UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

RELAZIONE CONCLUSIVA DI
BASI DI DATI I

Base di dati per la gestione dell'aeroporto di Napoli

Author:
Giandomenico Iameo
N86002856

Supervisor:
Prof. Silvio Barra

Un elaborato redatto nel rispetto dei requisiti previsti per l'esame di Basi di dati I

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Corso di Laurea Triennale in Informatica

Anno Accademico 2025/2026

Indice

Capitolo 1

Progettazione concettuale

1.1 Introduzione

Al fine di garantire la massima coerenza con il processo di progettazione di una base di dati, si è deciso di strutturare i capitoli seguendo le diverse fasi metodologiche.

Si è partiti dall'analisi generale dei requisiti, sono stati estrapolati i concetti fondamentali dall'enunciato e si è proseguito attraverso successivi raffinamenti di questi elementi fino alla definizione del primo schema concettuale. Dopodiché si è proceduto alla sua ristrutturazione e, successivamente, è stato effettuato il passaggio (mapping) dallo schema concettuale allo schema relazionale. Infine, il lavoro si è concluso con la trasformazione nello schema fisico dei dati.

1.2 Enunciato del problema

Si sviluppi un sistema informativo per la gestione dell'aeroporto di Napoli, composto da una base di dati relazionale e da un'applicativo Java con interfaccia grafica realizzata con Swing. Questo sistema deve consentire di organizzare e monitorare le operazioni aeroportuali in modo efficiente e intuitivo.

Il sistema può essere utilizzato da utenti autenticati tramite uno *username* e una *password*. Gli utenti sono suddivisi in due ruoli: *utenti generici*, che possono prenotare voli, e *amministratori del sistema*, che gestiscono l'inserimento e l'aggiornamento dei voli.

Il sistema gestisce i voli in *arrivo* e in *partenza*. Ogni volo è caratterizzato da un codice univoco, la compagnia aerea, l'aeroporto di origine (per i voli in arrivo a Napoli) e quello di destinazione (per i voli in partenza da Napoli), la data del volo, l'orario previsto e lo stato del volo (programmato, decollato, in ritardo, atterrato, cancellato). Gli amministratori di sistema hanno la possibilità di inserire nuovi voli e aggiornare le informazioni sui voli esistenti.

Gli utenti generici possono effettuare prenotazioni per i voli *programmati*. Ogni prenotazione è legata a un volo e contiene informazioni come i dati del passeggero (che non deve necessariamente coincidere con il nome dell'utente che lo ha prenotato), il numero del biglietto, il posto assegnato e lo stato della prenotazione (confermata, in attesa, cancellata). Gli utenti possono cercare e modificare le proprie prenotazioni in base al nome del passeggero e al numero del volo.

Il sistema gestisce anche i gate di imbarco (identificati da un numero), assegnandoli ai voli in partenza. Gli amministratori possono modificare l'assegnazione dei gate.

Il sistema consente agli utenti di visualizzare aggiornamenti sui voli prenotati accedendo alla propria area personale, dove possono controllare eventuali ritardi, cancellazioni o variazioni direttamente dall'interfaccia. Inoltre, all'interno della *homepage* degli utenti viene mostrata una tabella con gli orari aggiornati con i voli in partenza e in arrivo, fornendo una panoramica immediata delle operazioni aeroportuali.

Infine il sistema permette di eseguire ricerche rapide per trovare voli, passeggeri e bagagli in base a diversi criteri. Le informazioni più importanti vengono evidenziate, come i voli in ritardo o cancellati., per facilitare la gestione delle operazioni aeroportuali.

1.3 Individuazione dei tipi di entità

Dall'analisi dell'enunciato sono stati identificati i seguenti tipi di entità principali:

Generico
Evidenzia il tipo di entità che rappresenta i singoli utenti generici abilitati a creare o modificare le prenotazioni per i voli programmati.

Amministratore
Evidenzia il tipo di entità che rappresenta gli amministratori del sistema che hanno la responsabilità di mantenere aggiornata la base di dati, inserendo e modificando informazioni sui voli, nonché gestendo l'assegnamento ai rispettivi gate di imbarco.

Prenotazione
Evidenzia il tipo di entità che rappresenta l'insieme delle prenotazioni effettuate dagli utenti generici per un volo registrato nella base di dati.

VoloPartenza
Evidenzia il tipo di entità che rappresenta i voli in partenza dall'aeroporto di Napoli.

VoloArrivo
Evidenzia il tipo di entità che rappresenta i voli in arrivo all'aeroporto di Napoli.

Gate
Evidenzia il tipo di entità che rappresenta l'insieme dei gate abilitati per le operazioni di imbarco e sbarco.

1.4 Raffinamento dei concetti

Da un'analisi più accurata sono emersi successivi miglioramenti ai concetti definiti precedentemente. In questo paragrafo sono stati riportati tali aggiornamenti, procedendo all'integrazione di nuovi costrutti.

1.4.1 Generalizzazioni

Qui sono state elencate le generalizzazioni individuate e le relative specializzazioni.

Utente

Secondo la traccia, ogni utente (*Generico* o *Amministratore*) deve autenticarsi tramite username e password. Pertanto, poiché questi utenti condividono le medesime caratteristiche, è stato introdotto il tipo di entità *Utente* come generalizzazione, considerando l'insieme {*Generico*, *Amministratore*} come una sua specializzazione.

A questa specializzazione sono stati poi applicati due vincoli: il *vincolo di disgiunzione* e il *vincolo di completezza totale*, poiché ogni utente è esclusivamente o un utente generico oppure un utente amministratore.

Volo

Analogamente a quanto accaduto per i tipi di entità *Generico* e *Amministratore*, anche *VoloPartenza* e *VoloArrivo* sono stati generalizzati. La *superclasse generalizzata* è stata chiamata *Volo*.

La specializzazione {*VoloPartenza*, *VoloArrivo*} è di tipo *disgiunta*, poiché un volo non può essere contemporaneamente in arrivo e in partenza. Inoltre, è *totale*, in quanto ogni volo deve necessariamente essere classificato in almeno una delle due categorie (non esistono voli non assegnati, come un "Volo di Collaudo" o un "Volo di Trasferimento Tecnico" senza passeggeri).

1.5 Individuazione degli attributi

Per ogni tipo di entità, rispettivamente, sono state identificate diverse caratteristiche distintive, riportate qui di seguito in ordine.

Attributo	**Breve descrizione**
IdUtente	L'adozione di questo attributo, pur non prevista dalle specifiche originali, è motivata dalla necessità di ottimizzare le performance della base di dati rispetto all'uso di chiavi naturali, come verrà illustrato in seguito.
Username	Campo in cui è presente il nome utente associato all'account.
Password	Campo in cui è presente la password dell'utente per accedere all'account.

Tabella 1.1: Tipo di entità *Utente*

Attributo	**Breve descrizione**
IdPrenotazione	Ogni prenotazione ha un proprio codice identificativo.
PostoAssegnato	Indica il posto a sedere assegnato al passeggero per il volo selezionato.
StatoPrenotazione	Rappresenta lo stato corrente della prenotazione; l'insieme dei valori possibili definisce l'intero ciclo di vita della prenotazione stessa.
NumeroBagagli	Attributo che tiene traccia del numero di bagagli del passeggero.
NumeroBiglietto	Specifica il numero di biglietto assegnato a un passeggero.
Passeggero	Dal momento che nella traccia non è stato specificato il numero di passeggeri da associare a una prenotazione ho supposto che per ogni prenotazione sia associato un passeggero.

Tabella 1.2: Tipo di entità *Prenotazione*

Attributo	**Breve descrizione**
Codice	Ogni volo è caratterizzato da un codice identificativo univoco.
CompagniaAerea	Associa il volo alla compagnia aerea di appartenenza.
StatoVolo	Rappresenta lo stato corrente di un volo.
Prenotabile	Attributo booleano introdotto per semplificare la verifica della disponibilità. Segnala se il volo è ancora aperto alle prenotazioni o se la capienza dell'aereo è stata raggiunta.
Orario	Definisce un attributo composto che tiene traccia dell'orario di arrivo e partenza di ogni volo.

Tabella 1.3: Tipo di entità *Volo*

Attributo	**Breve descrizione**
AeroportoOrigine	In questo campo sarà presente il nome dell'aeroporto di partenza di un volo.

Tabella 1.4: Tipo di entità *VoloPartenza*

Attributo	**Breve descrizione**
AeroportoDestinazione	Nome dell'aeroporto di arrivo del volo.

Tabella 1.5: Tipo di entità *VoloArrivo*

Attributo	**Breve descrizione**
Numero	Come specificato nella traccia, ogni gate è identificato da un numero; pertanto, è stato imposto il vincolo di univocità su tale attributo.

Tabella 1.6: Tipo di entità *Gate*

Per quanto riguarda i tipi di entità *Generico* e *Amministratore* essi non hanno alcun attributo in più rispetto a quelli che derivano dalla generalizzazione *Utente*.

