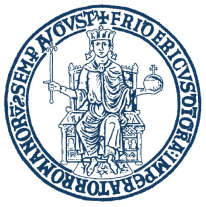


UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Documentazione Progetto Basi di Dati

Carmine Sgariglia (N86005069)
Mattia Lemma (N86005170)
Massimo Russo (N86005016)

Anno accademico 2024/2025

Indice

1 Introduzione **3**

2 Progettazione concettuale **4**

2.1 Class diagram 4

2.2 Ristrutturazione del Class Diagram 5

2.2.1 Analisi delle chiavi 5

2.2.2 Analisi degli attributi derivati 5

2.2.3 Analisi delle ridondanze 6

2.2.4 Analisi degli attributi strutturati 6

2.2.5 Analisi degli attributi a valore multiplo 6

2.2.6 Analisi delle generalizzazioni 6

2.3 Class Diagram Ristrutturato 7

2.4 Dizionario delle classi 8

2.5 Dizionario delle associazioni 10

2.6 Dizionario dei vincoli 11

3 Progettazione Logica **14**

3.1 Schema logico 14

4 Progettazione fisica **16**

4.1 Defnizione tabelle 16

4.1.1 Definizione dei tipi ENUM 16

4.1.2 Tabella Utente 17

4.1.3 Tabella Gate 17

4.1.4 Tabella Passeggero 17

4.1.5 Tabella Volo 18

4.1.6 Tabella Prenotazione 19

4.1.7 Tabella Bagaglio 19

4.2 Implementazione dei Vincoli 20

4.2.1 Gestione prenotazione Volo 20

4.2.2 Gestione documenti passeggero 21

4.2.3 Gestione stato prenotazione 21
4.2.4 Gestione stato bagaglio caricato 22
4.2.5 Gestione stato bagaglio ritirato 23

5 Funzioni, procedure e altre automazioni 25
5.1 Inserimento Volo (Amministratore) 25
5.2 Segnalazione bagaglio smarrito(Utente) 27
5.3 Ricerca Volo tramite destinazione, data, numero_posti(Tutti) 27

Link repository :

`https://github.com/CarmineSgariglia/BDD_Project.git`

Capitolo 1

Introduzione

Il progetto ha come obiettivo la realizzazione di un sistema informativo per supportare la gestione delle attività aeroportuali dell'aeroporto di Napoli. Il sistema è destinato a un utilizzo da parte sia del personale amministrativo dell'aeroporto sia dei passeggeri, e si propone di migliorare l'organizzazione, il monitoraggio e l'efficienza delle operazioni. Il sistema prevede l'accesso tramite credenziali personali, distinguendo due principali tipologie di utenti: gli utenti generici e gli amministratori. I primi hanno la possibilità di effettuare prenotazioni sui voli disponibili, consultare le informazioni a loro dedicate, ricevere aggiornamenti in tempo reale e segnalare bagagli smarriti. I secondi, invece, sono responsabili della gestione dei voli, dei gate di imbarco, dei bagagli e delle richieste di supporto da parte degli utenti. Il cuore del sistema è la gestione dei voli. Ogni volo in partenza o in arrivo viene monitorato e aggiornato dagli amministratori, includendo dati come l'orario previsto, eventuali ritardi, lo stato attuale e altre informazioni rilevanti. Gli amministratori possono inserire nuovi voli e modificarne le caratteristiche, mentre gli utenti possono visualizzare le informazioni e prenotare un eventuale posto disponibile a bordo. Il processo di prenotazione permette agli utenti di riservare un volo indicando i dati del passeggero, il numero del biglietto e il posto assegnato. Le prenotazioni possono essere cercate e gestite anche successivamente, rendendo l'esperienza utente più flessibile e personalizzata. Un altro aspetto centrale è la gestione dei gate, che vengono assegnati ai voli in partenza per facilitare le operazioni di imbarco. Anche questa funzione è riservata agli amministratori, che possono modificarne l'assegnazione secondo necessità operative. Un'ulteriore funzionalità rilevante riguarda il monitoraggio dei bagagli. Ogni bagaglio è collegato al volo del passeggero e viene tracciato durante tutte le fasi del viaggio. Il sistema consente di segnalare un eventuale smarrimento e permette agli amministratori di aggiornare lo stato del bagaglio, indicando ad esempio se è stato caricato sull'aereo o se è disponibile per il ritiro. Infine, il sistema supporta ricerche rapide e mirate su voli, passeggeri e bagagli, rendendo più semplice e immediata l'individuazione di informazioni rilevanti.

Capitolo 2

Progettazione concettuale

In questa sezione verrà illustrata la progettazione concettuale del sistema: definiremo entità, relazioni e vincoli di integrità attraverso il modello UML, al fine di rappresentare in modo chiaro e coerente la struttura informativa del database.

