



# UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

---

Documentazione progetto:

Basi di dati I

**“Aeroporto DB”**

*Antonio Soritto N86004962*

*Carmine Onorato N86005342*

**A.A. 2024/2025**

# INDICE PROGETTO

## "Sommario"

<b>1. INTRODUZIONE</b>	<b>1</b>
1.1. Descrizione del problema	1
<b>2. PROGETTAZIONE CONCETTUALE</b>	<b>2</b>
2.1. Class Diagram	2
2.2. Ristrutturazione del Class Diagram	2
2.2.1. Analisi delle chiavi	3
2.2.2. Analisi degli attributi derivati	3
2.2.3. Analisi delle ridondanze	3
2.2.4. Analisi degli attributi strutturati	3
2.2.5. Analisi degli attributi a valore multiplo	3
2.2.6. Analisi delle gerarchie di specializzazione	3
2.3. Class Diagram Ristrutturato	4
2.4. Dizionario delle classi	4
2.5. Dizionario delle associazioni	7
2.6. Dizionario dei vincoli	8
<b>3. PROGETTAZIONE LOGICA</b>	<b>10</b>
3.1. Schema Logico	10
<b>4. PROGETTAZIONE FISICA</b>	<b>11</b>
4.1. Definizione Tabelle	11
4.1.1. Definizione dei tipi ENUM	11
4.1.2. Tabella Utente	11
4.1.3. Tabella Passeggero	12
4.1.4. Tabella Amministratore	12
4.1.5. Tabella Gate	12
4.1.6. Tabella VoloOrigine	12
4.1.7. Tabella VoloDestinazione	13
4.1.8. Tabella Prenotazione	14
4.1.9. Tabella VoloAtteso	14
4.1.10. Tabella Amministra	14
4.1.11. Sequence	14
4.2. Implementazione dei Vincoli	15
4.2.1. Vincolo posto unico per volo	15
4.2.2. Validazione stato volo per prenotazioni	15

4.2.3.	Aggiornamento automatico stato volo .....	16
4.2.4.	Assegnazione unica gate per volo .....	16
4.2.5.	Validazione assegnazione amministratori .....	17
4.2.6.	Consistenza stato prenotazione .....	17
4.2.7.	Aggiornamento automatico stato volo (2) .....	18
5.	<b>FUNZIONI, PROCEDURE E ALTRE AUTOMAZIONI</b> .....	18
5.1.	Prenotazione Volo (Utente) .....	18
5.2.	Creazione Volo destinazione (Amministratore) .....	19
5.3.	Creazione Volo origine (Amministratore) .....	12
5.4.	Ricerca Voli (Per Tutti) .....	21

# Progetto di Basi di Dati

## Gestionale per l'aeroporto di Napoli

Carmine Onorato  
N86005342

Antonio Soritto  
N86004962

## 1 Introduzione

Il presente elaborato descrive la progettazione e lo sviluppo di un database relazionale basato su **PostgreSQL**, realizzato dagli studenti **Carmine Onorato** ed **Antonio Soritto** del Corso di Laurea in Informatica dell'Università degli Studi di Napoli Federico II. Il seguente progetto valutativo per l'insegnamento di Basi di Dati, implementa un gestionale per l'aeroporto di Napoli.

### 1.1 Descrizione del problema

Verranno riportate progettazione e sviluppo di una base di dati relazionale, che implementi un gestionale che permetta l'organizzazione e gestione di voli, passeggeri e di tutto ciò che concerne l'aeroporto di Napoli.

Si può accedere al sistema con due ruoli differenti (*amministratore* o *utente*), identificati da login e password.

**L'utente** può:

- effettuare prenotazioni per i voli programmati;
- modificare le prenotazioni;
- cercare le prenotazioni in base al nome del passeggero o al numero del volo.

**L'amministratore** può:

- inserire nuovi voli in arrivo o in partenza;
- aggiornare le informazioni dei voli esistenti (es. orario previsto, ritardo, stato);
- eliminare voli non più programmati;
- visualizzare lo stato generale dei voli e delle prenotazioni.

Ogni volo è caratterizzato da:

- un codice univoco;
- una compagnia aerea;
- l'aeroporto di origine (per i voli in arrivo a Napoli);
- l'aeroporto di destinazione (per i voli in partenza da Napoli);
- la data del volo;

- l'orario previsto;
- l'eventuale ritardo;
- lo stato del volo.

## 2 Progettazione Concettuale

In questa sezione del progetto descriviamo la progettazione del database al livello di astrazione più elevato. Partendo dall'analisi dei requisiti, si costruirà uno **schema concettuale** indipendente dalla struttura fisica dei dati, rappresentato con un *Class Diagram UML*. Il diagramma mostrerà le entità principali, le relazioni tra di esse e i vincoli da applicare.

