Università degli Studi di Napoli
Federico II

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Elaborato per il Corso di

Basi di Dati

Anno Accademico 2024/2025

Sistema Informativo per la Gestione dell'Aeroporto di Napoli (Traccia 3)

Studente:

Marco Lo Cunsolo
Matricola: N86004534

Docente:

Prof. Silvio Barra

Indice

1 Introduzione

Questo elaborato documenta le fasi di progettazione e sviluppo di un sistema informativo per la gestione dell'Aeroporto di Napoli, realizzato nell'ambito del corso di Basi di Dati. L'obiettivo primario di tale sistema è organizzare e monitorare in maniera efficiente e intuitiva tutte le operazioni aeroportuali. Il sistema è progettato per supportare sia gli utenti generici, che potranno gestire le proprie prenotazioni di volo, sia gli amministratori, responsabili dell'inserimento e aggiornamento dei voli e dell'assegnazione dei gate. Il database relazionale sottostante gestirà un'ampia gamma di informazioni, tra cui i dettagli completi dei voli (arrivi e partenze, compagnie, orari, stati e ritardi), le prenotazioni dei passeggeri con i relativi dati e lo stato della prenotazione, e l'assegnazione dei gate di imbarco.

2 Progettazione Concettuale

In questo capitolo viene documentata la fase di progettazione concettuale del sistema informativo per la gestione dell'Aeroporto di Napoli. Partendo da un'attenta analisi dei requisiti, si è giunti alla definizione di uno schema concettuale del database. Tale schema, rappresentato attraverso due class diagrams equivalenti, uno UML e l'altro ER, cattura l'essenza delle informazioni necessarie al sistema, identificando le entità chiave del dominio, le relazioni significative che le collegano e i vincoli da imporre. L'approccio adottato assicura che il modello risultante sia indipendente da qualsiasi specifica tecnologia di implementazione o DBMS, focalizzandosi esclusivamente sulla rappresentazione logica e coerente dei dati.