2.1 Class diagram

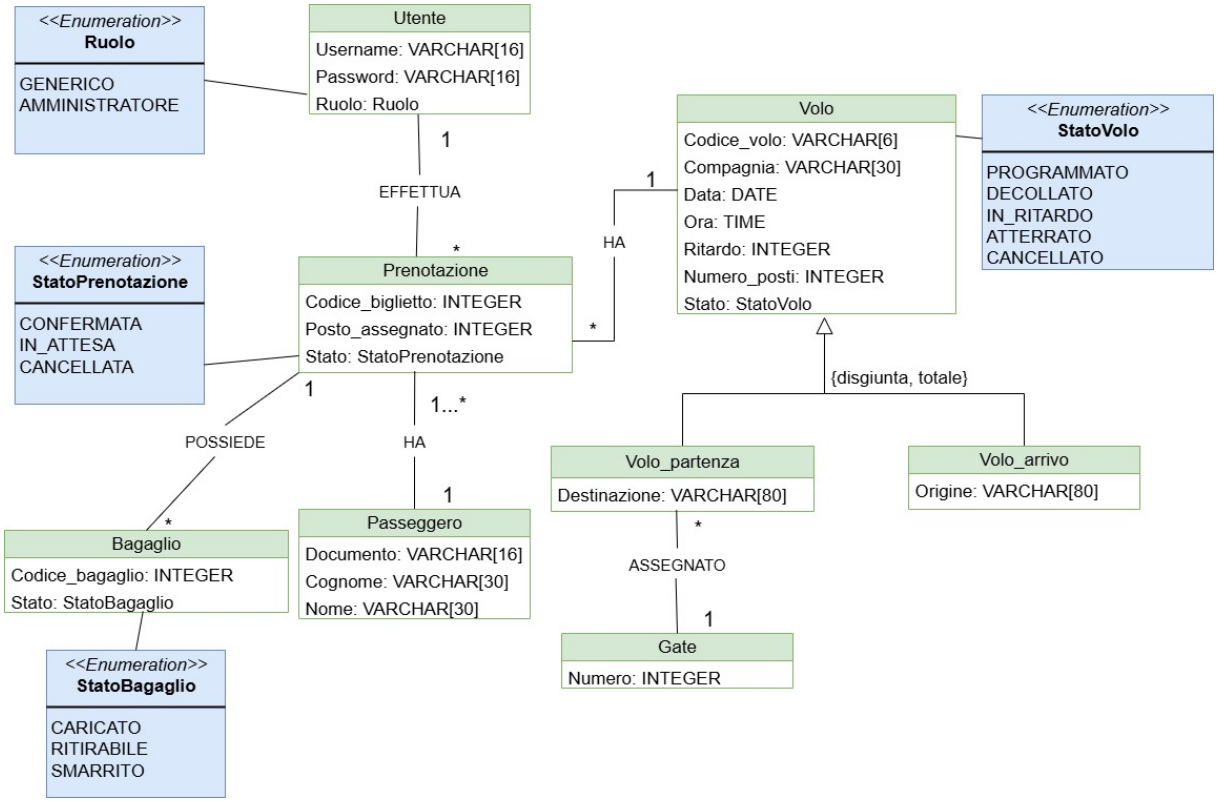


Figura 2.1: Diagramma UML delle entità e delle relazioni

2.2 Ristrutturazione del Class Diagram

In questa sezione procederemo alla ristrutturazione del diagramma concettuale per convertirlo in uno schema relazionale effettivo, pronto per l'implementazione come database. Di seguito le fasi operative che seguiremo:

1. **Analisi delle chiavi**
2. **Analisi degli attributi derivati**
3. **Analisi delle ridondanze**
4. **Analisi degli attributi strutturati**
5. **Analisi degli attributi a valore multiplo**
6. **Analisi delle gerarchie di specializzazione**

2.2.1 Analisi delle chiavi

Per garantire un'identificazione univoca e semplificare le operazioni di gestione dei record, abbiamo adottato una scelta mista di chiavi surrogate e naturali. Le chiavi surrogate vengono generate internamente dal sistema e risultano ideali per le entità più soggette a modifiche e cancellazioni: in particolare, **Codice_volo** identifica ogni `Volo`, **Codice_biglietto** contraddistingue ogni `Prenotazione` e un generico **Codice_bagaglio** numerico viene assegnato a ciascun `Bagaglio`. Questo approccio riduce il rischio di collisioni e semplifica le operazioni di aggiornamento dei collegamenti tra tabelle.

Per le altre entità, invece, abbiamo preferito chiavi naturali che riflettano elementi già esistenti nel mondo reale: il **Documento** del passeggero funge da chiave primaria nella tabella `Passeggero`, poiché ogni individuo possiede un documento d'identità univoco; il **Numero** del gate, assegnato direttamente dall'aeroporto, identifica in modo inequivocabile il punto di imbarco; infine, l'**Username** rappresenta la chiave primaria della tabella `Utente`, garantendo un meccanismo di autenticazione semplice ed efficace.

Questa strategia combina la robustezza delle chiavi surrogate con la trasparenza delle chiavi naturali, assicurando al contempo integrità referenziale, facilità di manutenzione e chiarezza semantica del modello dati.

2.2.2 Analisi degli attributi derivati

Nella presente modellazione non sono stati inseriti attributi derivati: ogni campo memorizza valori elementari e indipendenti, evitando calcoli o ricombinazioni di dati già presenti. Tale scelta elimina ridondanze e semplifica la manutenzione e l'integrità del database.

2.2.3 Analisi delle ridondanze

Per agevolare le interrogazioni e migliorare le performance del sistema, abbiamo introdotto un'attributo ridondante `Posti_occupati` nella tabella `Volo`. In questo modo, il numero di posti effettivamente occupati viene memorizzato direttamente e reso immediatamente disponibile, senza la necessità di calcolarlo dinamicamente a partire dalle prenotazioni associate. Questa scelta ottimizza i tempi di risposta alle query di monitoraggio dei voli, a fronte di un minimo aumento dello spazio di archiviazione.