### 2.1 Class Diagram

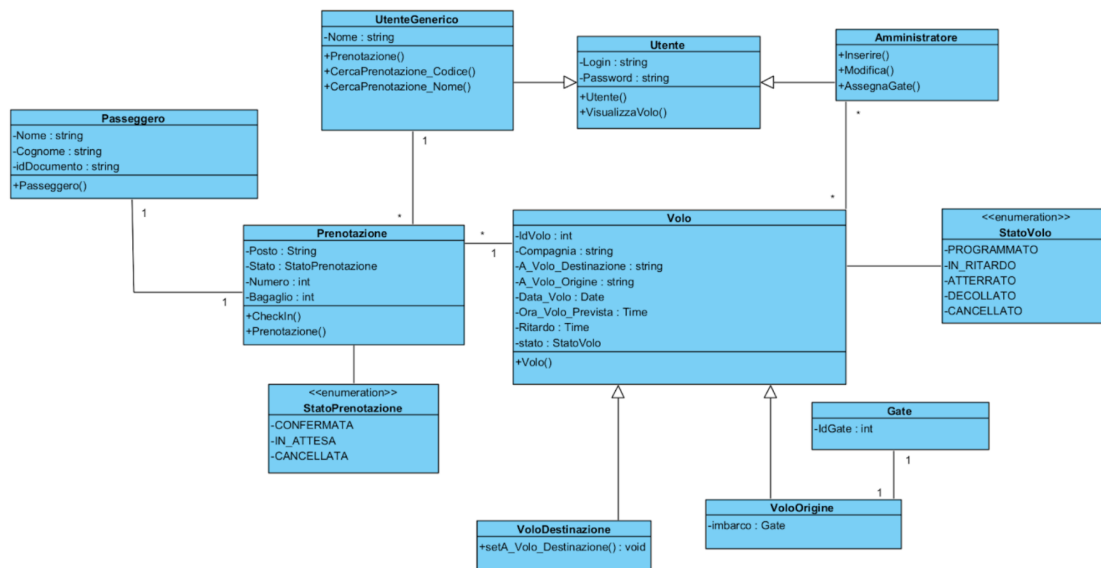


Figura 1: Diagramma UML delle classi del sistema gestionale aeroportuale

### 2.2 Ristrutturazione del Class Diagram

In questa fase ristrutturiamo il Class Diagram per prepararlo alla conversione in schema relazionale e ottimizzare le sue prestazioni. L'intervento seguirà questi step:

- 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, nello specifico Utente, Prenotazione, Volo, Passeggero e Gate, risulta conveniente l'introduzione di chiavi primarie surrogate, rappresentate da identificativi di tipo intero o stringa univoca (es. idPrenotazione, idVolo, idGate, idDocumento, Login). Tali identificatori non derivano da attributi descrittivi ma vengono assegnati per garantire l'univocità e facilitare l'accesso rapido alle istanze, riducendo i costi computazionali nelle operazioni di join tra le tabelle.

### **2.2.2 Analisi degli attributi derivati**

Per ottimizzare ulteriormente l'utilizzo delle risorse di calcolo, si è proceduto all'analisi degli attributi derivati, ovvero calcolabili a partire da altri dati già presenti nel sistema. In questo modello non sono stati individuati attributi significativamente derivabili, tali da giustificare la loro materializzazione. Alcuni esempi potrebbero essere la durata del ritardo di un volo, derivabile dalla differenza tra l'orario previsto e l'orario effettivo, ma si è deciso di trattarlo come attributo base (Ritardo) per semplificare le interrogazioni e migliorare l'efficienza in fase operativa.

### **2.2.3 Analisi delle ridondanze**

Analizziamo ora la eventuale presenza di associazioni ridondanti tra le varie entità, in maniera tale da evitare incoerenze nella rappresentazione logica dei dati. Il caso più evidente è rappresentato dal doppio legame tra Prenotazione, Utente e Passeggero: ogni prenotazione è effettuata da un utente, ma è anche intestata a un passeggero. Poiché il passeggero può essere diverso dall'utente (es. in caso di prenotazione per terzi), si è scelto di mantenere entrambe le associazioni, evitando ogni forma di ridondanza. Non risultano altre relazioni logicamente ridondanti nel modello.

### **2.2.4 Analisi degli attributi strutturati**

Vanno ora analizzati e concettualmente corretti eventuali attributi strutturati presenti nelle entità. Questi infatti non sono logicamente rappresentabili all'interno di un DBMS, e vanno quindi eliminati o codificati in altro modo. Nella rappresentazione concettuale presentata, non sono presenti attributi strutturati: ogni attributo è atomico e compatibile con la rappresentazione relazionale. Si è quindi ritenuto conforme il modello da questo punto di vista.

### **2.2.5 Analisi degli attributi a valore multiplo**

Verifichiamo ora la presenza di eventuali attributi a valore multiplo, anch'essi non logicamente rappresentabili e quindi da eliminare nello schema concettuale ristrutturato. Nel caso specifico, non si riscontra la presenza di attributi a valori multipli: ad esempio, ciascun volo è collegato a un solo gate, ciascuna prenotazione ha un solo posto, e ciascun passeggero ha un solo documento. Pertanto non è stato necessario introdurre entità ausiliarie per la rappresentazione di questi valori.

### **2.2.6 Analisi delle gerarchie di specializzazione**

Infine, andiamo a ristrutturare eventuali gerarchie di specializzazione, altro elemento non rappresentabile in un DBMS relazionale. Il modello originario prevedeva due gerarchie principali: la prima tra UtenteGenerico, Utente e Amministratore, la seconda tra Volo, VoloOrigine e VoloDestinazione. In entrambi i casi, si è scelto di eliminare la classe astratta e distribuire gli attributi del padre direttamente nelle entità figlie, secondo la tecnica dell'appiattimento. Tale

scelta è giustificata dal fatto che le entità risultanti operano in modo autonomo. In tal modo si evita la complessità della gestione di vincoli interclasse e si semplifica la successiva traduzione in schema logico relazionale.