2.1 Class Diagram UML

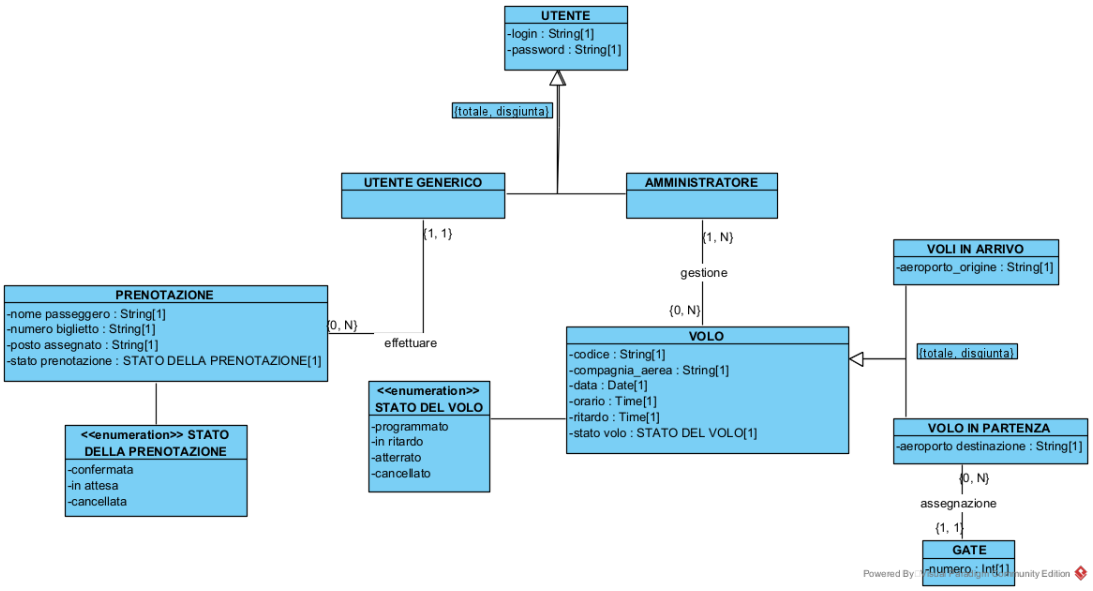

Figura 1: Class Diagram UML Iniziale del Sistema Aeroporto

2.2 Class Diagram ER

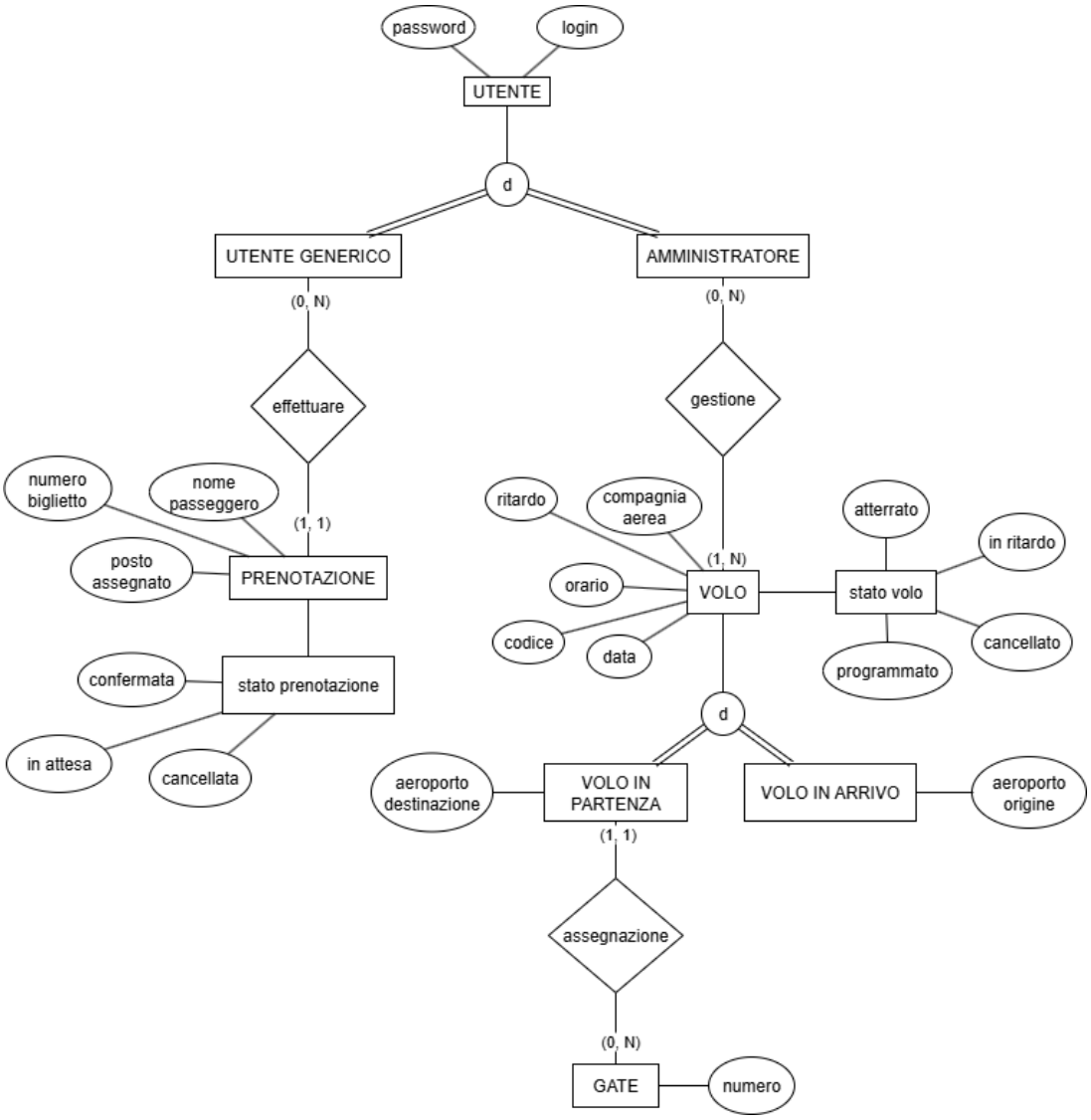

Figura 2: Class Diagram ER Iniziale del Sistema Aeroporto

3 Ristrutturazione del Class Diagram

Si procede alla fase di ristrutturazione del Class Diagram UML inizialmente definito. L'obiettivo primario di questa operazione è ottimizzare il modello concettuale, rendendolo maggiormente idoneo per una successiva traduzione in schemi relazionali e per migliorare l'efficienza complessiva del database. La ristrutturazione sarà condotta attraverso un'analisi approfondita di specifici aspetti del diagramma. Saranno seguiti i seguenti punti:

- Analisi delle ridondanze.
- Eliminazione delle generalizzazioni.
- Eliminazioni degli attributi multivalore.
- Eliminazione degli attributi strutturati.
- Partizionamento/accorpamento di entità e associazioni.
- Scelta degli identificatori primari.

3.0.1 Analisi delle ridondanze

L'analisi delle ridondanze consiste nell'individuare all'interno dello schema concettuale la presenza di dati derivabili da altri dati già presenti, ed eliminarli nel caso siano svantaggiosi. Una ridondanza può presentare svantaggi, come un aumento dell'occupazione di memoria o del costo delle operazioni di aggiornamento, e vantaggi, come la semplificazione delle interrogazioni.

Nello schema concettuale prodotto, non sono presenti attributi ridondanti o derivabili.

3.0.2 Eliminazione delle generalizzazioni

L'analisi delle gerarchie di specializzazione è cruciale per la traduzione del modello concettuale in uno schema relazionale, in quanto i sistemi tradizionali di database non consentono di rappresentare in modo diretto le generalizzazioni e le specializzazioni.
Per il sistema aeroportuale, sono state identificate due principali gerarchie:

- **Gerarchia dell'entità *Utente*:** Inizialmente, l'entità *Utente* presentava due specializzazioni in *Utente Generico* e *Amministratore*. Sebbene queste sottoclassi differiscano significativamente per le operazioni che possono eseguire (es. *effettuarePrenotazione* per l'utente generico e *aggiornaVolo* per l'amministratore), ricordiamo che per il database in questione i metodi non hanno importanza, in quanto sarà compito dell'applicativo gestirli. I loro attributi sono gli stessi, ossia *login* e *password*, ereditati dal padre. Per evitare ridondanze dei dati comuni e semplificare le interrogazioni sull'intera popolazione di utenti, si è optato per **l'appiattimento di questa gerarchia in una singola entità *Utente*** nel modello ristrutturato. Tale entità sarà arricchita da un nuovo attributo *ruolo*, che potrà assumere il valore di

'Utente Generico' o 'Amministratore'. Le diverse funzionalità e permessi associati a ciascun ruolo verranno gestiti a livello dell'applicazione, basandosi sul valore di questo attributo.