2.2.4 Analisi degli attributi strutturati

Nel modello concettuale non sono presenti attributi strutturati: ogni proprietà è rappresentata come valore atomico semplice, evitando l'uso di campi complessi o aggregati. Questa scelta semplifica le query, riduce la complessità del mapping relazionale e migliora l'efficienza della gestione dei dati.

2.2.5 Analisi degli attributi a valore multiplo

Nel modello concettuale non sono presenti attributi a valore multiplo: ciascun attributo è definito per contenere un solo valore per istanza, evitando liste o insiemi di valori. Questa scelta facilita la normalizzazione, garantisce l'integrità dei dati e semplifica le operazioni di query e aggiornamento.

2.2.6 Analisi delle generalizzazioni

Infine, durante la ristrutturazione del modello abbiamo scelto di eliminare la generalizzazione tra voli in arrivo e voli in partenza, unificando le relative gerarchie nell'unica entità `Volo`. Per mantenere l'informazione territoriale, abbiamo aggiunto gli attributi `Origine` e `Destinazione` che possono essere impostati su "Napoli" per identificare la tipologia di volo. Inoltre, la relazione verso `Gate`, necessaria soltanto per i voli in partenza, è stata ridefinita con cardinalità `0..1`, in modo da rappresentare la possibile (ma non obbligatoria) assegnazione di un gate.

2.3 Class Diagram Ristrutturato

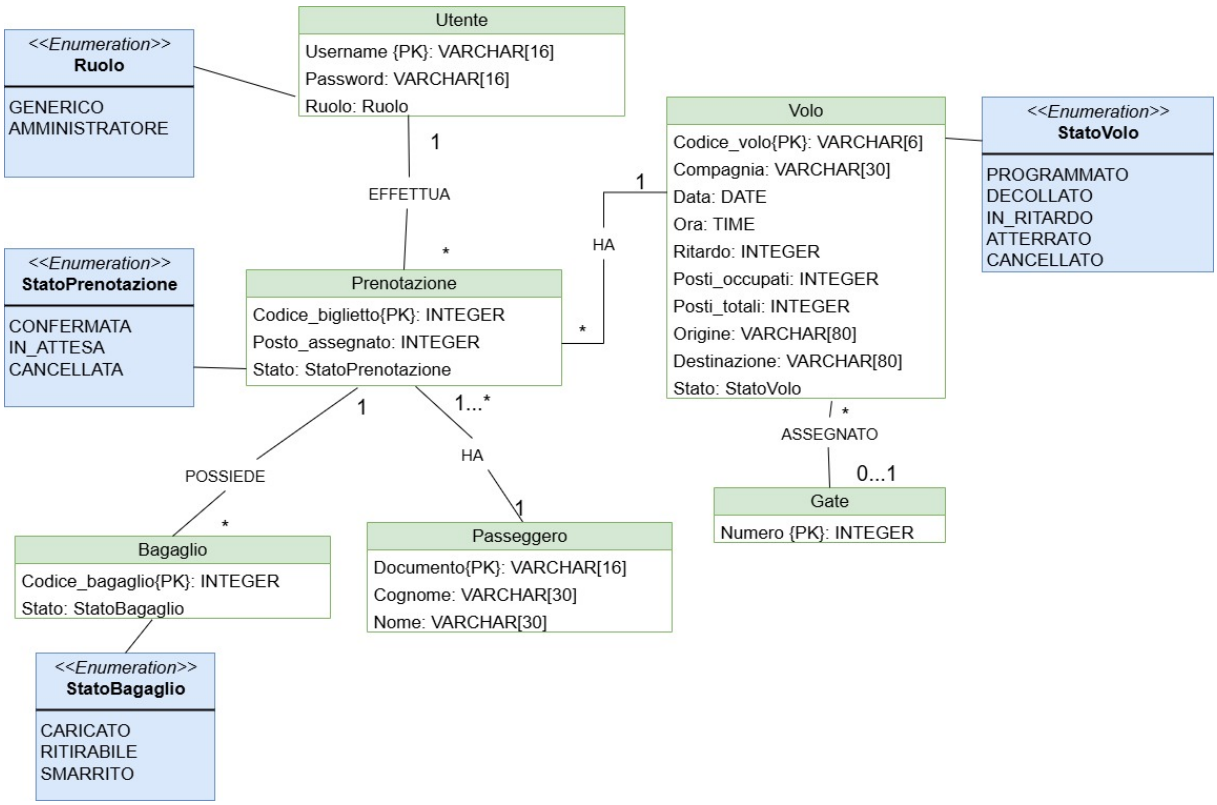


Figura 2.2: Diagramma UML ristrutturato delle entità e delle relazioni

2.4 Dizionario delle classi

Tabella 2.1: Dizionario delle classi

Classe	Descrizione	Attributi
Utente	Descrive ciascun utente (amministratore o generico) presente nel sistema; la tipologia è contraddistinta dal ruolo.	• **Username** (VARCHAR[16]): Nome utente univoco, massimo 16 caratteri. • **Password** (VARCHAR[16]): Codice di accesso alfanumerico, massimo 16 caratteri. • **Ruolo** (ENUM(amministratore, generico)): Ruolo assegnato all'utente.
Prenotazione	Descrive la prenotazione effettuata da un utente, collegata a un volo e a un passeggero.	• **Codice_biglietto** (INTEGER): Codice identificativo univoco del biglietto. • **Posto_assegnato** (INTEGER): Numero del posto sul volo. • **Stato** (ENUM(confermata, in_attesa, cancellata)): Stato aggiornato in tempo reale.
Passeggero	Descrive il passeggero per cui si effettua la prenotazione (può essere diverso dall'utente).	• **Documento** (VARCHAR[16]): Documento univoco del passeggero. • **Nome** (VARCHAR[30]): Nome del passeggero. • **Cognome** (VARCHAR[30]): Cognome del passeggero.