## 2.3 Class Diagram Ristrutturato

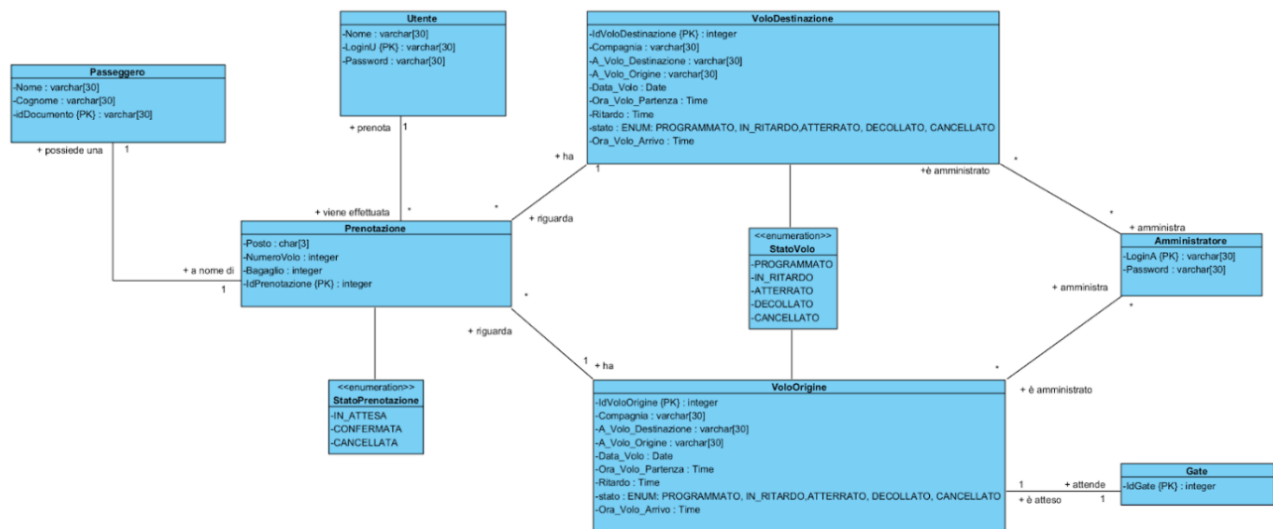


Figura 2: Diagramma UML ristrutturato per la traduzione in schema relazionale

## 2.4 Dizionario delle classi

### Utente

**Descrizione:** Rappresenta un utente generico del sistema, che può prenotare voli.

**Attributi:**

- **LoginU** (varchar[30]) Identificativo univoco dell'utente. **(PK)**
  - Formato: 3-30 caratteri alfanumerici o underscore
- **Password** (varchar[30]) Password dell'utente.
  - Lunghezza minima: 8 caratteri
- **Nome** (varchar[30]) Nome dell'utente.
  - Formato: 2-30 caratteri alfabetici con accenti e spazi

## Passeggero

**Descrizione:** Persona fisica per la quale viene effettuata una prenotazione.

**Attributi:**

- **idDocumento** (varchar[30]) Identificativo univoco del passeggero. **(PK)**
  - Formato: 5-30 caratteri alfanumerici
- **Nome** (varchar[30]) Nome del passeggero.
  - Stesso formato del Nome in Utente
- **Cognome** (varchar[30]) Cognome del passeggero.
  - Stesso formato del Nome in Utente

## Prenotazione

**Descrizione:** Rappresenta una prenotazione effettuata per un passeggero su un volo.

**Attributi:**

- **idPrenotazione** (integer) Identificativo univoco della prenotazione. **(PK)**
- **Posto** (char[3]) Codice del posto assegnato.
  - Formato: 1-2 numeri + 1 lettera (es: "12A")
- **Stato** (ENUM) Stato della prenotazione.
  - Valori ammessi: IN\_ATTESA, CONFERMATA, CANCELLATA
- **NumeroVolo** (integer) Id volo legato alla prenotazione.
  - Range: 1-9999
- **Bagaglio** (integer) Numero di bagagli associati.
  - Range: 0-10

## Gate

**Descrizione:** Indica il gate presso il quale è previsto il volo.

**Attributi:**

- **idGate** (integer) Identificativo univoco del gate. **(PK)**
  - Range: 1-999



## VoloOrigine

**Descrizione:** Descrive le informazioni relative a un volo in partenza.

**Attributi:**

- `idVoloOrigine` (integer) Identificativo univoco del volo. **(PK)**
- `Compagnia` (varchar[30]) Nome della compagnia aerea.
  - Formato: 3-30 caratteri alfanumerici con spazi, & e punto
- `A_Volo_Destinazione` (varchar[30]) Aeroporto di destinazione.
  - Formato: 3 lettere maiuscole (codice IATA)
- `A_Volo_Origine` (varchar[30]) Aeroporto di partenza.
  - Formato: 3 lettere maiuscole (codice IATA)
  - Vincolo: diverso da aeroporto destinazione
- `Data_Volo` (date) Data del volo.
  - Non può essere nel passato
- `Ora_Volo_Partenza` (time) Ora prevista del volo.
- `Ora_Volo_Arrivo` (time) Ora atterraggio.
- `Ritardo` (time) Eventuale ritardo previsto.
  - Non negativo
- `Stato` (ENUM) Stato corrente del volo.
  - Valori ammessi: PROGRAMMATO, IN\_RITARDO, ATTERRATO, DECOLLATO, CANCELLATO

## VoloDestinazione

**Descrizione:** Descrive le informazioni relative a un volo in arrivo.

**Attributi:** (analoghi a VoloOrigine)

- `idVoloDestinazione` (integer) Identificativo univoco del volo. **(PK)**
- `Compagnia` (varchar[30]) Nome della compagnia aerea.
- `A_Volo_Destinazione` (varchar[30]) Aeroporto di destinazione.
- `A_Volo_Origine` (varchar[30]) Aeroporto di partenza.
- `Data_Volo` (date) Data del volo.
- `Ora_Volo_Partenza` (time) Ora prevista del volo.
- `Ritardo` (time) Eventuale ritardo previsto.
- `Stato` (ENUM) Stato corrente del volo.

