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

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 linsegnamento di Basi di Dati, implementa un gestionale per laeroporto di Napoli.

1.1 Descrizione del problema

Verranno riportate progettazione e sviluppo di una base di dati relazionale, che implementi un gestionale che permetta lorganizzazione 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 dallanalisi 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

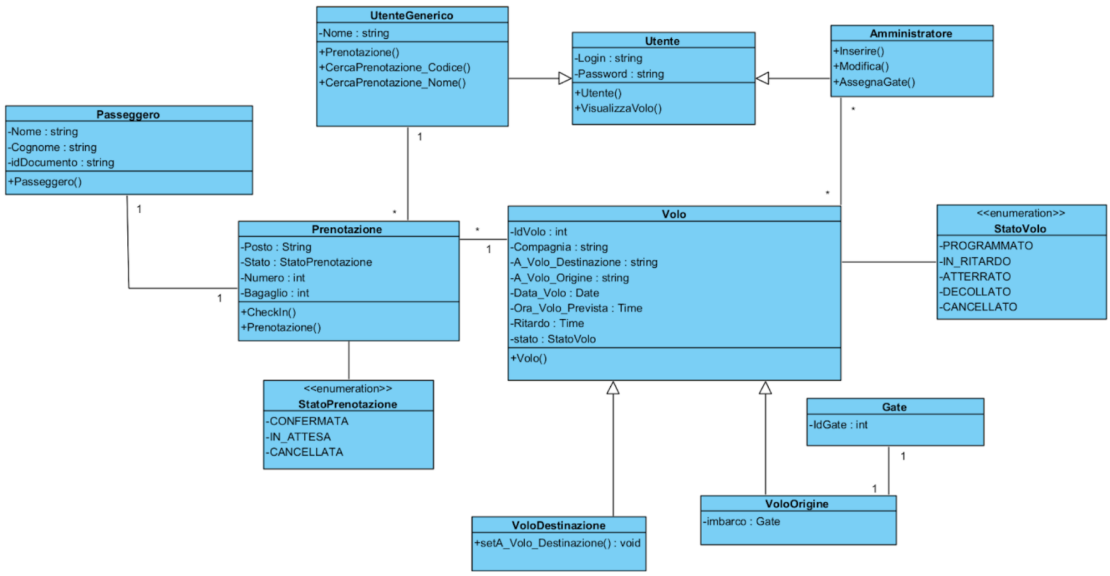

Figura 1: Diagramma UML delle classi del sistema gestionale aeroportuale

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. Lintervento 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 dellefficienza nella rappresentazione delle varie entità, nello specifico Utente, Prenotazione, Volo, Passeggero e Gate, risulta conveniente lintroduzione 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 lunivocità e facilitare laccesso rapido alle istanze, riducendo i costi computazionali nelle operazioni di join tra le tabelle.

2.2.2 Analisi degli attributi derivati

Per ottimizzare ulteriormente lutilizzo delle risorse di calcolo, si è proceduto allanalisi 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 lorario previsto e lorario effettivo, ma si è deciso di trattarlo come attributo base (Ritardo) per semplificare le interrogazioni e migliorare lefficienza in fase operativa.

2.2.3 Analisi delle ridondanze

Analizziamo ora leventuale 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 dallutente (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 allinterno 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, anchessi 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 dellappiattimento. 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