1.6 Individuazione dei tipi di associazione

In questo paragrafo sono stati descritti i tipi di associazioni individuati tra i tipi di entità. Al fine di chiarire le decisioni progettuali, per ciascun tipo di associazione sono stati analizzati il rapporto cardinalità e i vincoli di partecipazione.

1.6.1 Effettuazione

Il tipo di associazione ha lo scopo di correlare i tipi di entità *Generico* e *Prenotazione*, rappresentando logicamente l'atto di un utente che effettua una prenotazione di un volo.

1. (*Scelta del rapporto di cardinalità*): Per il tipo di associazione Generico:Prenotazione è stato definito un rapporto di cardinalità 1:N. Come indicato nella traccia, un utente può effettuare più prenotazioni. Viceversa, per garantire l'univocità della titolarità, è stato stabilito che ogni singola prenotazione debba riferirsi esclusivamente a un unico utente.
2. (*Scelta dei vincoli di partecipazione*):
    - *Generico*: Un'entità generico può registrarsi al sistema ed esistere nella base di dati anche prima di aver effettuato qualsiasi prenotazione. Per questo motivo, è stato scelto il vincolo di partecipazione parziale del tipo di entità *Generico* al tipo di associazione *Effettuazione*.
    - *Prenotazione*: Un'occorrenza di *Prenotazione* non può esistere se non è stata creata da nessun utente generico. A queto scopo, è stato scelto il vincolo di partecipazione totale (applicato a *Prenotazione*) al tipo di associazione.

1.6.2 Inclusione

Associa il tipo di entità *Prenotazione* al tipo di entità *Volo*, indicando a quale volo specifico si riferisce la prenotazione confermata.

1. (*Scelta del rapporto di cardinalità*): Per la coppia Volo:Prenotazione è stato scelto un rapporto di cardinalità 1:N. Come specificato nella traccia, una prenotazione deve essere legata a un unico volo. Viceversa, ogni volo può includere diverse prenotazioni effettuate dagli utenti.
2. (*Scelta dei vincoli di partecipazione*):
    - *Volo*: Un'entità volo può essere memorizzata anche se non è stata creata alcuna prenotazione. Un volo deve esistere prima di essere prenotato. Per tale

motivo, è stato stabilito che la partecipazione del tipo di entità *Volo* al tipo di associazione *Inclusione* sia parziale.

- *Prenotazione*: Ogni prenotazione deve necessariamente riferirsi a un volo per poter essere effettuata da un utente. Ragion per cui, è stato imposto il vincolo di partecipazione totale del tipo di entità *Prenotazione* rispetto al tipo di associazione.

1.6.3 Gestione

Unisce i tipi di entità *Amministratore* e *Volo*, denotando la responsabilità di un amministratore di coordinare i voli a esso assegnati.

1. (*Scelta del rapporto di cardinalità*): Per il tipo di associazione Amministratore:Volo è stato definito un rapporto di cardinalità 1:N. Sebbene un amministratore possa gestire molteplici voli, come specificato nell'enunciato del problema, è stato stabilito che ogni singolo volo sia assegnato a un unico responsabile. Questa scelta mira a evitare conflitti di gestione e a garantire una chiara attribuzione delle responsabilità.
2. (*Scelta dei vincoli di partecipazione*):
    - *Amministatore*: Un amministratore può esistere nella base di dati anche senza incarichi attivi. Pertanto, è stato scelto scelto il vincolo di partecipazione parziale al tipo di associazione.
    - *Volo*: Un'occorrenza di *Volo* deve essere necessariamente associata a un amministratore. In assenza di un responsabile, non sarebbe possibile gestire le operazioni critiche del volo, come l'aggiornamento degli orari o la gestione delle emergenze. Per questo motivo, la partecipazione del tipo di entità *Volo* al tipo di associazione *Gestione* è totale.

1.6.4 Assegnazione

Collega i tipi di entità *Volo* e *Gate*, definendo l'allocazione del volo presso lo specifico punto di imbarco o sbarco.

1. (*Scelta del rapporto di cardinalità*): Per la coppia Gate:Volo è stato scelto un rapporto di cardinalità 1:N (uno a molti). Ho ipotizzato che i gate gestiscano sia flussi di arrivo che di partenza. Pertanto, un singolo gate può ospitare più voli (in fasce orarie diverse), mentre ogni specifico volo è associato univocamente a un solo gate.
    - *Gate*: Un'entità gate ha un'esistenza indipendente dal volo. Un gate deve essere censito nel sistema a prescindere dal fatto che vi siano o meno voli assegnati in quel momento. Dunque, è stato scelto scelto il vincolo di partecipazione parziale.

- *Volo*: In genere, un volo è pianificato mesi prima. L'assegnazione del gate non è quindi un requisito necessario per la creazione dell'istanza di volo. Per questo motivo, è stato deciso che la partecipazione del tipo di entità *Volo* al tipo di associazione sia parziale.

Un gate può ospitare più voli, a condizione che gli intervalli di occupazione non si sovrappongano. Per garantire il rispetto di tale vincolo temporale, sono stati aggiunti al tipo di associazione *Assegnazione* gli attributi *OrarioInizioAssegnazione* e *OrarioFineAssegnazione*. Inoltre, l'attributo *Stato* definisce la fase operativa dell'assegnazione, facilitando il controllo sulla disponibilità della risorsa.

1.7 Schema ER

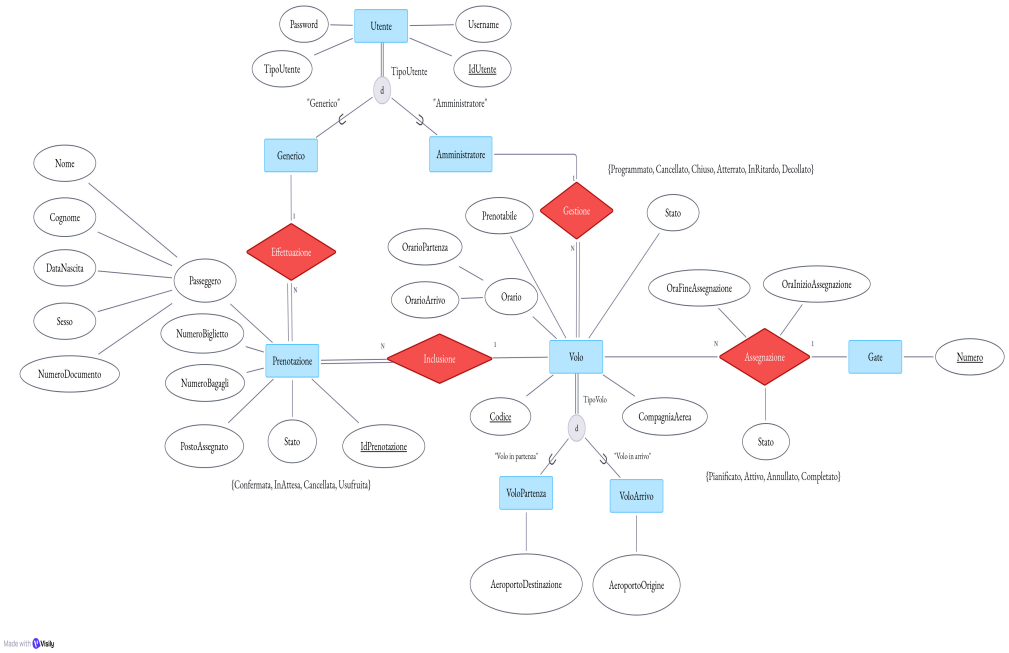

Figura 1.1: Diagramma ER per la base di dati

Capitolo 2

Ristrutturazione concettuale

2.1 Ristrutturazione dello schema ER

Nei successivi due capitoli si è affrontata la progettazione logica, procedendo alla traduzione dello schema ER, studiato e realizzato nel capitolo 1, nel corrispondente schema logico.

Con l'obiettivo di costruire uno schema logico capace di descrivere, in maniera *efficiente* e *corretta*, tutte le informazioni concettuali dello schema ER prodotto, è stato necessario suddividere il processo in due fasi.

1. Ristrutturazione dello schema ER;
2. Traduzione verso lo schema logico.

Nel presente capitolo è stato affrontato il primo punto, in cui sono state eseguite operazioni di semplificazione e ottimizzazione dello schema ER originale. La ristrutturazione ha comportato solo quattro interventi, che possono essere riassunti nei seguenti punti:

1. Analisi delle ridondanze;
2. Eliminazione delle generalizzazioni;
3. Eliminazione degli attributi composti;
4. Scelta degli identificatori primari.

2.1.1 Analisi delle ridondanze

Per quanto concerne le ridondanze, dallo schema riportato in Figura 1.1 non ne è emersa alcuna, almeno non nelle *forme* analizzate durante il corso. Tuttavia, a seguito di un esame più accurato della struttura, sono stati individuati dei concetti che, se espressi nella forma riportata in Figura 1.1, avrebbero portato inevitabilmente alla formazione di relazioni con molti dati ridondanti. Per evitare tale criticità, si è deciso quindi di rielaborare questi concetti.

