_assegnato = posto
        ) INTO occupato;

        EXIT WHEN NOT occupato;
    END LOOP;

    RETURN posto;
END;
$$ LANGUAGE plpgsql;
```

4.4.2 Function verifica_utente

```
CREATE OR REPLACE FUNCTION verifica_utente(
    p_nomeutente VARCHAR(20),
    p_password VARCHAR(20)
) RETURNS BOOLEAN AS $$
BEGIN
    RETURN EXISTS (
        SELECT 1
        FROM account a
        JOIN utente u ON a.id = u.id
        WHERE a.nomeutente = p_nomeutente
        AND a.password = p_password
    );
END;
$$ LANGUAGE plpgsql;
```

4.4.3 Function verifica_utente

```
CREATE OR REPLACE FUNCTION verifica_admin(
    p_nomeutente VARCHAR(20),
    p_password VARCHAR(20)
) RETURNS BOOLEAN AS $$
BEGIN
    RETURN EXISTS (
        SELECT 1
        FROM account a
        JOIN amministratore am ON a.id = am.id
        WHERE a.nomeutente = p_nomeutente
        AND a.password = p_password
    );
END;
$$ LANGUAGE plpgsql;
```

4.4.4 Function i_miei_voli

```
CREATE OR REPLACE FUNCTION i_miei_voli(
    p_nomeutente VARCHAR(20),
    p_password VARCHAR(20)
) RETURNS TABLE (
    numero_biglietto VARCHAR(20),
    posto_assegnato VARCHAR(4),
    stato_prenotazione stato_prenotazione,
    nome_passeggero VARCHAR(20),
    cognome_passeggero VARCHAR(20),
    codice_volo VARCHAR(10),
    compagnia_aerea VARCHAR(20),
    aeroporto_origine VARCHAR(20),
    aeroporto_destinazione VARCHAR(20),
    data_partenza DATE,
    orario TIME,
    ritardo INTEGER,
    stato_volo stato_volo,
    numero_gate INTEGER
) AS $$
BEGIN
    -- Verifica le credenziali dell'utente
    IF NOT verifica_utente(p_nomeutente, p_password) THEN
        RAISE EXCEPTION 'Credenziali non valide';
    END IF;

    RETURN QUERY
    SELECT
        p.numero_biglietto,
        p.posto_assegnato,
        p.stato,
        p.nome_passeggero,
        p.cognome_passeggero,
        p.codice_volo,
        v.compagnia_aerea,
        v.aeroporto_origine,
        v.aeroporto_destinazione,
        v.data_partenza,
        v.orario,
        v.ritardo,
        v.stato,
        g.numero_gate
    FROM prenotazione p
    JOIN volo v ON p.codice_volo = v.codice
    LEFT JOIN gate g ON v.codice = g.codice_volo
    WHERE p.username_prenotazione = p_nomeutente
    ORDER BY v.data_partenza, v.orario;
END;
$$ LANGUAGE plpgsql;
```

4.4.5 Function cerca_prenotazioni_passeggero

```
CREATE OR REPLACE FUNCTION cerca_prenotazioni_passeggero(
    p_nomeutente VARCHAR(20),
    p_password VARCHAR(20),
    p_nome_passeggero VARCHAR(20),
    p_cognome_passeggero VARCHAR(20)
) RETURNS TABLE (
    numero_biglietto VARCHAR(20),
    posto_assegnato VARCHAR(4),
    stato_prenotazione stato_prenotazione,
    codice_volo VARCHAR(10),
    compagnia_aerea VARCHAR(20),
    aeroporto_origine VARCHAR(20),
    aeroporto_destinazione VARCHAR(20),
    data_partenza DATE,
    orario TIME,
    ritardo INTEGER,
    stato_volo stato_volo,
    numero_gate INTEGER
) AS $$
BEGIN
    -- Verifica le credenziali dell'utente usando la funzione esistente
    IF NOT verifica_utente(p_nomeutente, p_password) THEN
        RAISE EXCEPTION 'Credenziali non valide';
    END IF;