Continua nella pagina successiva

Classe	Descrizione	Attributi
Volo	Descrive il volo presente nella base dati, gestito dagli amministratori per la prenotazione.	• **Codice_volo** (VARCHAR[6]): Identificativo univoco del volo. • **Compagnia** (VARCHAR[30]): Compagnia che effettua il volo. • **Data** (DATE): Data prevista. • **Ora** (TIME): Orario previsto. • **Ritardo** (INTEGER): Ritardo in minuti. • **Posti_occupati** (INTEGER): Numero posti già prenotati. • **Posti_totali** (INTEGER): Numero totale di posti. • **Origine** (VARCHAR[80]): Aeroporto di partenza. • **Destinazione** (VARCHAR[80]): Aeroporto di arrivo. • **Stato** (ENUM(programmato, decollato, in_ritardo, atterrato, cancellato)): Stato in tempo reale.
Gate	Descrive i gate dell'aeroporto di Napoli per la gestione degli imbarchi.	• **Numero** (INTEGER): Identificativo univoco del gate.
Bagaglio	Descrive i bagagli registrati per un passeggero.	• **Codice_bagaglio** (INTEGER): Codice identificativo del bagaglio. • **Stato** (ENUM(caricato, ritirabile, smarrito)): Stato del bagaglio.

2.5 Dizionario delle associazioni

Tabella 2.2: Dizionario delle associazioni

Classe	Descrizione	Attributi
EFFETTUA	Descrive la partecipazione di un Utente nell'effettuare una prenotazione.	• **Utente** 1 • **Prenotazione** * • **Note**: Indica che un Utente può effettuare molteplici prenotazioni e che ogni prenotazione è riferita ad un utente
HA	Va ad indicare che ogni prenotazione ha un volo associato.	• **Volo** 1 • **Prenotazione** * • **Note**: Indica ogni prenotazione deve essere relegata ad un volo esistente e che un volo può avere più prenotazioni in base al numero di posti disponibili.
ASSEGNATO	Va ad indicare che ogni volo in partenza ha un gate di imbarco di riferimento.	• **Volo** * • **Gate** 0..1 • **Note**: Indica che ogni volo in Partenza ha un gate di imbarco gestito dall'Aeroporto e che un gate può avere più voli che partono da lì.
HA	Va ad indicare che ogni prenotazione ha un passeggero associato.	• **Prenotazione** 1..* • **Passeggero** 1 • **Note**: Indica che ogni prenotazione ha un passeggero e che un passeggero può avere più prenotazioni.

Continua nella pagina successiva

Nome	Descrizione	Classi coinvolte
POSSIEDE	Va ad indicare che ogni prenotazione ha dei bagagli associati	• **Bagaglio** * • **Prenotazione** 1 • **Note**: Indica che ogni prenotazione può avere dei bagagli e che ogni bagaglio ha una prenotazione associata

2.6 Dizionario dei vincoli

chk_numero_gate_positive
Vincolo di check che controlla che il numero di gate sia positivo
chk_documento_format
Vincolo di check che controlla che il documento sia composto solo da numeri e lettere maiuscole. primo quindi di punteggiatura.
chk_nome
Vincolo di check che controlla che il nome sia formato solo da lettere, spazi e dall'apostrofo
chk_cognome
Vincolo di check che controlla che il nome sia formato solo da lettere, spazi e dall'apostrofo
uq_prenotazione_posto_volo
Vincolo di unicità sulla coppia Codice_volo, Posto_assegnato
chk_no_status_change_if_flight_cancelled
Questo trigger impedisce di modificare lo stato di una prenotazione se il volo a cui è legata è stato cancellato

chk_prenotazione_volo_programmato

Questo trigger impedisce di prenotare un volo se il suo stato è diverso da programmato

trg_no_change_documento_when_finalized

Questo trigger impedisce di modificare il documento(e relativo passeggero) se la prenotazione è confermata o cancellata

chk_password_policy

Vincolo di ckeck che controlla che la password sia formata da 8 a 16 caratteri, almeno una lettera maiuscola, almeno una lettera minuscola, almeno un numero e almeno un simbolo

chk_username_format

Vincolo di check che controlla che l'username sia formato da da 3 a 16 caratteri e che contenga lettere miauscole, lettere minuscola e cifre

chk_codice_volo_format

Vincolo di check che controlla che il codice del volo sia formato da tre lettere maiuscole e minuscole e da tre numeri

chk_compagnia_len

Vincolo di check che controlla il nome della compagnia deve essere di almeno tre caratteri

chk_gate_only_departure

Vincolo di check che controlla che solo i voli in partenza da Napoli possano avere assegnato un gate

chk_Napoli_one

Vincolo di check che controlla che per ogni tupla origine o destinazione sia uguale a Napoli

chk_orig_dest_diff

Vincolo di check che controlla che per ogni tupla origine e destinazione debbano essere diverse

chk_posti_occupati_range

Vincolo di check che controlla che i posti occupati sono compresi tra 0 e numero di posti totali

chk_posti_totali_positive

Vincolo di check che controlla che i posti totali siano maggiori di 0

chk_ritardo_non_negative

Vincolo di check che controlla che il ritardo sia maggiore o uguale a 0

ux_volo_gate_schedule

Questo vincolo impedisce che due voli in partenza da Napoli partano dallo stesso gate esattamente nello stesso giorno e alla stessa ora

chk_bagaglio_only_if_pren_confirmed

Questo trigger stabilisce che un bagaglio può essere caricato solamente se la presentazione risulta confermata

chk_bagaglio_ritiro_volo_atterrato

Questo trigger stabilisce che un bagaglio può essere ritirato solo se il volo a cui è associato è atterrato