## 2.5 Dizionario delle associazioni

### Prenota

**Classi coinvolte:** Utente Prenotazione

**Descrizione:** Un utente effettua una o più prenotazioni.

**Cardinalità:**

- **Utente:** 1
- **Prenotazione:** \*

**Note:** Ogni prenotazione è associata all'utente che l'ha effettuata.

### È a nome di

**Classi coinvolte:** Prenotazione Passeggero

**Descrizione:** Una prenotazione è intestata a un singolo passeggero.

**Cardinalità:**

- **Prenotazione:** 1
- **Passeggero:** 1

**Note:** Un passeggero può essere il destinatario di molte prenotazioni.

### Riguarda (Volo)

**Classi coinvolte:** Prenotazione Volo

**Descrizione:** Una prenotazione riguarda un volo specifico.

**Cardinalità:**

- **Prenotazione:** \*
- **Volo:** 1

**Note:** Un volo può avere più prenotazioni.

### È atteso

**Classi coinvolte:** Volo Gate

**Descrizione:** Un volo è previsto presso un gate.

**Cardinalità:**

- **Volo:** 1
- **Gate:** 1

**Note:** Ogni gate può ospitare più voli, ma un volo è assegnato a un solo gate.

### Amministra

**Classi coinvolte:** Amministratore Volo

**Descrizione:** Un amministratore è responsabile della gestione di uno o più voli.

**Cardinalità:**

- **Amministratore:** \*
- **Volo:** \*

**Note:** Ogni volo è amministrato dagli amministratori.

## 2.6 Dizionario dei vincoli

Nome del vincolo	Descrizione
Formato login utente	LoginU deve contenere tra 3 e 30 caratteri alfanumerici e/o underscore (_) secondo la regex <code>'^[a-zA-Z0-9_]{3,30}\$'</code> .
Formato nome e cognome	Campi Nome e Cognome devono contenere lettere e caratteri consentiti (accenti, spazi, apostrofi) tra 2 e 30 caratteri.
Password utente sicura	Lunghezza minima di 8 caratteri.
Password amministratore sicura	Lunghezza minima di 10 caratteri.
Login amministratore valido	Deve iniziare con <code>admin_</code> seguito da almeno 3 caratteri alfanumerici o underscore.
Formato documento passeggero	idDocumento deve contenere tra 5 e 30 caratteri alfanumerici.
Range codice gate valido	idGate deve essere compreso tra 1 e 999.
Formato aeroporti origine/destinazione	Devono essere codici di 3 lettere maiuscole (NAP, FCO, ecc.).

Compagnia aerea valida	Compagnia deve avere 3-30 caratteri, lettere, numeri, spazi e simboli &.
Aeroporti diversi nel volo	Origine e destinazione devono essere diversi (A_Volo_Origine <> A_Volo_Destinazione).
Data volo valida	Data_Volo deve essere maggiore o uguale alla data corrente (CURRENT_DATE).
Tempi coerenti di volo	Orario di partenza + ritardo deve essere precedente all'orario di arrivo.
Formato posto prenotazione	Posto deve rispettare la regex <code>^[0-9]{1,2}[A-Z]\$\</code> (es: 12B, 7A).
Numero bagagli accettato	Bagaglio compreso tra 0 e 10.
Unicità del posto per volo	Un trigger impedisce prenotazioni con stesso Posto sullo stesso idVolo.
Blocco prenotazioni per voli invalidi	Trigger impedisce prenotazioni su voli con stato CANCELLATO, DECOLLATO o ATTERRATO.
Aggiornamento automatico stato volo	Se l'orario di partenza è passato stato diventa DECOLLATO; se l'orario di arrivo è passato diventa ATTERRATO.

Gate assegnabile ad un solo volo alla volta	Trigger verifica che non ci siano sovrapposizioni temporali sullo stesso idGate tra voli.
Massimo voli per amministratore	Un amministratore non può essere assegnato a più di 50 voli contemporaneamente.
Validazione conferma prenotazione	Non è possibile confermare una prenotazione se il volo associato è CANCELLATO.
Conformità referenziale completa	Tutte le FOREIGN KEY tra tabelle (idDocumento, LoginU, idVolo, LoginA, idGate) sono definite e vincolano i dati a esistenza coerente.
Annullamento automatico prenotazioni	Quando un volo viene impostato come CANCELLATO, tutte le prenotazioni ad esso collegate vengono automaticamente aggiornate allo stato CANCELLATA. Il vincolo si applica sia ai voli in partenza (VoloOrigine) che in arrivo (VoloDestinazione).

## 3. Progettazione Logica

In questo capitolo tratteremo la seconda fase della progettazione, scendendo ad un livello di astrazione più basso rispetto al precedente. Lo schema concettuale verrà tradotto, anche grazie alla predisposizione conseguente la ristrutturazione, in uno schema logico, questa volta dipendente dalla struttura dei dati prescelta, nello specifico quella relazionale pura.

### 3.1. Schema Logico

Di seguito è riportato lo schema logico della base di dati. Le chiavi primarie sono sottolineate con una linea singola.

- **Utente** (LoginU, Nome, Password)

- **Passeggero** (idDocumento, Nome, Cognome)
- **Amministratore** (LoginA, Password)
- **VoloOrigine** (idVoloOrigine, Compagnia, A\_Volo\_Origine, A\_Volo\_Destinazione, Data\_Volo, Ora\_Volo\_Partenza, Ora\_Volo\_Arrivo, Ritardo, Stato)
- **VoloDestinazione** (idVoloDestinazione, Compagnia, A\_Volo\_Origine, A\_Volo\_Destinazione, Data\_Volo, Ora\_Volo\_Partenza, Ora\_Volo\_Arrivo, Ritardo, Stato)
- **Gate** (idGate)
- **Prenotazione** (idPrenotazione, Numero, Posto, Bagaglio, Stato, Login, idDocumento, idVolo)  
Login → Utente.LoginU  
idDocumento → Passeggero.idDocumento  
idVolo → VoloOrigine.idVoloOrigine (o VoloDestinazione.idVoloDestinazione)
- **VoloAtteso** (idVolo, idGate)  
idVolo → VoloOrigine.idVoloOrigine  
idGate → Gate.idGate
- **Amministra** (Login, idVolo)  
Login → Amministratore.LoginA  
idVolo → VoloOrigine.idVoloOrigine (o VoloDestinazione.idVoloDestinazione)

## 4. Progettazione Fisica

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

### 4.1 Definizione Tabelle

Di seguito sono riportate le definizioni delle tabelle con i relativi vincoli.