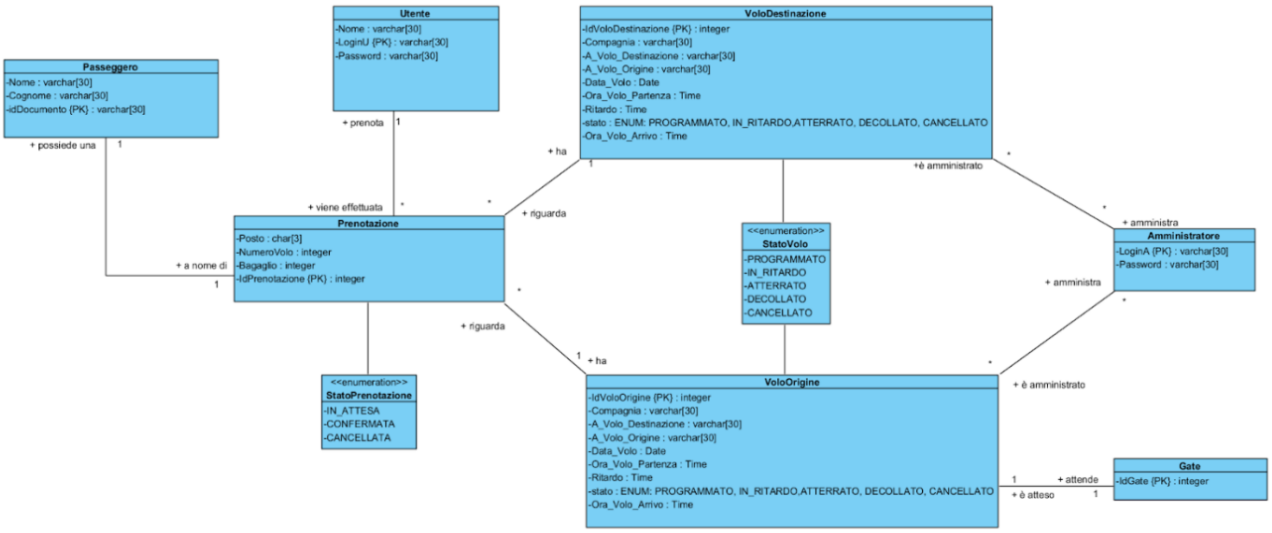

Figura 2: Diagramma UML ristrutturato per la traduzione in schema relazionale

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 '^[a-zA-Z0-9_]{3,30}$'.
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 admin_ 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 '^[0-9]{1,2}[A-Z]$' (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.
Generazione automatica numero volo	Trigger genera automaticamente NumeroVolo come (idVoloOrigine % 9000) + 1000.

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

Listing 1: Tipi enumerativi

4.1.2 Tabella Utente

```
CREATE TABLE Utente (
    LoginU VARCHAR(30) PRIMARY KEY
        CHECK (LoginU ~ '^[a-zA-Z0-9_]{3,30}$'),
    Nome VARCHAR(30) NOT NULL
        CHECK (Nome ~ '^[a-zA-ZaàèéìòùÀÈÉÌÒÙ'' -]{2,30}$'),
    Password VARCHAR(30) NOT NULL
        CHECK (LENGTH(Password) >= 8)
);
```

Listing 2: Creazione tabella Utente

4.1.3 Tabella Passeggero

```
CREATE TABLE Passeggero (
    idDocumento VARCHAR(30) PRIMARY KEY
        CHECK (idDocumento ~ '^[A-Za-z0-9]{5,30}$'),
    Nome VARCHAR(30) NOT NULL
        CHECK (Nome ~ '^[a-zA-ZaàèéìòùÀÈÉÌÒÙ'' -]{2,30}$'),
    Cognome VARCHAR(30) NOT NULL
        CHECK (Cognome ~ '^[a-zA-ZaàèéìòùÀÈÉÌÒÙ'' -]{2,30}$')
);
```

Listing 3: Creazione tabella Passeggero

4.1.4 Tabella Amministratore

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

Listing 4: Creazione tabella Amministratore

4.1.5 Tabella Gate

```
CREATE TABLE Gate (
    idGate INTEGER PRIMARY KEY
        CHECK (idGate BETWEEN 1 AND 999)
);
```

Listing 5: Creazione tabella Gate

4.1.6 Tabella VoloOrigine

```
CREATE TABLE VoloOrigine (
    idVoloOrigine INTEGER PRIMARY KEY,
    Compagnia VARCHAR(30) NOT NULL
        CHECK (Compagnia ~ '^[A-Za-z0-9 &.]{3,30}$'),
    A_Volo_Origine VARCHAR(30) NOT NULL
        CHECK (A_Volo_Origine ~ '^[A-Z]{3}$'),
    A_Volo_Destinazione VARCHAR(30) NOT NULL
        CHECK (A_Volo_Destinazione ~ '^[A-Z]{3}$'),
    Data_Volo DATE NOT NULL
        CHECK (Data_Volo >= CURRENT_DATE),
    Ora_Volo_Partenza TIME NOT NULL,
    Ora_Volo_Arrivo TIME NOT NULL,
    Ritardo INTERVAL
        CHECK (Ritardo >= INTERVAL '0 minutes'),
    Stato stato_volo NOT NULL,
    CONSTRAINT Aeroporti_Diversi_Origine
        CHECK (A_Volo_Origine <> A_Volo_Destinazione),
    CONSTRAINT Tempi_Coerenti_Origine
        CHECK ((Data_Volo + Ora_Volo_Partenza + Ritardo) < (Data_Volo +
    Ora_Volo_Arrivo))
);
```