    RETURN QUERY
    SELECT
        p.numero_biglietto,
        p.posto_assegnato,
        p.stato,
        p.codice_volo,
        v.compagnia_aerea,
        v.aeroporto_origine,
        v.aeroporto_destinazione,
        v.data_partenza,
        v.orario,
        v.ritardo,
        v.stato,
        g.numero_gate
    FROM prenotazione p
    JOIN volo v ON p.codice_volo = v.codice
    LEFT JOIN gate g ON v.codice = g.codice_volo
    WHERE p.nome_passeggero = p_nome_passeggero
    AND p.cognome_passeggero = p_cognome_passeggero
    ORDER BY v.data_partenza, v.orario;
END;
$$ LANGUAGE plpgsql;
```

4.4.6 Function cerca_prenotazioni_volo

```
CREATE OR REPLACE FUNCTION cerca_prenotazioni_volo(
    p_nomeutente VARCHAR(20),
    p_password VARCHAR(20),
    p_codice_volo VARCHAR(10)
) RETURNS TABLE (
    numero_biglietto VARCHAR(20),
    posto_assegnato VARCHAR(4),
    stato_prenotazione stato_prenotazione,
    nome_passeggero VARCHAR(20),
    cognome_passeggero VARCHAR(20),
    compagnia_aerea VARCHAR(20),
    aeroporto_origine VARCHAR(20),
    aeroporto_destinazione VARCHAR(20),
    data_partenza DATE,
    orario TIME,
    ritardo INTEGER,
    stato_volo stato_volo,
    numero_gate INTEGER
) AS $$
BEGIN
    -- Verifica le credenziali dell'utente usando la funzione esistente
    IF NOT verifica_utente(p_nomeutente, p_password) THEN
        RAISE EXCEPTION 'Credenziali non valide';
    END IF;

    -- Verifica che il volo esista
    IF NOT EXISTS (SELECT 1 FROM volo WHERE codice = p_codice_volo) THEN
        RAISE EXCEPTION 'Volo con codice % non trovato', p_codice_volo;
    END IF;

    RETURN QUERY
    SELECT
        p.numero_biglietto,
        p.posto_assegnato,
        p.stato,
        p.nome_passeggero,
        p.cognome_passeggero,
        v.compagnia_aerea,
        v.aeroporto_origine,
        v.aeroporto_destinazione,
        v.data_partenza,
        v.orario,
        v.ritardo,
        v.stato,
        g.numero_gate
    FROM prenotazione p
    JOIN volo v ON p.codice_volo = v.codice
    LEFT JOIN gate g ON v.codice = g.codice_volo
    WHERE p.codice_volo = p_codice_volo
    ORDER BY p.posto_assegnato;
END;
$$ LANGUAGE plpgsql;
```

4.5 Definizione Procedures

4.5.1 Procedure crea_account_utente

```
CREATE OR REPLACE PROCEDURE crea_account_utente(
    p_nomeutente character varying,
    p_password character varying,
    p_nome character varying,
    p_cognome character varying
    )
AS $$
DECLARE
    v_account_id INTEGER;
BEGIN
    -- Verifica che il nome utente non esista già
    IF EXISTS (
        SELECT 1
        FROM account
        WHERE nomeutente = p_nomeutente
    ) THEN
        RAISE EXCEPTION 'Il nome utente % è già in uso', p_nomeutente;
    END IF;

    -- Inserisci il nuovo account e ottieni l'ID generato
    INSERT INTO account (nomeutente, password)
    VALUES (p_nomeutente, p_password)
    RETURNING id INTO v_account_id;

    -- Inserisci i dati dell'utente
    INSERT INTO utente (id, nome, cognome)
    VALUES (v_account_id, p_nome, p_cognome);