#### 4.1.1 Definizione dei tipi ENUM

```

1 CREATE TYPE stato_prenotazione AS ENUM ('IN_ATTESA', 'CONFERMATATA', '
  CANCELLATA');
2 CREATE TYPE stato_volo AS ENUM ('PROGRAMMATO', 'IN_RITARDO', 'ATTERRATO',
  'DECOLLATO', 'CANCELLATO');
```

Listing 1: Tipi enumerativi

#### 4.1.2 Tabella Utente

```

1 CREATE TABLE Utente (
2   LoginU VARCHAR(30) PRIMARY KEY
3   CHECK (LoginU ~ '^[a-zA-Z0-9_]{3,30}$'),
4   Nome VARCHAR(30) NOT NULL
5   CHECK (Nome ~ '^[a-zA-ZaàèéìòùÀÈÉÌÒÙ'' -]{2,30}$'),
6   Password VARCHAR(30) NOT NULL
```

```

7         CHECK (LENGTH(Password) >= 8)
8     );

```

Listing 2: Creazione tabella Utente

### 4.1.3 Tabella Passeggero

```

1 CREATE TABLE Passeggero (
2     idDocumento VARCHAR(30) PRIMARY KEY
3     CHECK (idDocumento ~ '^[A-Za-z0-9]{5,30}$'),
4     Nome VARCHAR(30) NOT NULL
5     CHECK (Nome ~ '^[a-zA-ZàèéìòùÀÈÉÌÒÙ'' -]{2,30}$'),
6     Cognome VARCHAR(30) NOT NULL
7     CHECK (Cognome ~ '^[a-zA-ZàèéìòùÀÈÉÌÒÙ'' -]{2,30}$')
8 );

```

Listing 3: Creazione tabella Passeggero

### 4.1.4 Tabella Amministratore

```

1 CREATE TABLE Amministratore (
2     LoginA VARCHAR(30) PRIMARY KEY
3     CHECK (LoginA ~ '^admin_[a-zA-Z0-9_]{3,25}$'),
4     Password VARCHAR(30) NOT NULL
5     CHECK (LENGTH(Password) >= 10)
6 );

```

Listing 4: Creazione tabella Amministratore

### 4.1.5 Tabella Gate

```

1 CREATE TABLE Gate (
2     idGate INTEGER PRIMARY KEY
3     CHECK (idGate BETWEEN 1 AND 999)
4 );

```

Listing 5: Creazione tabella Gate

### 4.1.6 Tabella VoloOrigine

```

1 CREATE TABLE VoloOrigine (
2     idVoloOrigine INTEGER PRIMARY KEY,
3     Compagnia VARCHAR(30) NOT NULL
4     CHECK (Compagnia ~ '^[A-Za-z0-9 &.] {3,30}$'),
5     A_Volo_Origine VARCHAR(30) NOT NULL
6     CHECK (A_Volo_Origine = 'NAP'),
7     A_Volo_Destinazione VARCHAR(30) NOT NULL
8     CHECK (A_Volo_Destinazione ~ '^[A-Z]{3}$'),
9     Data_Volo DATE NOT NULL

```

```

10         CHECK (Data_Volo >= CURRENT_DATE),
11     Ora_Volo_Partenza TIME NOT NULL,
12     Ora_Volo_Arrivo TIME NOT NULL,
13     Ritardo INTERVAL
14         CHECK (Ritardo >= INTERVAL '0 minutes'),
15     Stato stato_volo NOT NULL,
16     CONSTRAINT Aeroporti_Diversi_Origine
17         CHECK (A_Volo_Origine <> A_Volo_Destinazione),
18     CONSTRAINT Tempi_Coerenti_Origine
19         CHECK ((Data_Volo + Ora_Volo_Partenza + Ritardo) < (Data_Volo +
20 Ora_Volo_Arrivo))
    );

```

Listing 6: Creazione tabella VoloOrigine

#### 4.1.7 Tabella VoloDestinazione

```

1 CREATE TABLE VoloDestinazione (
2     idVoloDestinazione INTEGER PRIMARY KEY,
3     Compagnia VARCHAR(30) NOT NULL
4         CHECK (Compagnia ~ '^[A-Za-z0-9 &.] {3,30}$'),
5     A_Volo_Origine VARCHAR(30) NOT NULL
6         CHECK (A_Volo_Origine ~ '^[A-Z] {3}$'),
7     A_Volo_Destinazione VARCHAR(30) NOT NULL
8         CHECK (A_Volo_Destinazione = 'NAP'),
9     Data_Volo DATE NOT NULL
10         CHECK (Data_Volo >= CURRENT_DATE),
11     Ora_Volo_Partenza TIME NOT NULL,
12     Ora_Volo_Arrivo TIME NOT NULL,
13     Ritardo INTERVAL
14         CHECK (Ritardo >= INTERVAL '0 minutes'),
15     Stato stato_volo NOT NULL,
16     CONSTRAINT Aeroporti_Diversi_Destinazione
17         CHECK (A_Volo_Origine <> A_Volo_Destinazione),
18     CONSTRAINT Tempi_Coerenti_Destinazione
19         CHECK ((Data_Volo + Ora_Volo_Partenza + Ritardo) < (Data_Volo +
20 Ora_Volo_Arrivo))
    );