Listing 6: Creazione tabella VoloOrigine

4.1.7 Tabella VoloDestinazione

```
CREATE TABLE VoloDestinazione (
    idVoloDestinazione INTEGER PRIMARY KEY,
    Compagnia VARCHAR(30) NOT NULL
        CHECK (Compagnia ~ '^[A-Za-z0-9 &.]{3,30}$'),
    A_Volo_Origine VARCHAR(30) NOT NULL
        CHECK (A_Volo_Origine ~ '^[A-Z]{3}$'),
    A_Volo_Destinazione VARCHAR(30) NOT NULL
        CHECK (A_Volo_Destinazione ~ '^[A-Z]{3}$'),
    Data_Volo DATE NOT NULL
        CHECK (Data_Volo >= CURRENT_DATE),
    Ora_Volo_Partenza TIME NOT NULL,
    Ora_Volo_Arrivo TIME NOT NULL,
    Ritardo INTERVAL
        CHECK (Ritardo >= INTERVAL '0 minutes'),
    Stato stato_volo NOT NULL,
    CONSTRAINT Aeroporti_Diversi_Destinazione
        CHECK (A_Volo_Origine <> A_Volo_Destinazione),
    CONSTRAINT Tempi_Coerenti_Destinazione
        CHECK ((Data_Volo + Ora_Volo_Partenza + Ritardo) < (Data_Volo +
    Ora_Volo_Arrivo))
);
```

Listing 7: Creazione tabella VoloDestinazione

4.1.8 Tabella Prenotazione

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

Listing 8: Creazione tabella Prenotazione

4.1.9 Tabella VoloAtteso

```
CREATE TABLE VoloAtteso (
    idVolo INTEGER PRIMARY KEY,
    idGate INTEGER NOT NULL,
    FOREIGN KEY (idVolo) REFERENCES VoloOrigine(idVoloOrigine),
    FOREIGN KEY (idGate) REFERENCES Gate(idGate)
);
```

Listing 9: Creazione tabella VoloAtteso

4.1.10 Tabella Amministra

```
CREATE TABLE Amministra (
    LoginA VARCHAR(30),
    idVolo INTEGER,
    PRIMARY KEY (LoginA, idVolo),
    FOREIGN KEY (LoginA) REFERENCES Amministratore(LoginA),
    FOREIGN KEY (idVolo) REFERENCES VoloOrigine(idVoloOrigine)
);
```

Listing 10: Creazione tabella Amministra

4.1.11 Sequenze

```
CREATE SEQUENCE seq_volo_origine START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_volo_destinazione START WITH 1 INCREMENT BY 1;
CREATE SEQUENCE seq_prenotazione START WITH 1 INCREMENT BY 1;
```

Listing 11: Creazione sequenze

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

```
CREATE OR REPLACE FUNCTION check_unique_seat()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM Prenotazione
        WHERE idVolo = NEW.idVolo AND Posto = NEW.Posto
    ) THEN
        RAISE EXCEPTION 'Posto % già occupato sul volo %', NEW.Posto, NEW.idVolo;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_unique_seat
BEFORE INSERT OR UPDATE ON Prenotazione
FOR EACH ROW EXECUTE FUNCTION check_unique_seat();
```

Listing 12: Trigger per posto unico

4.2.2 Validazione stato volo per prenotazioni

```
CREATE OR REPLACE FUNCTION check_flight_status_for_reservation()
RETURNS TRIGGER AS $$
DECLARE
    flight_status stato_volo;
BEGIN
    SELECT Stato INTO flight_status
    FROM VoloOrigine WHERE idVoloOrigine = NEW.idVolo;

    IF flight_status = 'CANCELLATO' THEN
        RAISE EXCEPTION 'Impossibile prenotare su volo cancellato';
    ELSIF flight_status IN ('DECOLLATO', 'ATTERRATO') THEN
        RAISE EXCEPTION 'Impossibile prenotare su volo completato';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_flight_status_reservation
BEFORE INSERT OR UPDATE ON Prenotazione
FOR EACH ROW EXECUTE FUNCTION check_flight_status_for_reservation();
```

Listing 13: Trigger per validazione stato volo

4.2.3 Aggiornamento automatico stato volo

```
CREATE OR REPLACE FUNCTION update_flight_status()
RETURNS TRIGGER AS $$
DECLARE
    flight_time TIMESTAMP;
    arrival_time TIMESTAMP;
BEGIN
    flight_time := (NEW.Data_Volo + NEW.Ora_Volo_Partenza)::TIMESTAMP;
    arrival_time := (NEW.Data_Volo + NEW.Ora_Volo_Arrivo)::TIMESTAMP;

    IF NEW.Stato = 'PROGRAMMATO' AND NOW() >= flight_time THEN
        NEW.Stato := 'DECOLLATO';
    ELSIF NEW.Stato = 'DECOLLATO' AND NOW() >= arrival_time THEN
        NEW.Stato := 'ATTERRATO';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_update_flight_status_origine
BEFORE UPDATE ON VoloOrigine
FOR EACH ROW EXECUTE FUNCTION update_flight_status();

CREATE TRIGGER trg_update_flight_status_destinazione
BEFORE UPDATE ON VoloDestinazione
FOR EACH ROW EXECUTE FUNCTION update_flight_status();
```