EXCEPTION
    WHEN others THEN
        -- In caso di errore, annulla entrambe le operazioni
        RAISE EXCEPTION 'Errore durante la creazione dell''account';
END;
$$ LANGUAGE plpgsql;
```

4.5.2 Procedure elimina_utente

```
CREATE OR REPLACE PROCEDURE elimina_utente(
    p_username_richiedente VARCHAR(20),  -- nomeutente dell'account in uso
    p_password_richiedente VARCHAR(20),   -- password dell'account in uso
        p_nomeutente VARCHAR(20)         -- nomeutente dell'account che si
vuole eliminare
) AS $$
DECLARE
    user_id INTEGER;
    is_self BOOLEAN;
    is_admin BOOLEAN;
BEGIN
    -- Verifica se è una richiesta da amministratore o dall'utente stesso
    is_admin := verifica_admin(p_username_richiedente,
p_password_richiedente);
    is_self := verifica_utente(p_username_richiedente,
p_password_richiedente);

    -- Se né l'admin né l'utente sono autenticati, errore
    IF NOT (is_admin OR is_self) THEN
        RAISE EXCEPTION 'Credenziali non valide';
    END IF;

    -- Se non è admin, verifica che stia eliminando se stesso
    IF NOT is_admin AND p_nomeutente != p_username_richiedente THEN
        RAISE EXCEPTION 'Un utente può eliminare solo il proprio account';
    END IF;

    -- Verifica che l'account da eliminare sia effettivamente un utente
    SELECT a.id INTO user_id
    FROM account a
    JOIN utente u ON a.id = u.id
    WHERE a.nomeutente = p_nomeutente;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Utente % non trovato o non è un utente normale',
p_nomeutente;
    END IF;

    -- Prima elimina tutte le prenotazioni dell'utente
    DELETE FROM prenotazione
    WHERE username_prenotazione = p_nomeutente;

    -- Poi elimina il record utente
    DELETE FROM utente
    WHERE id = user_id;

    -- Infine elimina l'account
    DELETE FROM account
    WHERE id = user_id;

EXCEPTION
    WHEN others THEN
        RAISE EXCEPTION 'Errore durante l''eliminazione dell''utente: %',
SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

4.5.3 Procedure crea_prenotazione

```
-- Procedura per creare una nuova prenotazione
CREATE OR REPLACE PROCEDURE crea_prenotazione(
    p_nomeutente VARCHAR,
    p_password VARCHAR,
    p_nome_passeggero VARCHAR,
    p_cognome_passeggero VARCHAR,
    p_codice_volo VARCHAR
) AS $$
DECLARE
    v_tipo_volo VARCHAR(10);
    v_stato_volo stato_volo;
BEGIN
    -- Verifica le credenziali dell'utente
    IF NOT verifica_utente(p_nomeutente, p_password) THEN
        RAISE EXCEPTION 'Credenziali non valide';
    END IF;

    -- Verifica che il volo esista e ottieni il tipo e lo stato
    SELECT tipo_volo, stato
    INTO v_tipo_volo, v_stato_volo
    FROM volo
    WHERE codice = p_codice_volo;

    -- Se il volo non esiste
    IF NOT FOUND THEN
        RAISE EXCEPTION 'Il volo specificato non esiste';
    END IF;

    -- Verifica che sia un volo in partenza
    IF v_tipo_volo != 'PARTENZA' THEN
        RAISE EXCEPTION 'Possono essere prenotati solo voli in partenza';
    END IF;

    -- Verifica che il volo non sia già partito o cancellato
    IF v_stato_volo IN ('DECOLLATO', 'ATTERRATO', 'CANCELLATO') THEN
        RAISE EXCEPTION 'Non è possibile prenotare un volo %', v_stato_volo;
    END IF;

    -- Procedi con l'inserimento della prenotazione
    INSERT INTO prenotazione (
        numero_biglietto,
        posto_assegnato,
        stato,
        nome_passeggero,
        cognome_passeggero,
        codice_volo,
        username_prenotazione
    ) VALUES (
        genera_numero_biglietto(),
        genera_posto_casuale(p_codice_volo),
        'CONFERMATA',
        p_nome_passeggero,
        p_cognome_passeggero,
        p_codice_volo,
        p_nomeutente
    );
END;
$$ LANGUAGE plpgsql;
```