```

Listing 7: Creazione tabella VoloDestinazione



#### 4.1.8 Tabella Prenotazione

```
1 CREATE TABLE Prenotazione (  
2     idPrenotazione INTEGER PRIMARY KEY,  
3     NumeroVolo INTEGER NOT NULL  
4         CHECK (NumeroVolo BETWEEN 1 AND 9999),  
5     Posto CHAR(3) NOT NULL  
6         CHECK (Posto ~ '^[0-9]{1,2}[A-Z]$'),  
7     Bagaglio INTEGER NOT NULL DEFAULT 0  
8         CHECK (Bagaglio BETWEEN 0 AND 10),  
9     Stato stato_prenotazione NOT NULL,  
10    LoginU VARCHAR(30) NOT NULL,  
11    idDocumento VARCHAR(30) NOT NULL,  
12    idVolo INTEGER NOT NULL,  
13    FOREIGN KEY (LoginU) REFERENCES Utente(LoginU),  
14    FOREIGN KEY (idDocumento) REFERENCES Passeggero(idDocumento),  
15    FOREIGN KEY (idVolo) REFERENCES VoloOrigine(idVoloOrigine)  
16 );
```

Listing 8: Creazione tabella Prenotazione

#### 4.1.9 Tabella VoloAtteso

```
1 CREATE TABLE VoloAtteso (  
2     idVolo INTEGER PRIMARY KEY,  
3     idGate INTEGER NOT NULL,  
4     FOREIGN KEY (idVolo) REFERENCES VoloOrigine(idVoloOrigine),  
5     FOREIGN KEY (idGate) REFERENCES Gate(idGate)  
6 );
```

Listing 9: Creazione tabella VoloAtteso

#### 4.1.10 Tabella Amministra

```
1 CREATE TABLE Amministra (  
2     LoginA VARCHAR(30),  
3     idVolo INTEGER,  
4     PRIMARY KEY (LoginA, idVolo),  
5     FOREIGN KEY (LoginA) REFERENCES Amministratore(LoginA),  
6     FOREIGN KEY (idVolo) REFERENCES VoloOrigine(idVoloOrigine)  
7 );
```

Listing 10: Creazione tabella Amministra

#### 4.1.11 Creazione Sequence

```
1 CREATE SEQUENCE seq_idprenotazione  
2 START WITH 10  
3 INCREMENT BY 1;
```

Listing 11: Creazione Sequence

## 4.2 Implementazione dei Vincoli

Di seguito sono riportati i trigger e le funzioni aggiuntive per l'implementazione dei vincoli.

### 4.2.1 Vincolo posto unico per volo

```
1 CREATE OR REPLACE FUNCTION check_unique_seat()
2 RETURNS TRIGGER AS $$
3 BEGIN
4     IF EXISTS (
5         SELECT 1 FROM Prenotazione
6         WHERE idVolo = NEW.idVolo AND Posto = NEW.Posto
7     ) THEN
8         RAISE EXCEPTION 'Posto % già occupato sul volo %', NEW.Posto, NEW
          .idVolo;
9     END IF;
10    RETURN NEW;
11 END;
12 $$ LANGUAGE plpgsql;
13
14 CREATE TRIGGER trg_unique_seat
15 BEFORE INSERT OR UPDATE ON Prenotazione
16 FOR EACH ROW EXECUTE FUNCTION check_unique_seat();
```

Listing 12: Trigger per posto unico

### 4.2.2 Validazione stato volo per prenotazioni

```
1 CREATE OR REPLACE FUNCTION check_flight_status_for_reservation()
2 RETURNS TRIGGER AS $$
3 DECLARE
4     flight_status stato_volo;
5 BEGIN
6     SELECT Stato INTO flight_status
7     FROM VoloOrigine WHERE idVoloOrigine = NEW.idVolo;
8
9     IF flight_status = 'CANCELLATO' THEN
10         RAISE EXCEPTION 'Impossibile prenotare su volo cancellato';
11     ELSIF flight_status IN ('DECOLLATO', 'ATTERRATO') THEN
12         RAISE EXCEPTION 'Impossibile prenotare su volo completato';
13     END IF;
14    RETURN NEW;
15 END;
16 $$ LANGUAGE plpgsql;
17
18 CREATE TRIGGER trg_flight_status_reservation
19 BEFORE INSERT OR UPDATE ON Prenotazione
20 FOR EACH ROW EXECUTE FUNCTION check_flight_status_for_reservation();
```

Listing 13: Trigger per validazione stato volo

### 4.2.3 Aggiornamento automatico stato volo

```
1 CREATE OR REPLACE FUNCTION update_flight_status()
2 RETURNS TRIGGER AS $$
3 DECLARE
4     flight_time TIMESTAMP;
5     arrival_time TIMESTAMP;
6 BEGIN
7     flight_time := (NEW.Data_Volo + NEW.Ora_Volo_Partenza)::TIMESTAMP;
8     arrival_time := (NEW.Data_Volo + NEW.Ora_Volo_Arrivo)::TIMESTAMP;
9
10    IF NEW.Stato = 'PROGRAMMATO' AND NOW() >= flight_time THEN
11        NEW.Stato := 'DECOLLATO';
12    ELSIF NEW.Stato = 'DECOLLATO' AND NOW() >= arrival_time THEN
13        NEW.Stato := 'ATTERRATO';
14    END IF;
15
16    RETURN NEW;
17 END;
18 $$ LANGUAGE plpgsql;
19
20 CREATE TRIGGER trg_update_flight_status_origine
21 BEFORE UPDATE ON VoloOrigine
22 FOR EACH ROW EXECUTE FUNCTION update_flight_status();
23
24 CREATE TRIGGER trg_update_flight_status_destinazione
25 BEFORE UPDATE ON VoloDestinazione
26 FOR EACH ROW EXECUTE FUNCTION update_flight_status();
```