Listing 14: Trigger per aggiornamento stato volo

4.2.4 Assegnazione unica gate per volo

```
CREATE OR REPLACE FUNCTION check_single_gate_assignment()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM VoloAtteso
        WHERE idGate = NEW.idGate
        AND idVolo <> NEW.idVolo
        AND EXISTS (
            SELECT 1 FROM VoloOrigine v1, VoloOrigine v2
            WHERE v1.idVoloOrigine = NEW.idVolo
            AND v2.idVoloOrigine = idVolo
            AND v1.Data_Volo = v2.Data_Volo
            AND (v1.Ora_Volo_Partenza, v1.Ora_Volo_Arrivo) OVERLAPS
                (v2.Ora_Volo_Partenza, v2.Ora_Volo_Arrivo)
        )
    ) THEN
        RAISE EXCEPTION 'Gate già assegnato ad un altro volo in questo
   orario';
    END IF;
    RETURN NEW;
END;
$$ 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 $$
3  BEGIN
4      IF (SELECT COUNT(*) FROM Amministra WHERE LoginA = NEW.LoginA) >= 50
       THEN
5          RAISE EXCEPTION 'Un amministratore non può gestire più di 50 voli
       ';
6      END IF;
7      RETURN NEW;
8  END;
9  $$ LANGUAGE plpgsql;
10
11 CREATE TRIGGER trg_admin_assignment
12 BEFORE INSERT OR UPDATE ON Amministra
13 FOR EACH ROW EXECUTE FUNCTION check_admin_assignment();
```

Listing 16: Trigger per validazione amministratori

4.2.6 Generazione automatica numero volo

```
1  CREATE OR REPLACE FUNCTION generate_flight_number()
2  RETURNS TRIGGER AS $$
3  BEGIN
4      IF NEW.idVoloOrigine IS NOT NULL THEN
5          NEW.idVoloOrigine := (NEW.idVoloOrigine % 9000) + 1000;
6      END IF;
7          IF NEW.idVoloDestinazione IS NOT NULL THEN
8          NEW.idVoloDestinazione := (NEW.idVoloDestinazione % 9000) + 1000;
9      END IF;
10     RETURN NEW;
11 END;
12 $$ LANGUAGE plpgsql;
13
14 CREATE TRIGGER trg_generate_flight_number_origine
15 BEFORE INSERT ON VoloOrigine
16 FOR EACH ROW EXECUTE FUNCTION generate_flight_number();
17
18 CREATE TRIGGER trg_generate_flight_number_destinazione
19 BEFORE INSERT ON VoloDestinazione
20 FOR EACH ROW EXECUTE FUNCTION generate_flight_number();
```

Listing 17: Trigger per generazione numero volo

4.2.7 Consistenza stato prenotazione

```
CREATE OR REPLACE FUNCTION check_reservation_status()
RETURNS TRIGGER AS $$
DECLARE
    flight_status stato_volo;
BEGIN
    IF NEW.Stato = 'CONFERMATA' THEN
        SELECT Stato INTO flight_status
        FROM VoloOrigine WHERE idVoloOrigine = NEW.idVolo;

        IF flight_status = 'CANCELLATO' THEN
            RAISE EXCEPTION 'Impossibile confermare prenotazione per volo cancellato';
        END IF;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_reservation_status
BEFORE UPDATE ON Prenotazione
FOR EACH ROW EXECUTE FUNCTION check_reservation_status();
```

Listing 18: Trigger per consistenza prenotazioni

4.2.8 Aggiornamento automatico stato volo (2)

```
CREATE OR REPLACE FUNCTION cancel_bookings_on_flight_cancellation()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.Stato = 'CANCELLATO' AND OLD.Stato <> 'CANCELLATO' THEN
        UPDATE Prenotazione SET Stato = 'CANCELLATA'
        WHERE idVolo = NEW.idVoloOrigine OR idVolo = NEW.idVoloDestinazione;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trg_cancel_bookings_on_flight_cancellation_origine
AFTER UPDATE ON VoloOrigine
FOR EACH ROW
EXECUTE FUNCTION cancel_bookings_on_flight_cancellation();

CREATE TRIGGER trg_cancel_bookings_on_flight_cancellation_destinazione
AFTER UPDATE ON VoloDestinazione
FOR EACH ROW
EXECUTE FUNCTION cancel_bookings_on_flight_cancellation();
```