- **Gerarchia dell'entità *Volo*:** Similmente, l'entità *Volo* era specializzata in *Voli in Arrivo* e *Voli in Partenza*. Anche in questo caso, le differenze negli attributi tra le sottoclassi sono minime rispetto agli attributi comuni. La gestione di due tabelle distinte per voli che condividono la maggior parte delle informazioni comporterebbe una maggiore complessità nelle query. Pertanto, si è deciso di **appiattire anche questa gerarchia in una singola entità *Volo*** nel modello ristrutturato. Un attributo *tipo volo*, che potrà assumere i valori di 'Arrivo' o 'Partenza', sarà introdotto per distinguere i tipi di volo. Per la relazione con l'entità *Gate* della ex-sottoclasse *VoloInPartenza* verrà introdotta la partecipazione parziale dipendente dal tipo di volo.

3.0.3 Eliminazione degli attributi multivalore

In questa fase si procede all'identificazione e all'eliminazione degli attributi a valore multiplo, ovvero quegli attributi che, per una singola istanza di un'entità, possono assumere più di un valore. Il modello relazionale non permette la rappresentazione di attributi multivalore.

Nel contesto del sistema di gestione aeroportuale, non sono presenti attributi multivalore.

3.0.4 Eliminazione degli attributi strutturati

Gli attributi strutturati introducono un'inutile complessità e tendono a complicare le query, oltre a non essere supportati dal modello relazionale.

In questo sistema non sono presenti attributi strutturati.

3.0.5 Partizionamento/accorpamento di entità e associazioni.

In alcune circostanze, può essere vantaggioso unire entità quando le loro funzioni utilizzano informazioni comuni a entrambe, oppure suddividere un'entità quando risulta eccessivamente complessa.

In questo caso non è ritenuto necessario partizionare o accorpare entità.

3.0.6 Scelta degli identificatori primari.

Questa fase della ristrutturazione è dedicata alla selezione e definizione degli identificatori primari per ciascuna entità, essenziali per garantire l'unicità di ogni record e facilitare le relazioni tra le entità.

Per il sistema di gestione aeroportuale, si è optato per l'utilizzo prevalente di **chiavi**

surrogate per le entità principali. Nonostante la presenza di possibili chiavi primarie, come Volo(codice) e Gate(id), si è scelto di utilizzare chiavi surrogate per garantire la massima stabilità e immutabilità degli identificatori, rendendoli completamente indipendenti da qualsiasi evoluzione o riorganizzazione futura dei codici.
Di seguito sono elencati gli identificatori primari selezionati per le entità principali del modello: Utente(**idUtente**), Prenotazione(**idPrenotazione**), Volo(**idVolo**), Gate(**idGate**).

3.0.7 Ulteriori modifiche

Durante l'analisi del modello e delle sue funzionalità, è emersa la necessità di introdurre una nuova associazione tra la classe *PRENOTAZIONE* e la classe *VOLO*. Sebbene non esplicitata nel diagramma concettuale iniziale, è fondamentale che ogni prenotazione sia univocamente collegata al volo a cui si riferisce.