Ristrutturazione dell'attributo *CompagniaAerea*

Dopo un'attenta analisi del diagramma ER, si è deciso di modellare *CompagniaAerea* come un tipo di entità separata. Questa decisione è stata motivata anche dal fatto che non era stato specificato nella traccia come memorizzare tale informazione.

È stato valutato che considerare la Compagnia aerea come un semplice attributo del tipo di entità *Volo* avrebbe introdotto inutili ridondanze. Infatti, come si notava dal diagramma, la compagnia aerea è funzione del solo *Volo*. Se fosse stata mantenuta come attributo, il nome della stessa compagnia sarebbe stato ripetuto in ogni tupla relativa ai voli da essa operati. Questo fatto avrebbe avuto impatti sul contenuto della relazione *Volo* e sulle operazioni effettuabili su di essa, comportando due conseguenze negative.

In primo luogo, se una compagnia avesse operato su n voli, il suo nome sarebbe stato duplicato n volte, occupando spazio inutilmente. In secondo luogo, se il nome della compagnia fosse variato, sarebbe stato necessario andarne a modificare il valore in tutte le tuple corrispondenti, affinché la dipendenza continuasse a valere.

Ristrutturazione dell'attributo *Aeroporto*

Analogamente a quanto osservato per la compagnia aerea, è stato rilevato che anche trattare l'aeroporto come un semplice attributo avrebbe comportato significative ridondanze.

Supponendo ad esempio che lo scalo di Napoli Capodichino avesse gestito 500 voli al giorno, sarebbe stato necessario ripetere la stringa "Napoli Capodichino" 500 volte all'interno della tabella. Inoltre, se il nome ufficiale dell'aeroporto fosse cambiato (o se si fosse voluto semplicemente correggere un refuso, come ad esempio "Napole" invece di "Napoli"), sarebbe stato indispensabile ricercare e aggiornare tutte le 500 tuple coinvolte, rischiando incongruenze nei dati.

Ristrutturazione degli attributi *VoloInPartenza* e *VoloInArrivo*

L'eliminazione della generalizzazione *Volo*, avrebbe inevitabilmente portanto all'introduzione di molte ridondanze strutturali. Per questo motivo, si è optato per una modellazione diversa, definendo *Aeroporto* come un tipo di entità autonomo collegato al tipo di entità *Volo*.

Un singolo aeroporto può fungere da punto di origine o di destinazione per una moltitudine di voli, sia in entrata che in uscita. Pertanto, quelle che inizialmente erano state concepite come sottoclassi (*VoloInPartenza* e *VoloInArrivo*) sono state trasformate in due distinte associazioni che collegano l'entità *Volo* all'entità *Aeroporto*. In questo modo, si è fatto si che ogni volo riferisse semplicemente l'identificativo dell'aeroporto di partenza e di quello di arrivo, eliminando ogni ridondanza.

2.1.2 Eliminazione delle generalizzazioni

In questo paragrafo si è procedeuto all'eliminazione dell'unica generalizzazione presente nello schema, ovvero quella chiamata *Utente*, analizzando i vantaggi della soluzione adottata rispetto ai possibili metodi di risoluzione esistenti:

1. Accorpamento delle figlie della generalizzazione nel genitore;
2. Accorpamento del genitore della generalizzazione nelle figlie;
3. Sostituzione della generalizzazione con associazioni.

Nel caso preso in esame, si è scelto di adottare la seconda soluzione. Infatti, il tipo di entità *Utente* è stato eliminato e, per la proprietà dell'ereditarietà, i suoi attributi e il suo identificatore sono stati aggiunti ai tipi di entità figlie.

Vantaggi del secondo approccio ripetto al terzo

Uno dei vantaggi di questo approccio rispetto alla *Sostituzione della generalizzazione con associazioni* è la *riduzione degli accessi*, trascurando il numero di attributi del tipo di entità *Generico* (o *Amministratore*) dopo la ristrutturazione.

Supponendo, infatti, che un utente *Generico* effettua in media m prenotazioni. Allora, con il terzo approccio, il DBMS per accedere (in lettura) alle prenotazioni di un particolare utente dovrebbe accedere a un'occorrenza del tipo di entità *Utente* poi a un'occorrenza dell'associazione che lega *Utente* e *Generico* e, attraverso questa, a un'occorrenza del tipo di entità *Generico*.

Successivamente, dovrebbe accedere a m occorrenze del tipo di associazione *Effettuazione* e, attraverso queste, a m occorrenze del tipo di entità *Prenotazione*. Per un totale di $3 + 2m$ accessi.

Sembra chiaro che con il primo approccio il numero degli accessi è decisamente minore ($2m$). Gli accessi in più potrebbero avere un impatto significativo su larga scala, cioè quando le query aumentano e diventano estremamente complesse.

Nonostante che l'approccio *Sostituzione della generalizzazione con associazioni* ha come vantaggio teorico quello di creare tabelle figlie sparse (con pochi attributi), in questo caso il tipo di entità *Generico* è naturalmente sparso fin dall'inizio. Pertanto, il principale vantaggio di quest'ultimo approccio scompare e il costo in termini di prestazioni in lettura diventa il fattore dominante.

Vantaggi del secondo approccio rispetto al primo

Si ha un risparmio di memoria perché non è necessario riferirsi a occorrenze di *Utente* che non sono occorrenze né di *Generico* e né di *Amministratore*, quindi non è necessario aggiungere un attributo che serve a distinguere il "tipo" di occorrenza di *Utente*.

2.1.3 Eliminazione degli attributi composti

Risoluzione dell'attributo *Passeggero*

La scelta di modellare *Passeggero* come entità autonoma, anziché come attributo, è nata dall'esigenza di supportare scenari futuri più complessi, come la possibilità di associare più passeggeri a una singola prenotazione.

Tale operazione ha alleggerito la struttura del tipo di entità *Prenotazione*, riducendone il numero di attributi. Tuttavia, si è tenuto conto che questa separazione avrebbe comportato un leggero costo prestazionale: per ricostruire l'informazione completa (prenotazione unita ai dettagli del passeggero) sarebbe stato necessario un maggior numero di accessi ai dati.

Sebbene non fosse espressamente specificato nella traccia, si è deciso di adottare un modello orientato alla privacy degli utenti. A tale scopo, è stato stabilito che i dati personali di ogni *Passeggero* venissero eliminati contestualmente alla cancellazione della *Prenotazione*. Tuttavia, poiché le entità sono collegate, per evitare la proliferazione di valori nulli nella base di dati, è stato introdotto il tipo di entità *StoricoPrenotazione*. Questa soluzione ha permesso di conservare i dati statistici delle prenotazioni effettuate senza trattenere i dati sensibili dei passeggeri.

2.1.4 Scelta degli identificatori primari

Tipi di entità *Generico* e *Amministratore*

Per questi tipi di entità sono stati scelti gli attributi *IdGenerico* e *IdAmministratore* come identificatori primari in virtù del loro dominio numerico. L'attributo *Username*, infatti, presenta dati non sequenziali e di lunghezza variabile, caratteristiche che renderebbero le operazioni di confronto e indicizzazione meno efficienti rispetto all'uso di un identificatore intero.

Inoltre, è stato privilegiato un identificatore immutabile: un *Username* può variare nel tempo, e ciò richiederebbe di propagare l'aggiornamento a cascata in tutte le tabelle in cui è referenziato.

Tipo di entità *Prenotazione*

Per questo tipo di entità è stato scelto l'attributo *IdPrenotazione* in virtù dell'immutabilità rispetto a *NumeroBiglietto*. Per tale motivo, si è ritenuto inappropriato selezionare quest'ultimo come chiave primaria.

Tipo di entità *CompagniaAerea*

Per questo tipo di entità è stato selezionato *IdCompagnia* come chiave primaria per via del suo dominio intero. Come è accaduto per *Generico* e *Amministratore*, l'uso di un identificatore numerico, a differenza del *Nome* della compagnia, garantisce la sequenzialità dei valori e riduce drasticamente i tempi di confronto ed elaborazione da parte della base di dati.

Tipo di entità *Aeroporto*

Per tale tipo di entità è stato scelto l'attributo *Codice* come identificatore primario. La scelta è stata dettata dal fatto che un identificatore costituito da pochi attributi è da preferire a identificatori costituiti da molti attributi. Questo permette un risparmio di memoria nella realizzazione dei legami logici tra le varie relazioni e facilita le operazioni di join.