Listing 19: 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:

```
CREATE OR REPLACE FUNCTION prenota_volo(
    p_login VARCHAR(30),
    p_documento VARCHAR(30),
    p_id_volo INTEGER,
    p_posto CHAR(3),
    p_bagaglio INTEGER DEFAULT 0
) RETURNS INTEGER AS $$
DECLARE
    nuova_prenotazione INTEGER;
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Utente WHERE LoginU = p_login) THEN
        RAISE EXCEPTION 'Utente non trovato';
    END IF;
        nuova_prenotazione := nextval('seq_prenotazione');

    INSERT INTO Prenotazione (
        idPrenotazione, NumeroVolo, Posto, Bagaglio, Stato, LoginU,
    idDocumento, idVolo
    ) VALUES (
        nuova_prenotazione,
        (SELECT NumeroVolo FROM VoloOrigine WHERE idVoloOrigine =
    p_id_volo),
        p_posto,
        p_bagaglio,
        'IN_ATTESA',
        p_login,
        p_documento,
        p_id_volo
    );

    RETURN nuova_prenotazione;
END;
$$ LANGUAGE plpgsql;
```

5.2 Creazione Volo (Amministratore)

Permette agli amministratori di creare nuovi voli, di seguito la definizione:

```
CREATE OR REPLACE PROCEDURE crea_volo(
    p_compagnia VARCHAR(30),
    p_aeroporto_partenza VARCHAR(3),
    p_aeroporto_destinazione VARCHAR(3),
    p_data DATE,
    p_ora_partenza TIME,
    p_ora_arrivo TIME,
    p_login_amministratore VARCHAR(30)
) AS $$
```

```
DECLARE
    nuovo_id INTEGER;
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Amministratore WHERE LoginA =
   p_login_amministratore) THEN
        RAISE EXCEPTION 'Amministratore non trovato';
    END IF;

    nuovo_id := nextval('seq_volo_origine');

    INSERT INTO VoloOrigine (
        idVoloOrigine, Compagnia, A_Volo_Origine, A_Volo_Destinazione,
        Data_Volo, Ora_Volo_Partenza, Ora_Volo_Arrivo, Stato
    ) VALUES (
        nuovo_id,
        p_compagnia,
        p_aeroporto_partenza,
        p_aeroporto_destinazione,
        p_data,
        p_ora_partenza,
        p_ora_arrivo,
        'PROGRAMMATO'
    );

    INSERT INTO Amministra (LoginA, idVolo) VALUES (
   p_login_amministratore, nuovo_id);
END;
$$ LANGUAGE plpgsql;
```

5.3 Ricerca Voli (Per Tutti)

Ricerca voli disponibili con filtri, di seguito la definizione:

```
CREATE OR REPLACE FUNCTION cerca_voli(
    p_aeroporto_partenza VARCHAR(3) DEFAULT NULL,
    p_aeroporto_destinazione VARCHAR(3) DEFAULT NULL,
    p_data DATE DEFAULT NULL
) RETURNS TABLE (
    id_volo INTEGER,
    numero_volo VARCHAR(10),
    compagnia VARCHAR(30),
    partenza VARCHAR(3),
    destinazione VARCHAR(3),
    data_ora_partenza TIMESTAMP,
    stato VARCHAR(20)
) AS $$
BEGIN
    RETURN QUERY
    SELECT
        vo.idVoloOrigine,
        vo.NumeroVolo,
        vo.Compagnia,
        vo.A_Volo_Origine,
        vo.A_Volo_Destinazione,
        (vo.Data_Volo + vo.Ora_Volo_Partenza)::TIMESTAMP,
        vo.Stato
    FROM
        VoloOrigine vo
```

```
26      WHERE
27          (p_aeroporto_partenza IS NULL OR vo.A_Volo_Origine =
        p_aeroporto_partenza)
28          AND (p_aeroporto_destinazione IS NULL OR vo.A_Volo_Destinazione =
         p_aeroporto_destinazione)
29          AND (p_data IS NULL OR vo.Data_Volo = p_data)
30          AND vo.Data_Volo >= CURRENT_DATE
31      ORDER BY vo.Data_Volo, vo.Ora_Volo_Partenza;
32  END;
33  $$ LANGUAGE plpgsql;
```