3.1 Class Diagram UML Ristrutturato

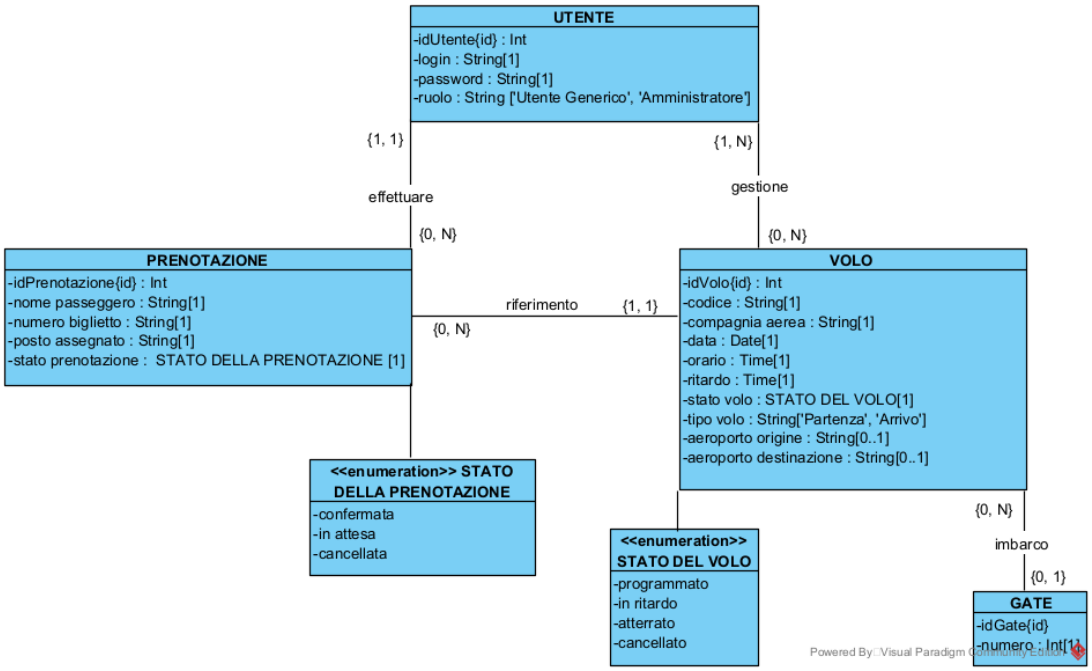

Figura 3: Class Diagram UML Ristrutturato del Sistema Aeroporto

3.2 Class Diagram Ristrutturato

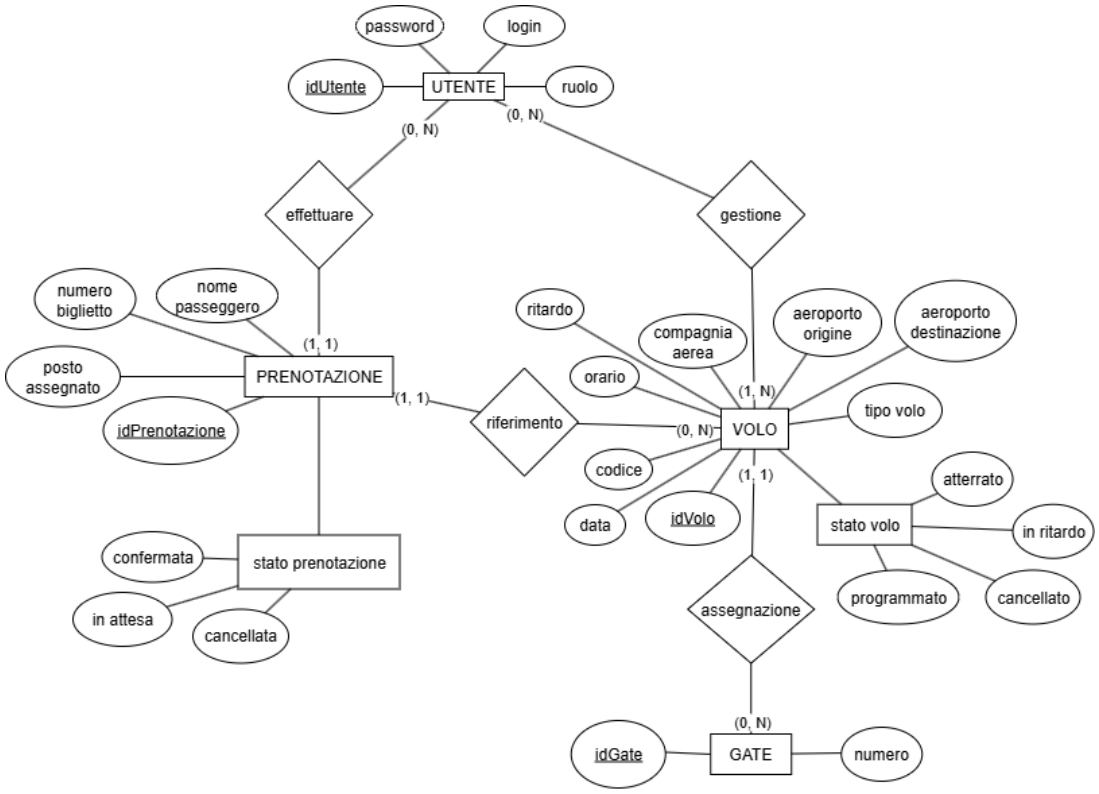

Figura 4: Class Diagram ER Ristrutturato del Sistema Aeroporto

4 Dizionari

I dizionari sono una documentazione fondamentale del database, in quanto forniscono una descrizione dettagliata di ogni elemento della sua progettazione. Di seguito vi sono il dizionario delle classi, il dizionario delle associazioni e il dizionario dei vincoli.

4.1 Dizionario delle Classi

Questo dizionario fornisce una descrizione dettagliata delle entità (classi) identificate nel Class Diagram Ristrutturato, elencando i loro attributi principali con il relativo tipo, la molteplicità e una breve spiegazione. Ogni voce specifica il nome della classe, una descrizione del suo scopo nel sistema e la lista dei suoi attributi.

Classe	Descrizione	Attributi
UTENTE	Rappresenta un utente generico del sistema, dotato di credenziali di accesso e di un ruolo definito, che può essere 'Utente Generico' o 'Amministratore'.	• **idUtente{id}**: `Int` - Identificatore univoco del sistema per l'utente. • **login**: `String[1]` - Nome utente utilizzato per l'accesso al sistema. • **password**: `String[1]` - Credenziale segreta per l'autenticazione dell'utente. • **ruolo**: `String ['Utente Generico','Amministratore']` - Specifica il tipo di utente e le sue autorizzazioni nel sistema.
PRENOTAZIONE	Registra i dettagli di una specifica prenotazione effettuata da un passeggero su un volo, includendo informazioni identificative e lo stato corrente della prenotazione.	• **idPrenotazione{id}**: `Int` - Identificatore univoco del sistema per la prenotazione. • **nome passeggero**: `String[1]` - Nome completo del passeggero associato alla prenotazione. • **numero biglietto**: `String[1]` - Codice univoco del biglietto emesso per la prenotazione. • **posto assegnato**: `String[1]` - Codice del posto assegnato al passeggero sul volo. • **stato prenotazione**: `STATO DELLA PRENOTAZIONE[1]` - Stato attuale della prenotazione (es. confermata, in attesa, cancellata). • **idVoloFK**: `Int[1]` - Chiave esterna che collega la prenotazione al volo a cui si riferisce.