2.2 Schema ER ristrutturato

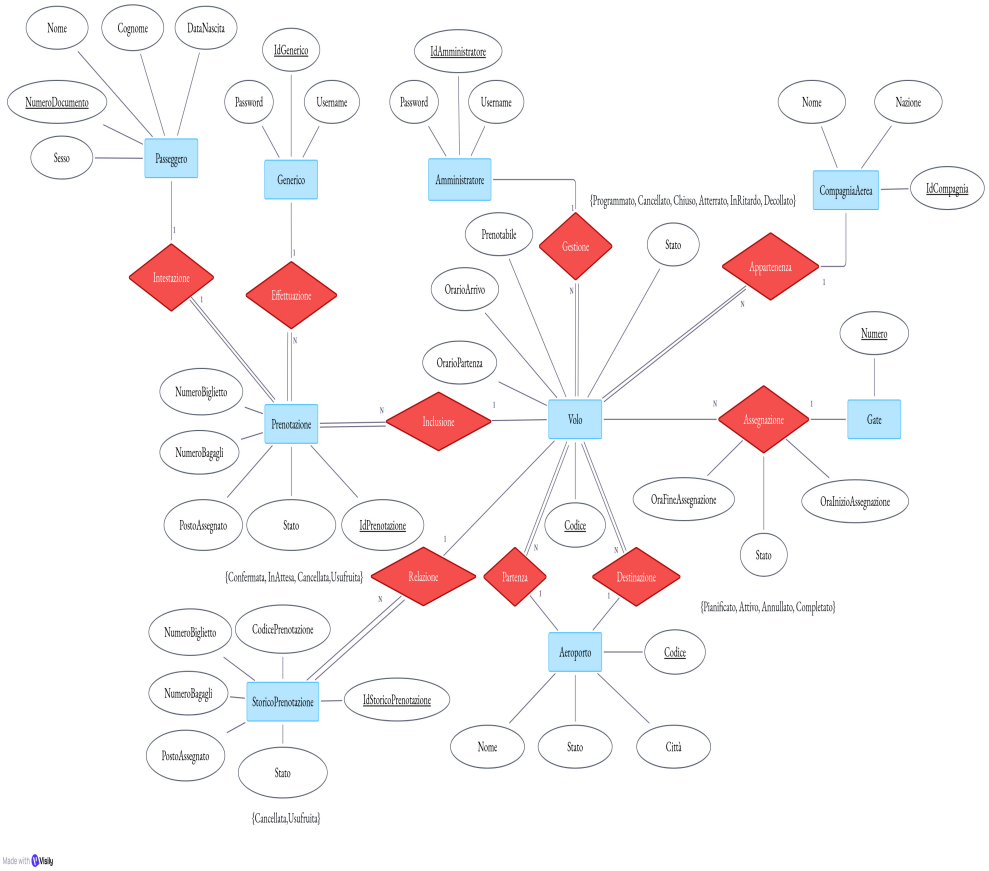

Figura 2.1: Diagramma ER ristrutturato

2.3 Dizionario dei tipi di entità

Generico

Descrizione: Utente del sistema che effettua prenotazioni.

Attributi:

- *IdGenerico* (**PK**): identificativo univoco dell'utente.
  - *Dominio*: Integer.
- *Username*: nome associato all'account utente.
  - *Dominio*: Varchar[30].
- *Password*: password associata all'account utente.
  - *Dominio*: Varchar[30].

Amministratore

Descrizione: Utente del sistema che gestisce i voli.

Attributi:

- *IdAmministratore* (**PK**): identificativo univoco dell'utente.
  - *Dominio*: Integer.
- *Username*: nome associato all'account utente.
  - *Dominio*: Varchar[30].
- *Password*: password associata all'account utente.
  - *Dominio*: Varchar[30].

Prenotazione

Descrizione: Rappresenta un contratto stipulato da un utente generico tramite il sistema.

Attributi:

- *IdPrenotazione* (**PK**) : Codice identificativo della prenotazione.
  - *Dominio*: Serial.
- *Stato*: Stato corrente della prenotazione.
  - *Dominio*: $\{Confermata, InAttesa, Cancellata, Usufruita\}$.
- *Prenotabile*: Flag che indica se è possibile effettuare nuove prenotazioni per il volo.
  - *Dominio*: Boolean.
- *PostoAssegnato*: Numero di posto a sedere assegnato al passeggero.
  - *Dominio*: Integer.
- *NumeroBagagli*: Numero di bagagli del passeggero.
  - *Dominio*: Integer.
- *NumeroBiglietto*: Numero biglietto assegnato al passeggero.
  - *Dominio*: Integer.

Passeggero

Descrizione: Persona fisica identificata come titolare del titolo di viaggio.

Attributi:

- *NumeroDocumento* (**PK**): Numero identificativo del passeggero.
  - *Dominio*: Varchar[20].
- *Nome*: Nome del passeggero.
  - *Dominio*: Varchar[50].
- *Cognome*: Cognome del passeggero.
  - *Dominio*: Varchar[50].
- *DataNascita*: Data di nascita del passeggero.
  - *Dominio*: Date.
- *Sesso*: Sesso del passeggero.
  - *Dominio*: $\{M, F, X\}$.

StoricoPrenotazione

Descrizione: Tiene traccia di tutte le prenotazioni effettuate, prive dei dati anagrafici del passeggero, che verranno utilizzate per analisi statistiche.

Attributi:

- *IdStoricoPrenotazione* (**PK**): Identificativo univoco StoricoPrenotazione
  - *Dominio*: Serial.
- *CodicePrenotazione*: Identificativo univoco della prenotazione effettuata.
  - *Dominio*: Integer.
- *Stato*: Stato corrente della prenotazione.
  - *Dominio*: $\{Cancellata, Usufruita\}$.
- *PostoAssegnato*: Numero di posto a sedere assegnato al passeggero.
  - *Dominio*: Integer.
- *NumeroBagagli*: Numero di bagagli del passeggero.
  - *Dominio*: Integer.
- *NumeroBiglietto*: Numero biglietto assegnato al passeggero.
  - *Dominio*: Integer.

Volo

Descrizione: Raccoglie le informazioni di ogni volo inserito nel sistema.

Attributi:

- *Codice* (**PK**): Codice univoco del volo.
  - *Dominio*: Serial.
- *Stato*: Stato corrente del volo.
  - *Dominio*: $\{Programmato, InRitardo, Chiuso, Atterrato, Cancellato\}$
- *OrarioPartenza*: Orario di partenza del volo.
  - *Dominio*: Timestamp with time zone.
- *OrarioArrivo*: Orario di arrivo del volo.
  - *Dominio*: Timestamp with time zone.

Aeroporto

Descrizione: Rappresenta l'infrastruttura aeroportuale che funge da punto di partenza o destinazione per i voli.

Attributi:

- *Codice* (**PK**): Codice univoco dell'aeroporto.
  - *Dominio*: Char(3).
- *Nome*: Nome dell'aeroporto.
  - *Dominio*: Varchar[100].
- *Stato*: Stato in cui è situato l'aeroporto.
  - *Dominio*: Varchar[50].
- *Citta*: Città in cui si trova l'aeroporto.
  - *Dominio*: Varchar[50].

CompagniaAerea

Descrizione: Rappresenta l'organizzazione o l'azienda che gestisce e opera i voli.

Attributi:

- *IdCompagnia* (**PK**): Codice univoco della comapgnia aerea.
  - *Dominio*: Integer.
- *Nome*: Nome della compagnia aerea.
  - *Dominio*: Varchar[50].
- *Nazione*: "Nazione in cui ha sede la compagnia aerea."
  - *Dominio*: Varchar[50].

Gate

Descrizione: Punto di imbarco o sbarco per i voli.

Attributi:

- *Numero* (**PK**): Numero univoco del gate.
  - *Dominio*: Integer.

2.4 Dizionario dei tipi di associazioni

Intestazione

Descrizione:

- Un passeggero intesta una prenotazione.
- Una prenotazione è intestata a un passeggero.

Classi coinvolte: *Passeggero:Prenotazione*

- Rapporto di cardinalità: 1:1
- Vincoli di partecipazione:
  - *Passeggero*: parziale
  - *Prenotazione*: totale

Effettuazione

Descrizione:

- Un utente generico effettua numerose prenotazione.
- Una prenotazione è effettuata da un utente generico.

Classi coinvolte: *Generico:Prenotazione*

- Rapporto di cardinalità: 1:N
- Vincoli di partecipazione:
  - *Generico*: parziale
  - *Prenotazione*: totale

Inclusione

Descrizione:

- Un volo include più prenotazioni.
- Una prenotazione appartiene a un volo.

Classi coinvolte: *Volo:Prenotazione*

- Rapporto di cardinalità: 1:N
- Vincoli di partecipazione:
  - *Volo*: parziale
  - *Prenotazione*: totale

Relazione

Descrizione:

- Un volo è referenziato da numerosi storicoPrenotazione.
- Uno storicoPrenotazione riguarda un volo.

Classi coinvolte: *Volo:StoricoPrenotazione*

- Rapporto di cardinalità: 1:N
- Vincoli di partecipazione:
  - *Volo*: parziale
  - *StoricoPrenotazione*: totale

Gestione

Descrizione:

- Un amministratore gestisce numerosi voli.
- Un volo è gestito da un amministratore.