Capitolo 3

Progettazione Logica

La fase di progettazione logica rappresenta il passaggio intermedio tra la modellazione concettuale e l'effettiva implementazione fisica della base di dati. A partire dal modello concettuale, descritto attraverso entità, relazioni e vincoli, si procede alla definizione di uno schema relazionale capace di rappresentare in maniera fedele e completa le informazioni, mantenendo al contempo un'elevata efficienza nella memorizzazione e nell'accesso ai dati. In questa fase, le entità vengono mappate in tabelle, gli attributi sono tradotti in colonne con tipi di dato appropriati e vengono individuate le chiavi primarie e le chiavi esterne necessarie a garantire l'integrità referenziale. Il risultato è uno schema logico coerente e ottimizzato, pronto per essere implementato all'interno del sistema di gestione di basi di dati scelto, in questo caso PostgreSQL.

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** (Username, Password, Ruolo)
- **Volo** (Codice_volo, Compagnia, Data, Ora, Ritardo, Posti_occupati, Posti_totali, Origine, Destinazione, Stato)
- **Prenotazione** (Codice_biglietto, Posto_assegnato, Volo, Stato, Username, Documento)
  - Username → Utente.Username
  - Documento → Passeggero.Documento
  - Volo → Volo.Codice_volo
- **Passeggero** (Documento, Nome, Cognome)

- **Bagaglio** (Codice_bagaglio, Stato, Codice_biglietto)
  - Codice_biglietto → Prenotazione.Codice_biglietto
- **Gate** (Numero_gate)
- **Volo** (Codice_volo, Compagnia, Data, Ora, Ritardo, Posti_occupati, Posti_totali, Origine, Destinazione, Stato, Numero_gate)
  - Numero_gate → Gate.Numero_gate

Di seguito forniamo anche uno schema visuale della progettazione logica:

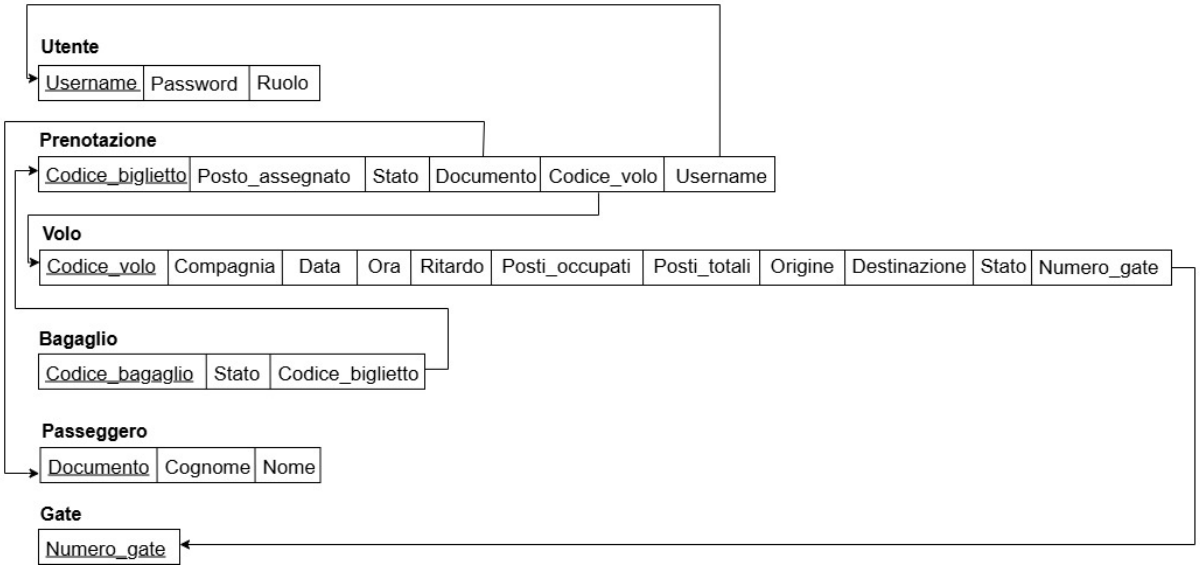


Figura 3.1: Schema logico

Capitolo 4

Progettazione fisica

La progettazione fisica è la fase in cui lo schema logico viene tradotto in un'implementazione concreta nel DBMS scelto. In essa si definiscono tipi di dato, vincoli, chiavi e indici, ottimizzando la struttura per garantire integrità, efficienza e sicurezza. L'obiettivo è ottenere un modello fisico coerente, performante e pronto all'uso, tenendo conto delle caratteristiche e delle ottimizzazioni offerte dal sistema di gestione adottato.