Classe	Descrizione	Attributi
VOLO	Contiene tutte le informazioni essenziali relative a un volo, come il codice identificativo, la compagnia aerea, gli orari, lo stato attuale, il tipo (partenza o arrivo) e i relativi aeroporti.	• **idVolo{id}**: `Int` - Identificatore univoco del sistema per il volo. • **codice**: `String[1]` - Codice alfanumerico identificativo del volo (es. AZ123). • **compagnia aerea**: `String[1]` - Nome della compagnia aerea che opera il volo. • **data**: `Date[1]` - Data prevista del volo. • **orario**: `Time[1]` - Orario previsto di partenza o arrivo del volo. • **ritardo**: `Time[1]` - Durata del ritardo del volo, se applicabile. • **stato volo**: `STATO DEL VOLO[1]` - Stato operativo corrente del volo (es. programmato, in ritardo, atterrato). **tipo volo**: `String ['Partenza','Arrivo']` - Indica se il volo è in partenza o in arrivo. • **aeroporto origine**: `String[0 .. 1]` - Nome dell'aeroporto di origine (se applicabile). • **aeroporto destinazione**: `String[0 .. 1]` - Nome dell'aeroporto di destinazione (se applicabile).
«enumeration» STATO DELLA PRENOTAZIONE	Definisce l'insieme chiuso e predefinito dei possibili stati in cui può trovarsi una prenotazione all'interno del sistema.	• **confermata** - La prenotazione è stata accettata e confermata. • **in attesa** - La prenotazione è in fase di elaborazione o attesa di conferma. • **cancellata** - La prenotazione è stata annullata.
«enumeration» STATO DEL VOLO	Definisce l'insieme chiuso e predefinito dei possibili stati operativi in cui può trovarsi un volo.	• **programmato** - Il volo è previsto e in orario. • **in ritardo** - Il volo ha subito un ritardo rispetto all'orario previsto. • **atterrato** - Il volo è arrivato a destinazione. • **cancellato** - Il volo è stato annullato.
GATE	Rappresenta un gate d'imbarco o sbarco fisico all'interno dell'aeroporto, identificato univocamente.	• **idGate{id}**: `Int` - Identificatore univoco del sistema per il gate. • **numero**: `Int[1]` - Numero o codice identificativo del gate (es. A01, B12).

4.2 Dizionario delle Associazioni

Questo dizionario fornisce una descrizione dettagliata delle associazioni identificate nel Class Diagram, elencando le classi coinvolte, le loro molteplicità e una breve descrizione della relazione.

Associazione	Classi Coinvolte	Molteplicità	Descrizione
effettuare	UTENTE - PRENOTAZIONE	{1, 1} - {0, N}	Un utente può effettuare da zero a molteplici prenotazioni, e una prenotazione specifica è sempre effettuata da un solo utente.
gestione	UTENTE - VOLO	{1, N} - {0, N}	Un utente (amministratore) può gestire da zero a molteplici voli, e ogni volo è gestito da almeno un utente.
imbarco	VOLO - GATE	{0, N} - {0, 1}	Un volo può essere associato a nessun gate (nel caso di volo in arrivo) o a uno specifico gate (nel caso di volo in partenza), mentre ogni gate è associato a uno o nessun volo.
riferimento	VOLO - PRENOTAZIONE	{0, N} - {1, 1}	Un volo può avere da zero a molteplici prenotazioni associate, mentre ogni prenotazione è sempre riferita a esattamente un volo.

4.3 Dizionario dei Vincoli - Parte 1

Questo dizionario fornisce una descrizione dettagliata dei vincoli di integrità applicati al modello logico, elencando il nome del vincolo, la sua tipologia e una breve spiegazione del suo scopo e delle condizioni che impone.

Nome Vincolo	Tipo Vincolo	Dettaglio/Descrizione
CHK_RUOLO_UTENTE	Dominio	Definisce i valori ammessi per il ruolo dell'utente: 'Generico' o 'Admin'.
CHK_STATO_PRENOTAZIONE	Dominio	Specifica i valori ammessi per lo stato della prenotazione: 'Confermata', 'Cancellata', 'In Attesa'.
CHK_STATO_VOLO	Dominio	Definisce i valori ammessi per lo stato del volo: 'Programmato', 'In Ritardo', 'Cancellato', 'Atterrato'.
CHK_TIPO_VOLO	Dominio	Specifica i valori ammessi per il tipo di volo: 'Partenza' o 'Arrivo'.
CHK_RITARDO_NON_NEG	Dominio	Indica che il valore di *ritardo* deve essere un numero non negativo, rappresentante una durata (ad esempio in minuti).
CHK_NUMERO_GATE_POSITIVO	Dominio	Il numero del gate deve essere un valore positivo.
UNQ_CODICE_VOLO	Unicità	Assicura che il *codice* identificativo del volo sia univoco.
UNQ_LOGIN_UTENTE	Unicità	Garantisce che il *login* dell'utente sia univoco nel sistema.
UNQ_NUMERO_BIGLIETTO	Unicità	Il *numero_biglietto* della prenotazione deve essere univoco.