Classi coinvolte: *Amministratore:Volo*

- Rapporto di cardinalità: 1:N
- Vincoli di partecipazione:
  - *Amministratore*: parziale
  - *Volo*: totale

Partenza

Descrizione:

- Un aeroporto è la partenza di numerosi voli.
- Un volo parte da un aeroporto.

Classi coinvolte: *Aeroporto:Volo*

- Rapporto di cardinalità: 1:N
- Vincoli di partecipazione:
  - *Aeroporto*: parziale
  - *Volo*: totale

Destinazione

Descrizione:

- Un aeroporto è la destinazione di numerosi voli.
- Un volo arriva a un aeroporto.

Classi coinvolte: *Aeroporto:Volo*

- Rapporto di cardinalità: 1:N
- Vincoli di partecipazione:
  - *Aeroporto*: parziale
  - *Volo*: totale

Appartenenza

Descrizione:

- Una compagniaAerea opera più voli.
- Un volo appartiene a una compagniaAerea.

Classi coinvolte: *CompagniaAerea:Volo*

- Rapporto di cardinalità: 1:N
- Vincoli di partecipazione:
  - *CompagniaAerea*: parziale
  - *Volo*: totale

Assegnazione

Descrizione:

- Un gate ospita numerosi voli.
- Un volo è assegnato a un gate.

Classi coinvolte: *Gate:Volo*

- Rapporto di cardinalità: 1:N
- Vincoli di partecipazione:
  - *Gate*: parziale
  - *Volo*: parziale

2.5 Dizionario dei vincoli

Nome vincolo	**Descrizione**
`AssegnazioneIdGenerico`	Assegna un codice univoco all'utente generico prelevando il valore dalla sequenza *IdUtente*.
`AssegnazioneIdAmministratore`	Assegna un codice univoco all'amministratore prelevando il valore dalla sequenza *IdUtente*.
`PartenzaArrivoNapoli`	Garantisce che ogni volo abbia Napoli come aeroporto di partenza o di destinazione.
`UnicoDocumentoPerVolo`	Garantisce che il medesimo numero di documento non possa essere associato più di una volta allo stesso volo.
`PuliziaOrfani`	Rimuove i dati del passeggero associati alla prenotazione quando quest'ultima viene invalidata.
`ArchiviazionePrenotazioni`	Archivia le informazioni sulla prenotazione dell'utente, omettendo i dettagli anagrafici del passeggero.
`VoloPrenotabile`	Verifica che il volo sia prenotabile.
`CancellazionePrenotazione`	Imposta lo stato della prenotazione su *Cancellata*. Se il volo è nello stato *Chiuso*, l'annullamento non è consentito.
`ControlloPostoUnicoPerVolo`	Garantisce l'unicità dei posti a sedere all'interno di ciascun volo.
`StatoVoloCancellato`	Imposta lo stato del volo su *Cancellato*, revocando l'assegnazione del gate e annullando le relative prenotazioni.

Nome vincolo	**Descrizione**
`StatoVoloAtterrato`	Imposta lo stato del volo su *Atterrato*, aggiorna le relative prenotazioni in *Usufruite* e segna lo stato di occupazione del gate come *Completato*.
`StatoVoloChiuso`	Imposta lo stato del volo su *Chiuso* e dichiara chiuse le prenotazioni, confermando quelle esistenti.
`StatoOccupazioneAttivo`	Consente la transizione dello stato di occupazione del gate in *Attivo* solo se lo stato precedente era *Pianificato*.
`AssegnazioneGate`	Assegna ai voli gli slot temporali per l'utilizzo del gate.
`OrarioConflittoAssegnazioneGate`	Verifica che gli slot temporali assegnati al medesimo gate non si sovrappongano.
`AggiornamentoAssegnazioneGate`	Fa scattare il vincolo *AssegnazioneGate* ogni qual volta un volo viene registrato nel sistema oppure modificato.

Capitolo 3

Progettazione logica

Dopo l'analisi svolta nei capitoli precedenti, è stato possibile passare alla traduzione nello schema relazionale. Questo procedimento ha richiesto molta attenzione: è stato infatti necessario sviluppare un processo di mappatura che tenesse conto dei seguenti punti:

- I tipi di entità;
- I tipi di associazioni tra i tipi di entità;
- I rapporti di cardinalità per i tipi di associazione.

3.1 Schema relazionale

Di seguito è riportato lo schema relazionale della base di dati. Nei risultati di mappatura sono stati specificati anche i vincoli del modello relazionale, fra i quali le chiavi primarie e i vincoli di integrità referenziale.

Generico(IdGenerico,Username,Password)

Amministratore(IdAmministratore,Username,Password)

Passeggero(NumeroDocumento,Nome,Cognome,DataNascita,Sesso)

Prenotazione(IdPrenotazione,Stato,PostoAssegnato,NumeroBagagli,NumeroBiglietto, IdGenerico,Codice,NumeroDocumento)
IdGenerico $\longrightarrow$ *Generico.IdGenerico*
Codice $\longrightarrow$ *Volo.Codice*
NumeroDocumento $\longrightarrow$ *Passeggero.NumeroDocumento*

StoricoPrenotazione(IdStoricoPrenotazione,CodicePrenotazione,PostoAssegnato,NumeroBagagli NumeroBiglietto,Codice)
Codice $\longrightarrow$ *Volo.Codice*

Volo(Codice,Stato,OrarioPartenza,OrarioArrivo,Prenotabile,IdAmministratore, IdCompagnia,CodiceAeroportoPartenza, CodiceAeroportoDestinazione)
IdAmministratore $\longrightarrow$ *Amministratore.IdAmministratore*

IdCompagnia $\longrightarrow$ *CompagniaAerea.IdCompagnia*
CodiceAeroportoPartenza $\longrightarrow$ *Aeroporto.Codice*
CodiceAeroportoDestinazione $\longrightarrow$ *Aeroporto.Codice*

Aeroporto(Codice,Nome,Stato,Citta)

CompagniaAerea(IdCompagnia,Nome,Nazione)

Gate(Numero)

Assegnazione(Codice,Numero,OrarioInizioAssegnazione,OrarioFineAssegnazione, StatoOccupazione)
Codice $\longrightarrow$ *Volo.Codice*
Numero $\longrightarrow$ *Gate.Numero*

Capitolo 4

Progettazione fisica

In questo capitolo si è affrontata l'implementazione delle tabelle nel linguaggio SQL, in particolare nello standard supportato dal DBMS relazionale commerciale PostgreSQL.

4.1 Definizione delle tabelle

Al fine di raggruppare logicamente le tabelle e i costrutti appartenenti alla medesima base di dati, è stato creato uno *schema SQL* mediante il seguente frammento di codice:

```
CREATE SCHEMA AEROPORTO AUTHORIZATION postgres;
```

Una volta predisposto lo schema, si è proceduto alla definizione delle singole tabelle. Di seguito sono riportate le relative istruzioni di creazione; per massimizzare la leggibilità del codice, la struttura è stata organizzata in due aree distinte: *definizione dei dati* e *definizione dei vincoli*.

4.1.1 Definizione della tabella *Generico*

```
CREATE TABLE AEROPORTO."Generico" (
    -- Definizione dei dati
    "IdGenerico" INTEGER     NOT NULL,
    "Username"   VARCHAR(30) NOT NULL,
    "Password"   VARCHAR(30) NOT NULL,

    -- Definizione dei vincoli basati sullo schema
    CONSTRAINT "GenericoChiavePrimaria"
        PRIMARY KEY ("IdGenerico"),
    CONSTRAINT "GenericoChiaveUnica"
        UNIQUE ("Username"),
    CONSTRAINT "LunghezzaPasswordGenerico"
        CHECK (LENGTH("Password") >= 6)
);
```

Listing 1: Tabella *Generico*

4.1.2 Definizione della tabella *Amministratore*

```
CREATE TABLE AEROPORTO."Amministratore" (
    -- Definizione dei dati
    "IdAmministratore" INTEGER     NOT NULL,
    "Username"         VARCHAR(30) NOT NULL,
    "Password"         VARCHAR(30) NOT NULL,

    -- Definizione dei vincoli basati sullo schema
    CONSTRAINT "AmministratoreChiavePrimaria"
        PRIMARY KEY ("IdAmministratore"),
    CONSTRAINT "AmministratoreChiaveUnica"
        UNIQUE ("Username"),
    CONSTRAINT "LunghezzaPasswordAmministratore"
        CHECK (LENGTH("Password") >= 6)
);
```