4.1 Defnizione tabelle

Di seguito forniamo l'implementazione SQL di tutte le tabelle nel DBMS scelto, in questo caso PostgreSQL

4.1.1 Definizione dei tipi ENUM

```
1
2  CREATE TYPE Ruolo AS ENUM
3      ('AMMINISTRATORE', 'GENERICO');
4  --
5  CREATE TYPE Statobagaglio AS ENUM
6      ('CARICATO', 'RITIRABILE', 'SMARRITO');
7  --
8  CREATE TYPE Statoprenotazione AS ENUM
9      ('CONFERMATA', 'IN_ATTESA', 'CANCELLATA');
10 --
11 CREATE TYPE Statovolo AS ENUM
12     ('PROGRAMMATO', 'DECOLLATO', 'IN_RITARDO', 'ATTERRATO', '
         CANCELLATO');
```

4.1.2 Tabella Utente

```
CREATE TABLE Utente (
  Username VARCHAR(16) PRIMARY KEY
    CONSTRAINT chk_username_format
      CHECK (Username ~ '^[A-Za-z0-9]{3,16}$'),

  Password VARCHAR(16) NOT NULL
    CONSTRAINT chk_password_policy
      CHECK (
        Password ~ '^.{8,16}$'
        AND Password ~ '[A-Z]'
        AND Password ~ '[a-z]'
        AND Password ~ '[0-9]'
        AND Password ~ '[!@#\$%\^&\*()_\+\-=\[\]{}|;:,.?]'
      ),

  Ruolo Ruolo NOT NULL
);
```

4.1.3 Tabella Gate

```
CREATE TABLE Gate (
  Numero_gate INTEGER PRIMARY KEY
    CONSTRAINT chk_numero_gate_positive CHECK (numero_gate > 0)
);
```

4.1.4 Tabella Passeggero

```
CREATE TABLE Passeggero (
  documento VARCHAR(16) PRIMARY KEY
    CONSTRAINT chk_documento_format
      CHECK (documento ~ '^[A-Z0-9]+$'),

  nome VARCHAR(30) NOT NULL
    CONSTRAINT chk_nome
      CHECK (
        char_length(nome) >= 3
        AND nome ~ '^[A-Za-z'' ]+$'
      ),
```

```
12
13    cognome VARCHAR(30) NOT NULL
14      CONSTRAINT chk_cognome
15        CHECK (
16          char_length(cognome) >= 3
17          AND cognome ~ '^[A-Za-z'' ]+$'
18        )
19  );
```

4.1.5 Tabella Volo

```
1   CREATE TABLE Volo (
2     Codice_volo     VARCHAR(6) PRIMARY KEY
3       CONSTRAINT chk_codice_volo_format CHECK (Codice_volo ~ '^[A-
          Za-z]{3}[0-9]{3}$'),
4
5     Compagnia       VARCHAR(30) NOT NULL
6       CONSTRAINT chk_compagnia_len CHECK (CHAR_LENGTH(Compagnia) >=
            3),
7
8     Data        DATE          NOT NULL,
9     Ora         TIME          NOT NULL,
10  Ritardo INTEGER NOT NULL DEFAULT 0
11      CONSTRAINT chk_ritardo_non_negative CHECK (Ritardo >= 0),
12    Posti_totali    INTEGER      NOT NULL
13      CONSTRAINT chk_posti_totali_positive CHECK (Posti_totali > 0)
          ,
14
15    Posti_occupati INTEGER      NOT NULL
16      DEFAULT 0
17      CONSTRAINT chk_posti_occupati_range
18        CHECK (Posti_occupati >= 0 AND Posti_occupati <=
            Posti_totali),
19
20    Origine         VARCHAR(80) NOT NULL,
21    Destinazione    VARCHAR(80) NOT NULL,
22
23    Stato           StatoVolo   NOT NULL,
24
25    Numero_gate     INTEGER
26      REFERENCES gate(Numero_gate),
```

```
27
28    CONSTRAINT chk_orig_dest_diff         CHECK (Origine <>
          Destinazione),
29    CONSTRAINT chk_napoli_one             CHECK (Origine = 'Napoli'
          OR Destinazione = 'Napoli'),
30    CONSTRAINT chk_gate_only_departure    CHECK (Numero_gate IS NULL
          OR Origine = 'Napoli')
31 );
32
33 CREATE UNIQUE INDEX ux_volo_gate_schedule
34    ON volo(Numero_gate, Data, Ora)
35    WHERE Origine = 'Napoli' AND Numero_gate IS NOT NULL;
```

4.1.6 Tabella Prenotazione

```
1  CREATE TABLE Prenotazione (
2    Codice_biglietto  SERIAL PRIMARY KEY,
3    Posto_assegnato     INTEGER      NOT NULL,
4    Stato               StatoPrenotazione NOT NULL,
5    Documento           VARCHAR(16) NOT NULL
6      REFERENCES Passeggero(Documento),
7    Codice_volo         VARCHAR(6)   NOT NULL
8      REFERENCES Volo(Codice_volo),
9    Username            VARCHAR(16) NOT NULL
10     REFERENCES Utente(Username),
11   CONSTRAINT uq_prenotazione_posto_volo
12     UNIQUE (Codice_volo, Posto_assegnato)
13 );
```

4.1.7 Tabella Bagaglio

```
1  CREATE TABLE Bagaglio (
2    Codice_bagaglio SERIAL PRIMARY KEY,
3    Stato             StatoBagaglio    NOT NULL,
4    Codice_biglietto INTEGER          NOT NULL
5      REFERENCES Prenotazione(Codice_biglietto)
6  );
```

4.2 Implementazione dei Vincoli

Di seguito sono riportati i trigger e le function necessarie all'implementazione dei vincoli interrelazionali.