4.4 Dizionario dei Vincoli - Parte 2

Nome Vincolo	Tipo Vincolo	Dettaglio/Descrizione
NOTNULL_LOGIN	Non Nullità	Il campo *login* non può essere vuoto.
NOTNULL_PASSWORD	Non Nullità	Il campo *password* non può essere vuoto.
NOTNULL_NOME_PASSEGGERO	Non Nullità	Il campo *nome_passeggero* non può essere vuoto.
NOTNULL_NUMERO_BIGLIETTO	Non Nullità	Il campo *numero_biglietto* non può essere vuoto.
NOTNULL_POSTO_ASSEGNATO	Non Nullità	Il campo *posto_assegnato* non può essere vuoto (per le prenotazioni confermate).
NOTNULL_CODICE_VOLO	Non Nullità	Il campo *codice* del volo non può essere vuoto.
NOTNULL_DATA_VOLO	Non Nullità	Il campo *data* del volo non può essere vuoto.
NOTNULL_ORARIO_VOLO	Non Nullità	Il campo *orario* del volo non può essere vuoto.
NOTNULL_NUMERO_GATE	Non Nullità	Il campo *numero* del gate non può essere vuoto.
NOTNULL_TIPO_VOLO	Non Nullità	Il campo *tipo_volo* non può essere vuoto.
CHK_AEROPORTI_DIVERSI	Integrità Semantica	L'aeroporto di origine e l'aeroporto di destinazione di un volo non possono essere uguali (*aeroporto_origine* $\neq$ aeroporto_destinazione).
DEF_AEROPORTO_ORIGINE	Default	Se non specificato, il valore predefinito per *aeroporto_origine* è 'Napoli'.

4.5 Dizionario dei Vincoli - Parte 3

Nome Vincolo	Tipo Vincolo	Dettaglio/Descrizione
DEF_AEROPORTO_DESTINAZIONE	Default	Se non specificato, il valore predefinito per *aeroporto_destinazione* è 'Napoli'.
DEF_RITARDO_VOLO	Default	Se non specificato, il valore predefinito per *ritardo* è 00:00:00.
FK_PRENOTAZIONE_UTENTE	Integrità Referenziale	Collega ogni prenotazione all'utente che l'ha effettuata tramite l'attributo *idUtenteFK*, garantendo l'integrità referenziale.
FK_VOLO_GATE	Integrità Referenziale	Collega ogni volo a un gate specifico tramite l'attributo *idGateFK*, indicando il gate di imbarco/sbarco del volo.
FK_GESTIONE_UTENTE	Integrità Referenziale	Collega ogni record di gestione all'utente (amministratore) che ha effettuato l'operazione tramite *idUtenteFK*, mantenendo l'integrità referenziale.
FK_GESTIONE_VOLO	Integrità Referenziale	Collega ogni record di gestione al volo specifico che è stato gestito tramite *idVoloFK*, mantenendo l'integrità referenziale.
FK_PRENOTAZIONE_VOLO	Integrità Referenziale	Collega ogni prenotazione al volo specifico a cui si riferisce tramite l'attributo *idVoloFK*, garantendo l'integrità referenziale tra le tabelle PRENOTAZIONE e VOLO.

5 Progettazione Logica

UTENTE (idUtente, login, password, ruolo)

PRENOTAZIONE (idPrenotazione, nome_passeggero, numero_biglietto, posto_assegnato, stato_prenotazione, idUtenteFK, idVoloFK)
idUtenteFK → UTENTE.idUtente
idVoloFK → VOLO.idVolo

GATE (idGate, numero)

VOLO (idVolo, codice, compagnia_aerea, data, orario, ritardo, stato_volo, tipo_volo, aeroporto_origine, aeroporto_destinazione, idGateFK)
idGateFK → GATE.idGate

GESTIONE (idUtente, idVolo)
- idUtente → UTENTE.idUtente
- idVolo → VOLO.idVolo

5.0.1 Scelte della Progettazione Logica

Nel presente progetto, la traduzione delle associazioni dal modello concettuale a quello logico è stata effettuata come segue.

Per l'associazione **effettuare** tra UTENTE e PRENOTAZIONE (1:N), è stata aggiunta la chiave primaria di UTENTE (*idUtente*) come chiave esterna *idUtenteFK* nella tabella PRENOTAZIONE. Questa FK è NOT NULL data la partecipazione obbligatoria (1) di UTENTE.

Per l'associazione **imbarco** tra VOLO e GATE (N:1), la chiave primaria di GATE (*idGate*) è stata inserita come chiave esterna *idGateFK* nella tabella VOLO. Questa FK è NULLABLE per la partecipazione opzionale di GATE.

Per l'associazione **riferimento** tra PRENOTAZIONE e VOLO (N:1), la chiave primaria di VOLO (*idVolo*) è stata aggiunta come chiave esterna *idVoloFK* nella tabella PRENOTAZIONE.

Per l'associazione **gestione** tra UTENTE e VOLO (N:M), è stata creata una tabella di associazione intermedia denominata GESTIONE. Questa tabella ha come attributi (chiavi esterne) le chiavi primarie di UTENTE e VOLO.

6 Progettazione Fisica

Questa sezione illustra l'implementazione pratica del modello logico in uno schema di database relazionale PostgreSQL. Vengono fornite le istruzioni per la creazione delle tabelle, riflettendo le scelte sui tipi di dato (es. 'VARCHAR(100)' per le stringhe di testo), la definizione delle chiavi primarie ed esterne e l'applicazione rigorosa di tutti i vincoli di integrità come specificato nei dizionari e nella traccia (es. i set di valori precisi per gli stati di volo e prenotazione).