Listing 2: Tabella *Amministratore*

4.1.3 Definizione della tabella *Prenotazione*

```
CREATE DOMAIN AEROPORTO."StatoPrenotazione" AS VARCHAR(10)
    CHECK (VALUE IN ('Confermata','InAttesa','Cancellata','Usufruita'));

CREATE TABLE AEROPORTO."Prenotazione" (
    -- Definizione dei dati
    "IdPrenotazione"   SERIAL,
    "Stato"            AEROPORTO."StatoPrenotazione" DEFAULT 'InAttesa',
    "PostoAssegnato"   INTEGER                       NOT NULL,
    "NumeroBagagli"    INTEGER                       NOT NULL,
    "NumeroBiglietto"  INTEGER                       NOT NULL,

    -- Definizione dei vincoli di partecipazione
    "IdGenerico"       INTEGER                       NOT NULL,
    "Codice"           INTEGER                       NOT NULL,
    "NumeroDocumento"  VARCHAR(20)                   NOT NULL,

    -- Definizione dei vincoli basati sullo schema
    CONSTRAINT "PrenotazioneChiavePrimaria"
        PRIMARY KEY ("IdPrenotazione"),
    CONSTRAINT "NumeroBigliettoChiaveUnica"
        UNIQUE ("NumeroBiglietto"),
    CONSTRAINT "PrenotazioneChiaveEsternaVolo"
        FOREIGN KEY ("Codice")
        REFERENCES AEROPORTO."Volo" ("Codice")
            ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT "PrenotazioneChiaveEsternaGenerico"
        FOREIGN KEY ("IdGenerico")
        REFERENCES AEROPORTO."Generico" ("IdGenerico")
            ON DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT "PrenotazioneChiaveEsternaPasseggero"
        FOREIGN KEY ("NumeroDocumento")
        REFERENCES AEROPORTO."Passeggero" ("NumeroDocumento")
            ON DELETE RESTRICT  ON UPDATE CASCADE
);
```

Listing 3: Tabella *Prenotazione*

4.1.4 Definizione della tabella *StoricoPrenotazione*

```
CREATE DOMAIN AEROPORTO."StatoStoricoPrenotazione" AS
    AEROPORTO."StatoPrenotazione"
    CHECK (VALUE IN ('Cancellata','Usufruita'));

CREATE TABLE AEROPORTO."StoricoPrenotazione" (
    -- Definizione dei dati
    "IdStoricoPrenotazione"   SERIAL,                                NOT NULL,
    "CodicePrenotazione"      INTEGER                                NOT NULL,
    "Stato"                   AEROPORTO."StatoStoricoPrenotazione" NOT NULL,
    "PostoAssegnato"          INTEGER                                NOT NULL,
    "NumeroBagagli"           INTEGER                                NOT NULL,
    "NumeroBiglietto"         INTEGER                                NOT NULL,

    -- Definizione dei vincoli di partecipazione
    "Codice"                  INTEGER                                NOT NULL,

    -- Definizione dei vincoli basati sullo schema
    CONSTRAINT "StoricoPrenotazioneChiavePrimaria"
        PRIMARY KEY ("IdStoricoPrenotazione"),
    CONSTRAINT "StoricoPrenotazioneChiaveEsternaVolo"
        FOREIGN KEY ("Codice")
        REFERENCES AEROPORTO."Volo" ("Codice")
            ON DELETE RESTRICT ON UPDATE CASCADE
);
```

Listing 4: Tabella *StoricoPrenotazione*

4.1.5 Definizione della tabella *Passeggero*

```
CREATE DOMAIN AEROPORTO."Sesso" AS CHAR(1)
    CHECK (VALUE IN ('M','F','X'));

CREATE TABLE AEROPORTO."Passeggero" (
    -- Definizione dei dati
    "NumeroDocumento" VARCHAR(20)       NOT NULL,
    "Nome"            VARCHAR(50)       NOT NULL,
    "Cognome"         VARCHAR(50)       NOT NULL,
    "DataNascita"     DATE              NOT NULL,
    "Sesso"           AEROPORTO."Sesso" NOT NULL,

    -- Definizione dei vincoli basati sullo schema
    CONSTRAINT "PasseggeroChiavePrimaria"
        PRIMARY KEY ("NumeroDocumento"),
);
```

Listing 5: Tabella *Passeggero*

4.1.6 Definizione della tabella *CompagniaAerea*

```
    CREATE TABLE AEROPORTO."CompagniaAerea" (
        -- Definizione dei dati
        "IdCompagnia" SERIAL      NOT NULL,
        "Nome"        VARCHAR(50) NOT NULL,
        "Nazione"     VARCHAR(50) NOT NULL,

        -- Definizione dei vincoli basati sullo schema
        CONSTRAINT "CompagniaAereaChiavePrimaria"
            PRIMARY KEY ("IdCompagnia"),
        CONSTRAINT "CompagniaAereaChiaveUnivoca"
            UNIQUE ("Nome")
);
```

Listing 6: Tabella *CompagniaAerea*

4.1.7 Definizione della tabella *Aeroporto*

```
    CREATE TABLE AEROPORTO."Aeroporto" (
        -- Definizione dei dati
        "Codice"  CHAR(3)      NOT NULL,
        "Nome"    VARCHAR(100) NOT NULL,
        "Stato"   VARCHAR(50)  NOT NULL,
        "Citta"   VARCHAR(50)  NOT NULL,

        -- Definizione dei vincoli basati sullo schema
        CONSTRAINT "AeroportoChiavePrimaria"
            PRIMARY KEY ("Codice"),
        CONSTRAINT "CodiceMaiuscolo"
            CHECK ("Codice" = UPPER("Codice")),
        CONSTRAINT "AeroportoFisicoUnivoco"
            UNIQUE ("Nome","Stato","Citta")
    );
```

Listing 7: Tabella *Aeroporto*

4.1.8 Definizione della tabella *Volo*

```
CREATE DOMAIN AEROPORTO."StatoVolo" AS VARCHAR(11)
    CHECK (VALUE IN ('Programmato','Atterrato','Cancellato',
    'InRitardo','Decollato','Chiuso'));

CREATE TABLE AEROPORTO."Volo" (
    -- Definizione dei dati
    "Codice"                        SERIAL,
    "Stato"                         AEROPORTO."StatoVolo"     DEFAULT 'Programmato',
    "OrarioPartenza"                TIMESTAMP WITH TIME ZONE  NOT NULL,
    "OrarioArrivo"                  TIMESTAMP WITH TIME ZONE  NOT NULL,
    "Prenotabile"                   BOOLEAN                   DEFAULT TRUE,

    -- Definizione dei vincoli di partecipazione
    "IdAmministratore"              INTEGER                   NOT NULL,
    "IdCompagnia"                   INTEGER                   NOT NULL,
    "CodiceAeroportoPartenza"       CHAR(3)                   NOT NULL,
    "CodiceAeroportoDestinazione"   CHAR(3)                   NOT NULL,

    -- Definizione dei vincoli basati sullo schema
    CONSTRAINT "VoloChiavePrimaria"
        PRIMARY KEY ("Codice"),
    CONSTRAINT "VoloChiaveEsternaAmministratore"
        FOREIGN KEY ("IdAmministratore")
        REFERENCES AEROPORTO."Amministratore" ("IdAmministratore")
            ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT "VoloChiaveEsternaCompagnia"
        FOREIGN KEY ("IdCompagnia")
        REFERENCES AEROPORTO."CompagniaAerea" ("IdCompagnia")
            ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT "VoloChiaveEsternaAeroportoPartenza"
        FOREIGN KEY ("CodiceAeroportoPartenza")
        REFERENCES AEROPORTO."Aeroporto" ("Codice")
            ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT "VoloChiaveEsternaAeroportoDestinazione"
        FOREIGN KEY ("CodiceAeroportoDestinazione")
        REFERENCES AEROPORTO."Aeroporto" ("Codice")
            ON DELETE RESTRICT ON UPDATE CASCADE,
    CONSTRAINT "ArrivoMaggioreDiPartenza"
        CHECK ("OrarioArrivo" > "OrarioPartenza")
);
```

Listing 8: Tabella *Volo*

4.1.9 Definizione della tabella *Gate*

```
    CREATE TABLE AEROPORTO."Gate" (
        -- Definizione dei dati
        "Numero" INTEGER NOT NULL,

        -- Definizione dei vincoli basati sullo schema
        CONSTRAINT "GateChiavePrimaria"
            PRIMARY KEY ("Numero")
    );
```

Listing 9: Tabella *Gate*

4.1.10 Definizione della tabella *Assegnazione*

```
CREATE DOMAIN AEROPORTO."StatoOccupazione" AS VARCHAR(11)
    CHECK (VALUE IN ('Pianificato','Attivo','Annullato',
    'Completato'));

CREATE TABLE AEROPORTO."Assegnazione" (
    -- Definizione dei dati
    "Codice"                    INTEGER                     NOT NULL,
    "Numero"                    INTEGER                     NOT NULL,
    "OrarioInizioAssegnazione"  TIMESTAMP WITH TIME ZONE    NOT NULL,
    "OrarioFineAssegnazione"    TIMESTAMP WITH TIME ZONE    NOT NULL,
    "Stato"                     AEROPORTO."StatoOccupazione"  DEFAULT 'Pianificato',

    -- Definizione dei vincoli basati sullo schema
    CONSTRAINT "AssegnazioneChiavePrimaria"
        PRIMARY KEY ("Codice"),
    CONSTRAINT "AssegnazioneChiaveEsternaVolo"
        FOREIGN KEY ("Codice")
        REFERENCES AEROPORTO."Volo" ("Codice")
            ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT "AssegnazioneChiaveEsternaGate"
        FOREIGN KEY ("Numero")
        REFERENCES AEROPORTO."Gate" ("Numero")
            ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT "ControlloOrarioAssegnazione"
        CHECK ("OrarioFineAssegnazione" > "OrarioInizioAssegnazione")
);
```

Listing 10: Tabella *Assegnazione*

4.1.11 Definizione di una sequenza

```
CREATE SEQUENCE "IdUtente"
START WITH 1
INCREMENT BY 1;
```

Listing 11: Sequenza

4.2 Definizione dei vincoli di integrità semantici

Qui di seguito sono stati implementati i vincoli specificati nella fase di progettazione logica e non esprimibili come vincoli basati sullo schema relazionale.