4.5.4 Procedure modifica_prenotazione

```
CREATE OR REPLACE PROCEDURE modifica_prenotazione(
    p_nomeutente VARCHAR(20),
    p_password VARCHAR(20),
    p_numero_biglietto VARCHAR(20),
    p_nuovo_nome VARCHAR(20),
    p_nuovo_cognome VARCHAR(20)
) AS $$
DECLARE
    v_username_prenotazione VARCHAR(20);
    v_codice_volo VARCHAR(10);
    v_stato_volo stato_volo;
BEGIN
    -- Verifica le credenziali dell'utente
    IF NOT verifica_utente(p_nomeutente, p_password) THEN
        RAISE EXCEPTION 'Credenziali non valide';
    END IF;

    -- Verifica se la prenotazione esiste e ottiene le informazioni necessarie
    SELECT
        p.username_prenotazione,
        p.codice_volo,
        v.stato
    INTO
        v_username_prenotazione,
        v_codice_volo,
        v_stato_volo
    FROM prenotazione p
    JOIN volo v ON p.codice_volo = v.codice
    WHERE p.numero_biglietto = p_numero_biglietto;

    -- Se la prenotazione non esiste
    IF v_username_prenotazione IS NULL THEN
        RAISE EXCEPTION 'Prenotazione con numero biglietto % non trovata',
p_numero_biglietto;
    END IF;

    -- Verifica che l'utente sia il proprietario della prenotazione
    IF v_username_prenotazione != p_nomeutente THEN
        RAISE EXCEPTION 'Non hai i permessi per modificare questa prenotazione';
    END IF;

    -- Verifica che il volo non sia già partito o cancellato
    IF v_stato_volo IN ('DECOLLATO', 'ATTERRATO', 'CANCELLATO') THEN
        RAISE EXCEPTION 'Non è possibile modificare una prenotazione per un volo %',
v_stato_volo;
    END IF;

    -- Verifica che non esista già una prenotazione per lo stesso passeggero sul volo
    IF EXISTS (
        SELECT 1
        FROM prenotazione
        WHERE codice_volo = v_codice_volo
        AND nome_passeggero = p_nuovo_nome
        AND cognome_passeggero = p_nuovo_cognome
        AND numero_biglietto != p_numero_biglietto
    ) THEN
        RAISE EXCEPTION 'Esiste già una prenotazione per % % su questo volo', p_nuovo_nome,
p_nuovo_cognome;
    END IF;

    -- Aggiorna i dati del passeggero
    UPDATE prenotazione
    SET nome_passeggero = p_nuovo_nome,
        cognome_passeggero = p_nuovo_cognome
    WHERE numero_biglietto = p_numero_biglietto;

END;
$$ LANGUAGE plpgsql;
```