4.2.1 Gestione prenotazione Volo

Non si possono prenotare voli che sono atterrati, decollati, in ritardo o cancellati.

```
CREATE OR REPLACE FUNCTION trg_check_prenotazione_stato()
  RETURNS trigger AS $$
DECLARE
  v_stato StatoVolo;
BEGIN

  SELECT Stato INTO v_stato FROM Volo WHERE Codice_volo = NEW.
     Codice_volo;

IF v_stato <> 'PROGRAMMATO' THEN
    RAISE EXCEPTION
      'Impossibile prenotare: il volo % siccome    nello stato %'
         , NEW.Codice_volo, v_stato;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER chk_prenotazione_volo_programmato
  BEFORE INSERT ON Prenotazione
  FOR EACH ROW
  EXECUTE FUNCTION trg_check_prenotazione_stato();
```

4.2.2 Gestione documenti passeggero

Non si può cambiare il numero di documento (e quindi il relativo passeggero) se la prenotazione è cancellata o confermata.

```
CREATE OR REPLACE FUNCTION trg_block_documento_change()
  RETURNS TRIGGER AS $$
BEGIN
  IF OLD.Documento <> NEW.Documento THEN
    IF OLD.Stato IN ('CONFERMATA','CANCELLATA') THEN
      RAISE EXCEPTION
        'Non    possibile modificare il documento della
           prenotazione %: stato = %',
        OLD.Codice_biglietto, OLD.Stato;
    END IF;
  END IF;

  RETURN NEW;
END;
$$  LANGUAGE plpgsql;

CREATE TRIGGER trg_no_change_documento_when_finalized
  BEFORE UPDATE OF Documento ON Prenotazione
  FOR EACH ROW
  EXECUTE FUNCTION trg_block_documento_change();
```

4.2.3 Gestione stato prenotazione

Non si può modificare lo stato di una prenotazione se il volo a cui è legata è stato cancellato.

```
CREATE OR REPLACE FUNCTION
   trg_block_prenotazione_if_flight_cancelled()
  RETURNS trigger AS $$
DECLARE
  v_stato_volo StatoVolo;
BEGIN
  SELECT Stato
    INTO v_stato_volo
    FROM Volo
   WHERE Codice_volo = NEW.Codice_volo;

  IF v_stato_volo = 'CANCELLATO'
```

```
12      AND NEW.Stato <> OLD.stato THEN
13     RAISE EXCEPTION
14       'Impossibile modificare lo stato della prenotazione %: il
            volo %    cancellato',
15       NEW.Codice_biglietto, NEW.Codice_volo;
16   END IF;
17   RETURN NEW;
18 END;
19 $$ LANGUAGE plpgsql;
20
21
22 CREATE TRIGGER chk_no_status_change_if_flight_cancelled
23   BEFORE UPDATE OF Stato ON Prenotazione
24   FOR EACH ROW
25   EXECUTE FUNCTION trg_block_prenotazione_if_flight_cancelled();
```

4.2.4 Gestione stato bagaglio caricato

Un bagaglio può essere caricato solamente se la prenotazione è stata confermata.

```
1
2 CREATE OR REPLACE FUNCTION
     trg_check_bagaglio_prenotazione_confermata()
3   RETURNS trigger AS $$
4 DECLARE
5   stat_prenotazione  StatoPrenotazione;
6 BEGIN
7   SELECT Stato
8     INTO stat_prenotazione
9     FROM Prenotazione
10    WHERE Codice_biglietto = NEW.Codice_biglietto;
11
12   -- Se la prenotazione non    confermata, blocca l'operazione
13   IF v_pren_state <> 'CONFERMATA' THEN
14     RAISE EXCEPTION
15       'Impossibile caricare bagaglio: la prenotazione %    nello
            stato %',
16       NEW.Codice_biglietto, stat_prenotazione;
17   END IF;
18
19   RETURN NEW;
20 END;
```

```
$$ LANGUAGE plpgsql;

CREATE TRIGGER chk_bagaglio_only_if_pren_confirmed
  BEFORE INSERT ON Bagaglio
  FOR EACH ROW
  EXECUTE FUNCTION trg_check_bagaglio_prenotazione_confermata();
```

4.2.5 Gestione stato bagaglio ritirato

Un bagaglio può essere ritirato solamente se il volo è già atterrato.

```

CREATE OR REPLACE FUNCTION
   trg_validate_bagaglio_ritiro_after_landing()
  RETURNS TRIGGER

AS $$
DECLARE
  v_codice_volo    VARCHAR(6);
  v_stato_volo     StatoVolo;
BEGIN
  IF NEW.stato <> OLD.stato AND NEW.stato = 'RITIRATO' THEN

    SELECT p.Codice_volo, v.Stato
      INTO v_codice_volo, v_stato_volo
      FROM Prenotazione p
      JOIN Volo v ON p.Vodice_volo = v.Codice_volo
     WHERE p.Codice_biglietto = NEW.Codice_biglietto;

    IF v_stato_volo <>'ATTERRATO' THEN
      RAISE EXCEPTION
        'Impossibile ritirare il bagaglio %: il volo %    nello
           stato %',
        NEW.Codice_bagaglio, v_codice_volo, v_stato_volo;
    END IF;
  END IF;

  RETURN NEW;
END;
$$   LANGUAGE plpgsql;

CREATE TRIGGER chk_bagaglio_ritiro_volo_atterrato
```

```
30  BEFORE UPDATE OF Stato ON Bagaglio
31  FOR EACH ROW
32  EXECUTE FUNCTION trg_validate_bagaglio_ritiro_after_landing();
```

Capitolo 5

Funzioni, procedure e altre automazioni

Di seguito sono riportate le *stored functions* e le *stored procedures* implementate con l'obiettivo di semplificare e automatizzare alcune operazioni ricorrenti all'interno della base di dati, migliorando così l'efficienza e la manutenibilità del sistema.