4.2.1 Vincolo *AssegnazioneIdGenerico*

```
CREATE FUNCTION AEROPORTO.assegnazioneIdGenerico()
RETURNS TRIGGER AS $$
BEGIN
    NEW."IdGenerico" := nextval('AEROPORTO."IdUtente"');
    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER AssegnazioneIdGenerico
BEFORE INSERT ON AEROPORTO."Generico"
FOR EACH ROW
    EXECUTE FUNCTION AEROPORTO.assegnazioneIdGenerico();
```

Listing 12: Trigger *AssegnazioneIdGenerico*

4.2.2 Vincolo *AssegnazioneIdAmministratore*

```
CREATE FUNCTION AEROPORTO.assegnazioneIdAmministratore()
RETURNS TRIGGER AS $$
BEGIN
    NEW."IdAmministratore" := nextval('AEROPORTO."IdUtente"');
    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER AssegnazioneIdAmministratore
BEFORE INSERT ON AEROPORTO."Amministratore"
FOR EACH ROW
    EXECUTE FUNCTION AEROPORTO.assegnazioneIdAmministratore();
```

Listing 13: Trigger *AssegnazioneIdAmministratore*

4.2.3 Vincolo *PartenzaArrivoNapoli*

```
CREATE FUNCTION AEROPORTO.partenzaArrivoNapoli()
RETURNS TRIGGER AS $$
DECLARE
    arrivo   VARCHAR;
    partenza VARCHAR;
BEGIN
    SELECT Partenza."Codice", Arrivo."Codice"
        INTO arrivo, partenza
    FROM AEROPORTO."Aeroporto" AS Partenza,
         AEROPORTO."Aeroporto" AS Arrivo
    WHERE Partenza."Codice" = NEW."CodiceAeroportoPartenza"
    AND   Arrivo."Codice"   = NEW."CodiceAeroportoDestinazione";

    IF partenza = 'NAP' AND arrivo = 'NAP' THEN
        RAISE EXCEPTION 'Non sono ammessi voli
        interni nella stessa città.';
    ELSIF partenza <> 'NAP' AND arrivo <> 'NAP' THEN
        RAISE EXCEPTION 'Il volo deve necessariamente
        partire o arrivare a Napoli.';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER voli
BEFORE INSERT OR UPDATE OF
    "CodiceAeroportoPartenza", "CodiceAeroportoDestinazione"
ON AEROPORTO."Volo"
FOR EACH ROW
    EXECUTE FUNCTION AEROPORTO.PartenzaArrivoNapoli();
```

Listing 14: Trigger *PartenzaArrivoNapoli*

4.2.4 Vincolo *UnicoDocumentoPerVolo*

```
    CREATE FUNCTION AEROPORTO.unicoDocumentoPerVolo()
    RETURNS TRIGGER AS $$
    BEGIN
        IF EXISTS (
            SELECT 1 FROM AEROPORTO."Prenotazione"
            WHERE  "Codice"          = NEW."Codice"
            AND    "NumeroDocumento" = NEW."NumeroDocumento")
        THEN
            RAISE EXCEPTION 'Numero documento già esistente!';
        END IF;

        RETURN NEW;
    END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER UnicoDocumentoPerVolo
BEFORE INSERT ON AEROPORTO."Prenotazione"
FOR EACH ROW
        EXECUTE FUNCTION AEROPORTO.unicoDocumentoPerVolo();
```

Listing 15: Trigger *UnicoDocumentoPerVolo*

4.2.5 Vincolo *PuliziaOrfani*

```
    CREATE FUNCTION AEROPORTO.puliziaOrfani()
    RETURNS TRIGGER AS $$
    BEGIN
        DELETE FROM AEROPORTO."Passeggero"
        WHERE NumeroDocumento = OLD.NumeroDocumento;
        RETURN NULL;
    END;
    $$ LANGUAGE PLPGSQL;

    CREATE TRIGGER PuliziaOrfani
    AFTER DELETE ON AEROPORTO."Prenotazione"
    FOR EACH ROW
        EXECUTE FUNCTION AEROPORTO.pulizaOrfani();
```

Listing 16: Trigger *PuliziaOrfani*

4.2.6 Vincolo *ArchiviazionePrenotazioni*

```
CREATE FUNCTION AEROPORTO.archiviazionePrenotazioni()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO AEROPORTO."StoricoPrenotazione" (
        "CodicePrenotazione","Stato","PostoAssegnato",
        "NumeroBagagli","NumeroBiglietto","Codice"
    )
    VALUES (
        NEW."IdPrenotazione",NEW."Stato",NEW."PostoAssegnato",
        NEW."NumeroBagagli",NEW."NumeroBiglietto",NEW."Codice"
    );

    DELETE FROM AEROPORTO."Prenotazione"
    WHERE  "IdPrenotazione" = NEW."IdPrenotazione";

    RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER ArchiviazionePrenotazioni
AFTER UPDATE OF "Stato" ON AEROPORTO."Prenotazione"
FOR EACH ROW
    WHEN (NEW."Stato" = 'Cancellata' OR NEW."Stato" = 'Usufruita')
        EXECUTE FUNCTION AEROPORTO.archiviazionePrenotazioni();
```

Listing 17: Trigger *ArchiviazionePrenotazioni*

4.2.7 Vincolo *VoloPrenotabile*

```
CREATE FUNCTION AEROPORTO.voloPrenotabile()
RETURNS TRIGGER AS $$
DECLARE
    prenotabile BOOLEAN;
BEGIN
    SELECT volo."Prenotabile"
        INTO prenotabile
    FROM AEROPORTO."Volo" AS volo
    WHERE "Codice" = NEW."Codice";

    IF prenotabile = FALSE THEN
        RAISE EXCEPTION 'Il volo non è prenotabile!';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER VoloPrenotabile
BEFORE INSERT ON AEROPORTO."Prenotazione"
FOR EACH ROW
    EXECUTE FUNCTION AEROPORTO.voloPrenotabile();
```

Listing 18: Trigger *VoloPrenotabile*

4.2.8 Vincolo *CancellazionePrenotazione*

```
CREATE FUNCTION AEROPORTO.cancellazionePrenotazione()
RETURNS TRIGGER AS $$
DECLARE
    stato VARCHAR;
BEGIN
    SELECT "Stato"
        INTO stato
    FROM  AEROPORTO."Volo"
    WHERE "Codice" = NEW."Codice";

    IF stato <> 'Programmato' AND
       stato <> 'InRitardo'   AND
       stato <> 'Cancellato'
    THEN
        RAISE EXCEPTION 'Non è possibile annullare
        la prenotazione!';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER CancellazionePrenotazione
BEFORE UPDATE OF "Stato" ON AEROPORTO."Prenotazione"
FOR EACH ROW
    WHEN (NEW."Stato" = 'Cancellata')
        EXECUTE FUNCTION AEROPORTO.cancellazionePrenotazione();
```

Listing 19: Trigger *CancellazionePrenotazione*

4.2.9 Vincolo *ControlloPostoUnicoPerVolo*

```
CREATE FUNCTION AEROPORTO.controlloPostoUnicoPerVolo()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM AEROPORTO."Prenotazione"
        WHERE  "Codice" = NEW."Codice"                  AND
               "PostoAssegnato"  = NEW."PostoAssegnato" AND
               "IdPrenotazione" <> NEW."IdPrenotazione"
    ) THEN
        RAISE EXCEPTION 'Conflitto di posti a sedere!';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER ControlloPostoUnicoPerVolo
BEFORE INSERT OR UPDATE ON AEROPORTO."Prenotazione"
FOR EACH ROW
    EXECUTE FUNCTION AEROPORTO.controlloPostoUnicoPerVolo();
```