6.1 Definizione delle Tabelle

Di seguito le definizioni SQL delle tabelle del sistema.

```sql
-- Definizione della tabella UTENTE
CREATE TABLE utente (
  idUtente INTEGER PRIMARY KEY,
  login VARCHAR(100) NOT NULL,
  password VARCHAR(100) NOT NULL,
  ruolo VARCHAR(100) NOT NULL
);


-- Definizione della tabella GATE
CREATE TABLE gate (
  idGate INTEGER PRIMARY KEY,
  numero INTEGER NOT NULL
);


-- Definizione della tabella PRENOTAZIONE
CREATE TABLE prenotazione (
  idPrenotazione INTEGER PRIMARY KEY,
  nome_passeggero VARCHAR(100) NOT NULL,
  numero_biglietto VARCHAR(100) NOT NULL,
  posto_assegnato VARCHAR(100) NOT NULL,
  stato_prenotazione VARCHAR(100) NOT NULL,
  idUtenteFK INTEGER NOT NULL,
  idVoloFK INTEGER NOT NULL
);


-- Definizione della tabella VOLO
CREATE TABLE volo (
  idVolo INTEGER PRIMARY KEY,
  codice VARCHAR(100) NOT NULL,
  compagnia_aerea VARCHAR(100) NOT NULL,
  data DATE NOT NULL,
  orario TIME NOT NULL,
  ritardo TIME DEFAULT '00:00:00',
  stato_volo VARCHAR(100) NOT NULL,
```

```
38    tipo_volo VARCHAR(100) NOT NULL,
39    aeroporto_origine VARCHAR(100) DEFAULT 'Napoli',
40    aeroporto_destinazione VARCHAR(100) DEFAULT 'Napoli',
41    idGateFK INTEGER
42  );
43
44
45  -- Definizione della tabella GESTIONE (Tabella di associazione N:M
        tra UTENTE e VOLO)
46  CREATE TABLE gestione (
47    idUtente INTEGER,
48    idVolo INTEGER,
49    PRIMARY KEY (idUtente, idVolo)
50  );
```

6.2 Definizione delle Sequenze

Di seguito le sequenze utilizzate per la generazione automatica degli ID delle chiavi primarie.

```
-- Sequenza per l'ID utente
CREATE SEQUENCE SEQ_UTENTE
START WITH 1
MINVALUE 1
INCREMENT BY 1;


-- Sequenza per l'ID gate
CREATE SEQUENCE SEQ_GATE
START WITH 1
MINVALUE 1
INCREMENT BY 1;


-- Sequenza per l'ID prenotazione
CREATE SEQUENCE SEQ_PRENOTAZIONE
START WITH 1
MINVALUE 1
INCREMENT BY 1;


-- Sequenza per l'ID volo
CREATE SEQUENCE SEQ_VOLO
START WITH 1
MINVALUE 1
INCREMENT BY 1;
```

6.3 Definizione dei Vincoli

Di seguito i vincoli applicati alle tabelle.

```
-- Vincolo di unicità per il login dell'utente
ALTER TABLE utente
ADD CONSTRAINT UNQ_Utente_Login UNIQUE (login);


-- Vincolo di controllo per il ruolo dell'utente
ALTER TABLE utente
ADD CONSTRAINT CHK_Utente_Ruolo CHECK (ruolo IN ('Generico', '
    Amministratore'));


-- Vincolo di controllo per il numero del gate
ALTER TABLE gate
ADD CONSTRAINT CHK_Gate_Numero CHECK (numero > 0);


-- Vincolo di unicità per il numero del biglietto della
    prenotazione
ALTER TABLE prenotazione
ADD CONSTRAINT UNQ_Prenotazione_NumeroBiglietto UNIQUE (
    numero_biglietto);


-- Vincolo di controllo per lo stato della prenotazione
ALTER TABLE prenotazione
ADD CONSTRAINT CHK_Prenotazione_Stato CHECK (stato_prenotazione IN
    ('Confermata', 'In Attesa', 'Cancellata'));


-- Vincolo di chiave esterna per collegare la prenotazione all'
    utente
ALTER TABLE prenotazione
ADD CONSTRAINT FK_Prenotazione_Utente FOREIGN KEY (idUtenteFK)
    REFERENCES utente(idUtente) ON DELETE CASCADE;


-- Vincolo di chiave esterna per collegare la prenotazione al volo
ALTER TABLE prenotazione
ADD CONSTRAINT FK_Prenotazione_Volo FOREIGN KEY (idVoloFK)
    REFERENCES volo(idVolo) ON DELETE CASCADE;


-- Vincolo di unicità per il codice del volo
ALTER TABLE volo
ADD CONSTRAINT UNQ_Volo_Codice UNIQUE (codice);