4.5.5 Procedure elimina_prenotazione

```
-- Procedura per eliminare una prenotazione
CREATE OR REPLACE PROCEDURE elimina_prenotazione(
    p_nomeutente VARCHAR(20),
    p_password VARCHAR(20),
    p_numero_biglietto VARCHAR(20)

) AS $$
DECLARE
    v_username_prenotazione VARCHAR(20);
    v_stato_volo stato_volo;
    v_codice_volo VARCHAR(10);
BEGIN
    -- Verifica le credenziali dell'utente
    IF NOT verifica_utente(p_nomeutente, p_password) THEN
        RAISE EXCEPTION 'Credenziali non valide';
    END IF;

    -- Verifica se la prenotazione esiste e ottiene le informazioni
necessarie
    SELECT
        p.username_prenotazione,
        v.stato,
        p.codice_volo
    INTO
        v_username_prenotazione,
        v_stato_volo,
        v_codice_volo
    FROM prenotazione p
    JOIN volo v ON p.codice_volo = v.codice
    WHERE p.numero_biglietto = p_numero_biglietto;

    -- Se la prenotazione non esiste, solleva un'eccezione
    IF v_username_prenotazione IS NULL THEN
        RAISE EXCEPTION 'Prenotazione con numero biglietto % non trovata',
p_numero_biglietto;
    END IF;

    -- Verifica che l'utente che sta cercando di eliminare sia il
proprietario della prenotazione
    IF v_username_prenotazione != p_nomeutente THEN
        RAISE EXCEPTION 'Non hai i permessi per eliminare questa
prenotazione';
    END IF;

    -- Elimina la prenotazione
    DELETE FROM prenotazione
    WHERE numero_biglietto = p_numero_biglietto;
END;
$$ LANGUAGE plpgsql;
```

4.5.6 Procedure crea_account_admin

```
CREATE OR REPLACE PROCEDURE crea_account_admin(
    p_nomeutente VARCHAR(20),
    p_password VARCHAR(20)
) AS $$
DECLARE
    v_account_id INTEGER;
BEGIN
    -- Verifica che il nome utente non esista già
    IF EXISTS (
        SELECT 1
        FROM account
        WHERE nomeutente = p_nomeutente
    ) THEN
        RAISE EXCEPTION 'Il nome utente % è già in uso', p_nomeutente;
    END IF;

    -- Inserisci il nuovo account e ottieni l'ID generato
    INSERT INTO account (nomeutente, password)
    VALUES (p_nomeutente, p_password)
    RETURNING id INTO v_account_id;

    -- Inserisci il record amministratore
    INSERT INTO amministratore (id)
    VALUES (v_account_id);

EXCEPTION
    WHEN others THEN
        RAISE EXCEPTION 'Errore durante la creazione dell''account
amministratore: %', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

4.5.7 Procedure elimina_admin

```
CREATE OR REPLACE PROCEDURE elimina_admin(
    p_admin_username VARCHAR(20),
    p_admin_password VARCHAR(20),
    p_nomeutente VARCHAR(20)
) AS $$
DECLARE
    admin_id INTEGER;
    v_count INTEGER;
BEGIN
    -- Verifica che chi sta eliminando sia un amministratore
    IF NOT verifica_admin(p_admin_username, p_admin_password) THEN
        RAISE EXCEPTION 'Non hai i permessi per eliminare un amministratore';
    END IF;

    -- Verifica che l'amministratore da eliminare esista
    SELECT a.id INTO admin_id
    FROM account a
    JOIN amministratore am ON a.id = am.id
    WHERE a.nomeutente = p_nomeutente;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Amministratore % non trovato', p_nomeutente;
    END IF;

    -- Elimina il record amministratore
    DELETE FROM amministratore
    WHERE id = admin_id;

    -- Elimina l'account
    DELETE FROM account
    WHERE id = admin_id;

EXCEPTION
    WHEN others THEN
        RAISE EXCEPTION 'Errore durante l''eliminazione dell''amministratore:
%', SQLERRM;
END;
$$ LANGUAGE plpgsql;
```

4.5.8 Procedure aggiungi_volo

```
CREATE OR REPLACE PROCEDURE aggiungi_volo(
    p_admin_username VARCHAR(20),
    p_admin_password VARCHAR(20),
    p_codice VARCHAR(10),
    p_compagnia_aerea VARCHAR(20),
    p_aeroporto_origine VARCHAR(20),
    p_aeroporto_destinazione VARCHAR(20),
    p_data_partenza DATE,
    p_orario TIME,
    p_tipo_volo VARCHAR(10),
    p_ritardo INTEGER
) AS $$
BEGIN
    -- Verifica le credenziali dell'amministratore
    IF NOT verifica_admin(p_admin_username, p_admin_password) THEN
        RAISE EXCEPTION 'Credenziali amministratore non valide';
    END IF;