Listing 20: Trigger *ControlloPostoUnicoPerVolo*

4.2.10 Vincolo *StatoVoloCancellato*

```
CREATE FUNCTION AEROPORTO.statoVoloCancellato()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD."Stato" <> 'Programmato' AND
       OLD."Stato" <> 'InRitardo'   AND
       OLD."Stato" <> 'Cancellato'
    THEN
        RAISE EXCEPTION 'Non è possibile
        cambiare lo stato in Cancellato';
    ELSE
        UPDATE AEROPORTO."Prenotazione"
        SET    "Stato"  = 'Cancellata'
        WHERE  "Codice" = NEW."Codice";

        UPDATE AEROPORTO."Assegnazione"
        SET    "Stato"  = 'Annullato'
        WHERE  "Codice" = NEW."Codice";

        IF NEW."Prenotabile" <> FALSE THEN
           NEW."Prenotabile" := FALSE;
        END IF;

        RETURN NEW;
    END IF;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER StatoVoloCancellato
BEFORE UPDATE OF "Stato" ON AEROPORTO."Volo"
FOR EACH ROW
    WHEN (NEW."Stato" = 'Cancellato')
        EXECUTE FUNCTION AEROPORTO.StatoVoloCancellato();
```

Listing 21: Trigger *StatoVoloCancellato*

4.2.11 Vincolo *StatoVoloAtterrato*

```
CREATE FUNCTION AEROPORTO.statoVoloAtterrato()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD."Stato" <> 'Decollato' AND
       OLD."Stato" <> 'Atterrato'
    THEN
        RAISE EXCEPTION 'Non è possibile
        cambiare lo stato in Atterrato';
    ELSE
        UPDATE AEROPORTO."Prenotazione"
        SET    "Stato" = 'Usufruita'
        WHERE  "Codice" = NEW."Codice";

        UPDATE AEROPORTO."Assegnazione"
        SET    "Stato"  = 'Completato'
        WHERE  "Codice" = NEW."Codice";

        RETURN NEW;
    END IF;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER StatoVoloAtterrato
BEFORE UPDATE OF "Stato" ON AEROPORTO."Volo"
FOR EACH ROW
    WHEN (NEW."Stato" = 'Atterrato')
        EXECUTE FUNCTION AEROPORTO.statoVoloAtterrato();
```

Listing 22: Trigger *StatoVoloAtterrato*

4.2.12 Vincolo *StatoVoloChiuso*

```
CREATE FUNCTION AEROPORTO.statoVoloChiuso()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD."Stato" <> 'Programmato' AND
       OLD."Stato" <> 'Chiuso'
    THEN
        RAISE EXCEPTION 'Non è possibile
        cambiare lo stato in Chiuso';
    ELSE
        UPDATE AEROPORTO."Prenotazione"
        SET    "Stato"  = 'Confermata'
        WHERE  "Codice" = NEW."Codice";

        IF NEW."Prenotabile" <> FALSE THEN
           NEW."Prenotabile" := FALSE;
        END IF;

        RETURN NEW;
    END IF;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER StatoVoloChiuso
BEFORE UPDATE OF "Stato" ON AEROPORTO."Volo"
FOR EACH ROW
    WHEN (NEW."Stato" = 'Chiuso')
        EXECUTE FUNCTION AEROPORTO.statoVoloChiuso();
```

Listing 23: Trigger *StatoVoloChiuso*

4.2.13 Vincolo *StatoVoloProgrammato*

```
CREATE FUNCTION AEROPORTO.statoVoloProgrammato()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD."Stato" <> 'InRitardo' AND
       OLD."Stato" <> 'Programmato'
    THEN
        RAISE EXCEPTION 'Non è possibile
        cambiare lo stato in Programmato';
    ELSE
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER StatoVoloProgrammato
BEFORE UPDATE OF "Stato" ON AEROPORTO."Volo"
FOR EACH ROW
    WHEN (NEW."Stato" = 'Programmato')
        EXECUTE FUNCTION AEROPORTO.statoVoloProgrammato();
```

Listing 24: Trigger *StatoVoloProgrammato*

4.2.14 Vincolo *StatoVoloInRitardo*

```
CREATE FUNCTION AEROPORTO.statoVoloInRitardo()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD."Stato" <> 'Programmato' AND
       OLD."Stato" <> 'InRitardo'
    THEN
        RAISE EXCEPTION 'Non è possibile
        cambiare lo stato in Programmato';
    ELSE
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER StatoVoloInRitardo
BEFORE UPDATE OF "Stato" ON AEROPORTO."Volo"
FOR EACH ROW
    WHEN (NEW."Stato" = 'InRitardo')
        EXECUTE FUNCTION AEROPORTO.statoVoloInRitardo();
```

Listing 25: Trigger *StatoVoloInRitardo*

4.2.15 Vincolo *StatoOccupazioneAttivo*

```
CREATE FUNCTION AEROPORTO.statoOccupazioneAttivo()
RETURNS TRIGGER AS $$
BEGIN
    IF OLD."Stato" <> 'Pianificato' AND
       OLD."Stato" <> 'Attivo'
    THEN
        RAISE EXCEPTION 'Non è possibile
        cambiare lo stato in Attivo';
    ELSE
        RETURN NEW;
    END IF;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER StatoOccupazioneAttivo
BEFORE UPDATE OF "Stato" ON AEROPORTO."Assegnazione"
FOR EACH ROW
    WHEN (NEW."Stato" = 'Attivo')
        EXECUTE FUNCTION AEROPORTO.statoOccupazioneAttivo();
```

Listing 26: Trigger *StatoOccupazioneAttivo*

4.2.16 Vincolo *AssegnazioneGate*

```
CREATE FUNCTION AEROPORTO.assegnazioneGate()
RETURNS TRIGGER AS $$
DECLARE
    OrarioPartenza      TIMESTAMP WITH TIME ZONE;
    OrarioArrivo        TIMESTAMP WITH TIME ZONE;
    AeroportoPartenza   CHAR(3);
    AeroportoArrivo     CHAR(3);
BEGIN
    SELECT "OrarioPartenza", "CodiceAeroportoPartenza",
           "OrarioArrivo",   "CodiceAeroportoDestinazione"
    INTO    OrarioPartenza, AeroportoPartenza,
            OrarioArrivo,   AeroportoArrivo
    FROM    AEROPORTO."Volo"
    WHERE  "Codice" = NEW."Codice";

    IF AeroportoPartenza = 'NAP' THEN
        NEW."OrarioInizioAssegnazione" :=
            (OrarioPartenza - INTERVAL '50 min');
        NEW."OrarioFineAssegnazione"   :=
            (OrarioPartenza + INTERVAL '10 min');
    ELSIF AeroportoArrivo = 'NAP' THEN
        NEW."OrarioInizioAssegnazione" :=
            (OrarioArrivo - INTERVAL '10 min');
        NEW."OrarioFineAssegnazione"   :=
            (OrarioArrivo + INTERVAL '45 min');
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER GateAssegnazione
BEFORE INSERT OR UPDATE ON AEROPORTO."Assegnazione"
FOR EACH ROW
    EXECUTE FUNCTION AEROPORTO.assegnazioneGate();
```

Listing 27: Trigger *AssegnazioneGate*

4.2.17 Vincolo *OrarioConflittoAssegnazioneGate*

```
CREATE FUNCTION AEROPORTO.orarioConflittoAssegnazioneGate()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM AEROPORTO."Assegnazione"
        WHERE "Numero" = NEW."Numero"               AND
              "Codice" <> NEW."Codice"              AND
              "OraRioInizioAssegnazione" <
                    NEW."OrarioFineAssegnazione"    AND
              "OrarioFineAssegnazione"    >
                    NEW."OrarioInizioAssegnazione" AND
              "Stato" IN ('Pianificato','Attivo')
    ) THEN
        RAISE EXCEPTION 'Conflitto di orari!';
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER OrarioConflittoAssegnazioneGate
BEFORE INSERT OR UPDATE ON AEROPORTO."Assegnazione"
FOR EACH ROW
    EXECUTE FUNCTION AEROPORTO.orarioConflittoAssegnazioneGate();
```

Listing 28: Trigger *OrarioConflittoAssegnazioneGate*

4.2.18 Vincolo *AggiornamentoAssegnazioniGate*

```
CREATE FUNCTION AEROPORTO.aggiornamentoAssegnazioniGate()
RETURNS TRIGGER AS $$
BEGIN
    IF EXISTS (
        SELECT 1 FROM AEROPORTO."Assegnazione"
        WHERE "Codice" = NEW."Codice"
    ) THEN
        UPDATE AEROPORTO."Assegnazione"
        SET    "Codice" = "Codice"
        WHERE  "Codice" = NEW."Codice";
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;

CREATE TRIGGER AggiornamentoAssegnazioniGate
AFTER INSERT OR UPDATE ON AEROPORTO."Volo"
FOR EACH ROW
    EXECUTE FUNCTION AEROPORTO.aggiornamentoAssegnazioniGate();
```

Listing 29: Trigger *AggiornamentoAssegnazioniGate*