Listing 14: Trigger per aggiornamento stato volo

### 4.2.4 Assegnazione unica gate per volo

```
1 CREATE OR REPLACE FUNCTION check_single_gate_assignment()
2 RETURNS TRIGGER AS $$
3 BEGIN
4     IF EXISTS (
5         SELECT 1 FROM VoloAtteso
6         WHERE idGate = NEW.idGate
7         AND idVolo <> NEW.idVolo
8         AND EXISTS (
9             SELECT 1 FROM VoloOrigine v1, VoloOrigine v2
10            WHERE v1.idVoloOrigine = NEW.idVolo
11            AND v2.idVoloOrigine = idVolo
12            AND v1.Data_Volo = v2.Data_Volo
13            AND (v1.Ora_Volo_Partenza, v1.Ora_Volo_Arrivo) OVERLAPS
14                (v2.Ora_Volo_Partenza, v2.Ora_Volo_Arrivo)
15        )
16     ) THEN
17         RAISE EXCEPTION 'Gate già assegnato ad un altro volo in questo
18 orario';
19     END IF;
20     RETURN NEW;
21 END;
22 $$ LANGUAGE plpgsql;
```

```

23 CREATE TRIGGER trg_single_gate_assignment
24 BEFORE INSERT OR UPDATE ON VoloAtteso
25 FOR EACH ROW EXECUTE FUNCTION check_single_gate_assignment();

```

Listing 15: Trigger per assegnazione gate

#### 4.2.5 Validazione assegnazione amministratori

```

1 CREATE OR REPLACE FUNCTION check_admin_assignment()
2 RETURNS TRIGGER AS $$
3 BEGIN
4     IF (SELECT COUNT(*) FROM Amministra WHERE LoginA = NEW.LoginA) >= 50
5     THEN
6         RAISE EXCEPTION 'Un amministratore non può gestire più di 50 voli
7         ';
8     END IF;
9     RETURN NEW;
10 END;
11 $$ LANGUAGE plpgsql;
12
13 CREATE TRIGGER trg_admin_assignment
14 BEFORE INSERT OR UPDATE ON Amministra
15 FOR EACH ROW EXECUTE FUNCTION check_admin_assignment();

```

Listing 16: Trigger per validazione amministratori

#### 4.2.6 Consistenza stato prenotazione

```

1 CREATE OR REPLACE FUNCTION check_reservation_status()
2 RETURNS TRIGGER AS $$
3 DECLARE
4     flight_status stato_volo;
5 BEGIN
6     IF NEW.Stato = 'CONFERMATA' THEN
7         SELECT Stato INTO flight_status
8         FROM VoloOrigine WHERE idVoloOrigine = NEW.idVolo;
9
10        IF flight_status = 'CANCELLATO' THEN
11            RAISE EXCEPTION 'Impossibile confermare prenotazione per volo
12            cancellato';
13        END IF;
14    END IF;
15    RETURN NEW;
16 END;
17 $$ LANGUAGE plpgsql;
18
19 CREATE TRIGGER trg_reservation_status
20 BEFORE UPDATE ON Prenotazione
21 FOR EACH ROW EXECUTE FUNCTION check_reservation_status();

```

Listing 17: Trigger per consistenza prenotazioni

#### 4.2.7 Aggiornamento automatico stato volo (2)

```
1 CREATE OR REPLACE FUNCTION cancel_bookings_on_flight_cancellation()
2 RETURNS TRIGGER AS $$
3 BEGIN
4     IF NEW.Stato = 'CANCELLATO' AND OLD.Stato <> 'CANCELLATO' THEN
5         UPDATE Prenotazione SET Stato = 'CANCELLATA'
6         WHERE idVolo = NEW.idVoloOrigine OR idVolo = NEW.
          idVoloDestinazione;
7     END IF;
8     RETURN NEW;
9 END;
10 $$ LANGUAGE plpgsql;
11
12 CREATE TRIGGER trg_cancel_bookings_on_flight_cancellation_origine
13 AFTER UPDATE ON VoloOrigine
14 FOR EACH ROW
15 EXECUTE FUNCTION cancel_bookings_on_flight_cancellation();
16
17 CREATE TRIGGER trg_cancel_bookings_on_flight_cancellation_destinazione
18 AFTER UPDATE ON VoloDestinazione
19 FOR EACH ROW
20 EXECUTE FUNCTION cancel_bookings_on_flight_cancellation();
```

Listing 18: Trigger per annullamento prenotazioni

## 5. Funzioni, Procedure e altre automazioni

Di seguito sono riportate le stored function e le stored procedures che si è deciso di implementare per semplificare alcuni aspetti dell'utilizzo della base di dati.

### 5.1 Prenotazione Volo (Utente)

Permette a un utente registrato di prenotare un volo, di seguito la definizione:

```
1 CREATE OR REPLACE FUNCTION crea_prenotazione(
2     p_numerovolo INTEGER,
3     p_posto CHAR(3),
4     p_bagaglio INTEGER,
5     p_loginu VARCHAR(30),
6     p_iddocumento VARCHAR(30),
7     p_idvolo INTEGER
8 ) RETURNS INTEGER AS $$
9 DECLARE
10     nuovo_id INTEGER;
11 BEGIN
12     nuovo_id := nextval('seq_idprenotazione');
13
14     INSERT INTO prenotazione (
15         idprenotazione,
16         numerovolo,
17         posto,
18         bagaglio,
19         stato,
20         loginu,
```

```

21         iddocumento,
22         idvolo
23     ) VALUES (
24         nuovo_id,
25         p_numerovolo,
26         p_posto,
27         p_bagaglio,
28         'CONFERMATA'::stato_prenotazione,
29         p_loginu,
30         p_iddocumento,
31         p_idvolo
32     );
33
34     RETURN nuovo_id;
35 END;
36 $$ LANGUAGE plpgsql;