    -- Verifica che il tipo_volo sia valido
    IF p_tipo_volo NOT IN ('ARRIVO', 'PARTENZA') THEN
        RAISE EXCEPTION 'Il tipo volo deve essere ARRIVO o PARTENZA';
    END IF;

    -- Verifica che il ritardo non sia negativo
    IF p_ritardo < 0 THEN
        RAISE EXCEPTION 'Il ritardo non può essere negativo';
    END IF;

    -- Inserimento del nuovo volo
    INSERT INTO volo (
        codice,
        compagnia_aerea,
        aeroporto_origine,
        aeroporto_destinazione,
        data_partenza,
        orario,
        tipo_volo,
        stato,
        ritardo
    ) VALUES (
        p_codice,
        p_compagnia_aerea,
        p_aeroporto_origine,
        p_aeroporto_destinazione,
        p_data_partenza,
        p_orario,
        p_tipo_volo,
        CASE
            WHEN p_ritardo > 0 THEN 'IN_RITARDO'::stato_volo
            ELSE 'PROGRAMMATO'::stato_volo
        END,
        p_ritardo
    );

END;
$$ LANGUAGE plpgsql;
```

4.5.9 Procedure elimina_volo

```
CREATE OR REPLACE PROCEDURE elimina_volo(
    p_admin_username VARCHAR(20),
    p_admin_password VARCHAR(20),
    p_codice_volo VARCHAR(10)
) AS $$
BEGIN
    -- Verifica le credenziali dell'amministratore
    IF NOT verifica_admin(p_admin_username, p_admin_password) THEN
        RAISE EXCEPTION 'Credenziali amministratore non valide';
    END IF;

    -- Verifica che il volo esista
    IF NOT EXISTS (SELECT 1 FROM volo WHERE codice = p_codice_volo) THEN
        RAISE EXCEPTION 'Volo con codice % non trovato', p_codice_volo;
    END IF;

    -- Prima elimina eventuali riferimenti nella tabella gate
    DELETE FROM gate
    WHERE codice_volo = p_codice_volo;

    -- Poi elimina eventuali prenotazioni associate
    DELETE FROM prenotazione
    WHERE codice_volo = p_codice_volo;

    -- Infine elimina il volo
    DELETE FROM volo
    WHERE codice = p_codice_volo;

END;
$$ LANGUAGE plpgsql;
```

4.5.10 Procedure aggiorna_stato_volo

```
-- Procedura per aggiornare lo stato di un volo
CREATE OR REPLACE PROCEDURE aggiorna_stato_volo(
    p_admin_username VARCHAR(20),
    p_admin_password VARCHAR(20),
    p_codice_volo VARCHAR,
    p_nuovo_stato stato_volo,
    p_ritardo INTEGER
) AS $$
BEGIN
    -- Verifica le credenziali dell'amministratore
    IF NOT verifica_admin(p_admin_username, p_admin_password) THEN
        RAISE EXCEPTION 'Credenziali amministratore non valide';
    END IF;

    -- Verifica se il volo esiste
    IF NOT EXISTS (
        SELECT 1 FROM volo
        WHERE codice = p_codice_volo
    ) THEN
        RAISE EXCEPTION 'Volo con codice % non trovato', p_codice_volo;
    END IF;

    -- Verifica che il ritardo non sia negativo
    IF p_ritardo < 0 THEN
        RAISE EXCEPTION 'Il ritardo non può essere negativo';
    END IF;