5.1 Inserimento Volo (Amministratore)

Questa function permette di inserire un volo controllando che a farlo sia un amministratore come indicato nei requisiti di sistema.

```
CREATE OR REPLACE FUNCTION insert_volo_if_admin(
  p_username        VARCHAR(16),
  p_codice_volo     VARCHAR(6),
  p_compagnia       VARCHAR(30),
  p_data_volo       DATE,
  p_ora_volo        TIME,
  p_posti_totali    INTEGER,
  p_origine         VARCHAR(80),
  p_destinazione    VARCHAR(80),
  p_stato           StatoVolo,
  p_numero_gate     INTEGER    DEFAULT NULL,
  p_ritardo         INTEGER    DEFAULT 0
)
RETURNS VOID

AS $$
DECLARE
  v_ruolo Ruolo;
```

```
19 BEGIN
20   SELECT ruolo INTO v_ruolo FROM utente WHERE username =
        p_username;
21
22   IF NOT FOUND OR v_ruolo <> 'AMMINISTRATORE' THEN
23     RAISE EXCEPTION 'Utente "%" non autorizzato ad inserire voli'
          , p_username;
24   END IF;
25
26   INSERT INTO volo (
27     codice_volo,
28     compagnia,
29     data,
30     ora,
31     posti_totali,
32     posti_occupati,
33     origine,
34     destinazione,
35     stato,
36     numero_gate,
37     ritardo
38   ) VALUES (
39     p_codice_volo,
40     p_compagnia,
41     p_data_volo,
42     p_ora_volo,
43     p_posti_totali,
44     0,
45     p_origine,
46     p_destinazione,
47     p_stato,
48     p_numero_gate,
49     p_ritardo
50   );
51 END;
52 $$ LANGUAGE plpgsql;
```

5.2 Segnalazione bagaglio smarrito(Utente)

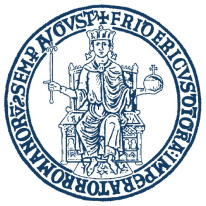
Questa funzione permette agli utenti di inserire lo smarrimento del bagaglio.

```
CREATE OR REPLACE FUNCTION segnala_bagaglio_smarrito(
  p_codice_bagaglio INTEGER,
  p_username        VARCHAR(16)
)
RETURNS VOID AS $$
DECLARE
  v_username_owner VARCHAR(16);
BEGIN
  SELECT pr.username
    INTO v_username_owner
    FROM bagaglio b
    JOIN prenotazione pr ON b.codice_biglietto = pr.
       codice_biglietto
   WHERE b.codice_bagaglio = p_codice_bagaglio;
  IF NOT FOUND THEN
    RAISE EXCEPTION 'Bagaglio con codice % inesistente',
       p_codice_bagaglio;
  END IF;

  IF v_username_owner <> p_username THEN
    RAISE EXCEPTION
      'Operazione non autorizzata: il bagaglio % non appartiene
         all''utente %',
      p_codice_bagaglio, p_username;
  END IF;

  UPDATE bagaglio
     SET stato = 'SMARRITO'
   WHERE codice_bagaglio = p_codice_bagaglio;
END;
$$ LANGUAGE plpgsql;
```

5.3 Ricerca Volo tramite destinazione, data, numero_posti(Tutti)

Questa funzione permette agli utenti della base di dati di effettuare una ricerca dei in partenza

```
CREATE OR REPLACE FUNCTION ricerca_voli_disponibili(
  p_destinazione     VARCHAR,
  p_data_volo        DATE,
  p_num_passeggeri   INTEGER
)
RETURNS TABLE (
  codice_volo        VARCHAR(6),
  compagnia          VARCHAR(30),
  data               DATE,
  ora                TIME,
  origine            VARCHAR(80),
  destinazione       VARCHAR(80),
  stato              StatoVolo,
  numero_gate        INTEGER,
  ritardo            INTEGER,
  posti_disponibili  INTEGER
) AS $$
  BEGIN
  RETURN QUERY
    SELECT
      v.codice_volo,
      v.compagnia,
      v.data,
      v.ora,
      v.origine,
      v.destinazione,
      v.stato,
      v.numero_gate,
      v.ritardo,
      (v.posti_totali - v.posti_occupati) AS posti_disponibili
    FROM volo v
    WHERE v.destinazione    = p_destinazione
      AND v.data            = p_data_volo
      AND v.stato           = 'PROGRAMMATO'
      AND (v.posti_totali - v.posti_occupati) >= p_num_passeggeri
    ORDER BY v.ora;
END;
$$ LANGUAGE plpgsql;
```



UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

Documentazione Progetto Object Orientation

Carmine Sgariglia (N86005069)
Mattia Lemma (N86005170)
Massimo Russo (N86005016)

Anno accademico 2024/2025

Grazie per la cortese attenzione