```

## 5.2 Creazione Volo destinazione (Amministratore)

Permette agli amministratori di creare nuovi voli che arrivano a napoli, di seguito la definizione:

```

1  CREATE OR REPLACE FUNCTION crea_volo_destinazione(
2      p_id_volo INTEGER,
3      p_login_amministratore VARCHAR (30),
4      p_compagnia VARCHAR (30),
5      p_aeroporto_partenza VARCHAR (30),
6      p_aeroporto_destinazione VARCHAR (30),
7      p_data DATE,
8      p_ora_partenza TIME,
9      p_ora_arrivo TIME
10 ) RETURNS INTEGER AS $$
11 BEGIN
12     IF NOT EXISTS (SELECT 1 FROM Amministratore WHERE LoginA =
13         p_login_amministratore) THEN
14         RAISE EXCEPTION 'Amministratore non trovato';
15     END IF;
16
17     INSERT INTO VoloDestinazione (
18         idVoloDestinazione,
19         Compagnia,
20         A_Volo_Origine,
21         A_Volo_Destinazione,
22         Data_Volo,
23         Ora_Volo_Partenza,
24         Ora_Volo_Arrivo,
25         Stato
26     ) VALUES (
27         p_id_volo,
28         p_compagnia,
29         p_aeroporto_partenza,
30         p_aeroporto_destinazione,
31         p_data,
32         p_ora_partenza,
33         p_ora_arrivo,
34         'PROGRAMMATO'
35     );
36     return p_id_volo;
37 END;

```

```
37 $$ LANGUAGE plpgsql;
```

### 5.3 Creazione Volo origine(Amministratore)

Permette agli amministratori di creare nuovi voli che partono a napoli, di seguito la definizione:

```
1
2 CREATE OR REPLACE FUNCTION crea_volo_origine(
3     p_id_volo INTEGER,
4     p_login_amministratore VARCHAR(30),
5     p_compagnia VARCHAR(30),
6     p_aeroporto_partenza VARCHAR(30),
7     p_aeroporto_destinazione VARCHAR(30),
8     p_data DATE,
9     p_ora_partenza TIME,
10    p_ora_arrivo TIME,
11    p_gate integer
12 ) RETURNS integer AS $$
13 BEGIN
14     IF NOT EXISTS (SELECT 1 FROM Amministratore WHERE LoginA =
15         p_login_amministratore) THEN
16         RAISE EXCEPTION 'Amministratore non trovato';
17     END IF;
18     INSERT INTO VoloOrigine (
19         idVoloOrigine,
20         Compagnia,
21         A_Volo_Origine,
22         A_Volo_Destinazione,
23         Data_Volo,
24         Ora_Volo_Partenza,
25         Ora_Volo_Arrivo,
26         Stato
27     ) VALUES (
28         p_id_volo,
29         p_compagnia,
30         p_aeroporto_partenza,
31         p_aeroporto_destinazione,
32         p_data,
33         p_ora_partenza,
34         p_ora_arrivo,
35         'PROGRAMMATO'
36     );
37
38     INSERT INTO Amministra (LoginA, idVolo)
39     VALUES (p_login_amministratore, p_id_volo);
40
41     INSERT INTO VoloAtteso (idVolo, idGate)
42     VALUES (p_id_volo, p_gate);
43
44     return p_id_volo;
45 END;
46 $$ LANGUAGE plpgsql;
```

## 5.4 Ricerca Voli (Per Tutti)

Ricerca voli disponibili con filtri, di seguito la definizione:

```
1  CREATE OR REPLACE FUNCTION cerca_voli(  
2      p_aeroporto_partenza TEXT DEFAULT NULL,  
3      p_aeroporto_destinazione TEXT DEFAULT NULL,  
4      p_data DATE DEFAULT NULL,  
5      p_tipo_volo TEXT DEFAULT 'partenza'  
6  ) RETURNS TABLE (  
7      id_volo INTEGER,  
8      numero_volo VARCHAR(30),  
9      compagnia VARCHAR(30),  
10     aeroporto_partenza VARCHAR(30),  
11     aeroporto_destinazione VARCHAR(30),  
12     data_ora_partenza TIMESTAMP,  
13     stato stato_volo  
14 ) AS $$  
15  
16 BEGIN  
17     IF p_tipo_volo = 'partenza' THEN  
18         RETURN QUERY  
19             SELECT  
20                 vo.idVoloOrigine,  
21                 vo.Compagnia,  
22                 vo.A_Volo_Origine,  
23                 vo.A_Volo_Destinazione,  
24                 (vo.Data_Volo + vo.Ora_Volo_Partenza)::TIMESTAMP,  
25                 vo.Stato  
26             FROM VoloOrigine vo  
27             WHERE  
28                 (p_aeroporto_partenza IS NULL OR vo.A_Volo_Origine =  
29                 p_aeroporto_partenza)  
30                 AND (p_aeroporto_destinazione IS NULL OR vo.  
31                 A_Volo_Destinazione = p_aeroporto_destinazione)  
32                 AND (p_data IS NULL OR vo.Data_Volo = p_data)  
33                 AND vo.Data_Volo >= CURRENT_DATE  
34             ORDER BY vo.Data_Volo, vo.Ora_Volo_Partenza;  
35     ELSE  
36         RETURN QUERY  
37             SELECT  
38                 vd.idVoloDestinazione,  
39                 vd.Compagnia,  
40                 vd.A_Volo_Origine,  
41                 vd.A_Volo_Destinazione,  
42                 (vd.Data_Volo + vd.Ora_Volo_Partenza)::TIMESTAMP,  
43                 vd.Stato  
44             FROM VoloDestinazione vd  
45             WHERE  
46                 (p_aeroporto_partenza IS NULL OR vd.A_Volo_Origine =  
47                 p_aeroporto_partenza)  
48                 AND (p_aeroporto_destinazione IS NULL OR vd.  
49                 A_Volo_Destinazione = p_aeroporto_destinazione)  
50                 AND (p_data IS NULL OR vd.Data_Volo = p_data)  
51                 AND vd.Data_Volo >= CURRENT_DATE  
52             ORDER BY vd.Data_Volo, vd.Ora_Volo_Partenza;  
53     END IF;  
54 END;  
55  
56 $$ LANGUAGE plpgsql;
```