    -- Se c'è un ritardo maggiore di 0, forza lo stato a IN_RITARDO
    IF p_ritardo > 0 THEN
        UPDATE volo
        SET stato = 'IN_RITARDO'::stato_volo,
            ritardo = p_ritardo
        WHERE codice = p_codice_volo;
    ELSE
        UPDATE volo
        SET stato = p_nuovo_stato,
            ritardo = p_ritardo
        WHERE codice = p_codice_volo;
    END IF;
END;
$$ LANGUAGE plpgsql;
```

4.5.11 Procedure aggiorna_stato_prenotazione

```
-- Procedura per aggiornare lo stato di una prenotazione
CREATE OR REPLACE PROCEDURE aggiorna_stato_prenotazione(
    p_admin_username VARCHAR(20),
    p_admin_password VARCHAR(20),
    p_numero_biglietto VARCHAR(20),
    p_nuovo_stato stato_prenotazione
) AS $$
BEGIN
    -- Verifica le credenziali dell'amministratore
    IF NOT verifica_admin(p_admin_username, p_admin_password) THEN
        RAISE EXCEPTION 'Credenziali amministratore non valide';
    END IF;

    -- Verifica se la prenotazione esiste
    IF NOT EXISTS (
        SELECT 1 FROM prenotazione
        WHERE numero_biglietto = p_numero_biglietto
    ) THEN
        RAISE EXCEPTION 'Prenotazione con numero biglietto % non trovata',
p_numero_biglietto;
    END IF;

    -- Aggiorna lo stato della prenotazione
    UPDATE prenotazione
    SET stato = p_nuovo_stato
    WHERE numero_biglietto = p_numero_biglietto;
END;
$$ LANGUAGE plpgsql;
```

4.5.12 Procedure assegna_gate

```
CREATE OR REPLACE PROCEDURE assegna_gate(
    p_admin_username VARCHAR(20),
    p_admin_password VARCHAR(20),
    p_codice_volo VARCHAR(10),
    p_numero_gate INTEGER

) AS $$
DECLARE
    v_tipo_volo VARCHAR(10);
    v_stato_volo stato_volo;
    v_gate_esistente INTEGER;
BEGIN
    -- Verifica che chi sta assegnando sia un amministratore
    IF NOT verifica_admin(p_admin_username, p_admin_password) THEN
        RAISE EXCEPTION 'Non hai i permessi per assegnare gate';
    END IF;

    -- Verifica che il volo esista e ottieni informazioni
    SELECT tipo_volo, stato
    INTO v_tipo_volo, v_stato_volo
    FROM volo
    WHERE codice = p_codice_volo;

    IF NOT FOUND THEN
        RAISE EXCEPTION 'Volo % non trovato', p_codice_volo;
    END IF;

    -- Verifica che il volo sia in partenza
    IF v_tipo_volo != 'PARTENZA' THEN
        RAISE EXCEPTION 'I gate possono essere assegnati solo ai voli in
partenza';
    END IF;

    -- Verifica che il volo non sia già partito o cancellato
    IF v_stato_volo IN ('DECOLLATO', 'ATTERRATO', 'CANCELLATO') THEN
        RAISE EXCEPTION 'Non è possibile assegnare un gate a un volo %',
v_stato_volo;
    END IF;

    -- Verifica che il gate non sia già assegnato a questo volo
    DELETE FROM gate WHERE codice_volo = p_codice_volo;

    -- Inserisci la nuova assegnazione
    INSERT INTO gate (numero_gate, codice_volo)
    VALUES (p_numero_gate, p_codice_volo);

EXCEPTION
    WHEN check_violation THEN
        RAISE EXCEPTION 'Il numero del gate deve essere positivo';
    WHEN unique_violation THEN
        RAISE EXCEPTION 'Il gate % è già assegnato', p_numero_gate;
    WHEN others THEN
        RAISE EXCEPTION 'Errore durante l''assegnazione del gate: %',
SQLERRM;
END;
$$ LANGUAGE plpgsql;
```