```

```
42 -- Vincolo di controllo per il ritardo del volo
43 ALTER TABLE volo
44 ADD CONSTRAINT CHK_Volo_Ritardo CHECK (ritardo >= '00:00:00');
45
46
47 -- Vincolo di controllo per lo stato del volo
48 ALTER TABLE volo
49 ADD CONSTRAINT CHK_Volo_Stato CHECK (stato_volo IN ('Programmato',
       'In Ritardo', 'Atterrato', 'Cancellato'));
50
51
52 -- Vincolo di controllo per il tipo di volo
53 ALTER TABLE volo
54 ADD CONSTRAINT CHK_Volo_Tipo CHECK (tipo_volo IN ('Partenza', '
       Arrivo'));
55
56
57 -- Vincolo di controllo per assicurare che aeroporto di origine e
       destinazione siano diversi
58 ALTER TABLE volo
59 ADD CONSTRAINT CHK_Volo_AeroportiDiversi CHECK (aeroporto_origine
       <> aeroporto_destinazione);
60
61
62 -- Vincolo di chiave esterna per collegare il volo al gate
63 ALTER TABLE volo
64 ADD CONSTRAINT FK_Volo_Gate FOREIGN KEY (idGateFK) REFERENCES gate(
       idGate) ON DELETE SET NULL;
65
66
67 -- Vincolo di chiave esterna per collegare la tabella GESTIONE all'
       utente
68 ALTER TABLE gestione
69 ADD CONSTRAINT FK_Gestione_Utente FOREIGN KEY (idUtente) REFERENCES
        utente(idUtente) ON DELETE CASCADE;
70
71
72 -- Vincolo di chiave esterna per collegare la tabella GESTIONE al
       volo
73 ALTER TABLE gestione
74 ADD CONSTRAINT FK_Gestione_Volo FOREIGN KEY (idVolo) REFERENCES
       volo(idVolo) ON DELETE CASCADE;
```

6.4 Definizione dei Trigger

```
-- Trigger per l'auto-incremento dell'ID utente tramite funzione
   generica
CREATE TRIGGER Trigger_Sequenza_Utente
BEFORE INSERT ON utente
FOR EACH ROW
EXECUTE FUNCTION Genera_ID_Sequenziale('SEQ_UTENTE');


-- Trigger per l'auto-incremento dell'ID gate tramite funzione
   generica
CREATE TRIGGER Trigger_Sequenza_Gate
BEFORE INSERT ON gate
FOR EACH ROW
EXECUTE FUNCTION Genera_ID_Sequenziale('SEQ_GATE');


-- Trigger per l'auto-incremento dell'ID prenotazione tramite
   funzione generica
CREATE TRIGGER Trigger_Sequenza_Prenotazione
BEFORE INSERT ON prenotazione
FOR EACH ROW
EXECUTE FUNCTION Genera_ID_Sequenziale('SEQ_PRENOTAZIONE');


-- Trigger per l'auto-incremento dell'ID volo tramite funzione
   generica
CREATE TRIGGER Trigger_Sequenza_Volo
BEFORE INSERT ON volo
FOR EACH ROW
EXECUTE FUNCTION Genera_ID_Sequenziale('SEQ_VOLO');


-- Trigger che attiva la funzione prima dell'inserimento o
   aggiornamento di un volo
CREATE TRIGGER Trigger_Check_Gate_Volo_Partenza
BEFORE INSERT OR UPDATE OF tipo_volo, idGateFK ON volo
FOR EACH ROW
EXECUTE FUNCTION Check_Gate_Volo_Partenza();


-- Trigger che si attiva dopo l'aggiornamento dello stato del volo
CREATE TRIGGER Trigger_Aggiorna_Prenotazioni_Volo_Cancellato
AFTER UPDATE OF stato_volo ON volo
FOR EACH ROW
EXECUTE FUNCTION Aggiorna_Prenotazioni_Volo_Cancellato();



```

```
45 -- Trigger che si attiva prima di ogni INSERT o UPDATE sulla
      tabella PRENOTAZIONE
46 CREATE TRIGGER Trigger_Check_Posto_Unico_Per_Volo
47 BEFORE INSERT OR UPDATE ON prenotazione
48 FOR EACH ROW
49 EXECUTE FUNCTION Check_Posto_Unico_Per_Volo();
50
51
52 -- Trigger che si attiva prima di ogni INSERT o UPDATE sulla
      tabella PRENOTAZIONE
53 CREATE TRIGGER Trigger_Check_Formato_Numero_Biglietto
54 BEFORE INSERT OR UPDATE ON prenotazione
55 FOR EACH ROW
56 EXECUTE FUNCTION Check_Formato_Numero_Biglietto();
57
58
59 -- Trigger che si attiva prima dell'inserimento in GESTIONE
60 CREATE TRIGGER Trigger_Check_Ruolo_Amministratore_Gestione
61 BEFORE INSERT ON gestione
62 FOR EACH ROW
63 EXECUTE FUNCTION Check_Ruolo_Amministratore_Gestione();
```

6.5 Definizione delle Funzioni

```
-- Funzione generica per la generazione automatica degli ID sequenziali
CREATE OR REPLACE FUNCTION Genera_ID_Sequenziale(nome_sequenza TEXT)
RETURNS TRIGGER AS $$
BEGIN
  CASE TG_TABLE_NAME
    WHEN 'utente' THEN
      IF NEW.idUtente IS NULL THEN
        NEW.idUtente := NEXTVAL(nome_sequenza);
      END IF;
    WHEN 'gate' THEN
      IF NEW.idGate IS NULL THEN
        NEW.idGate := NEXTVAL(nome_sequenza);
      END IF;
    WHEN 'prenotazione' THEN
      IF NEW.idPrenotazione IS NULL THEN
        NEW.idPrenotazione := NEXTVAL(nome_sequenza);
      END IF;
    WHEN 'volo' THEN
      IF NEW.idVolo IS NULL THEN
        NEW.idVolo := NEXTVAL(nome_sequenza);
      END IF;
    ELSE
        RAISE EXCEPTION 'Errore: la tabella % non è gestita dalla funzione Genera_ID_Sequenziale.', TG_TABLE_NAME;
  END CASE;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;



-- Funzione per verificare che un volo di tipo 'Partenza' abbia un gate assegnato
CREATE OR REPLACE FUNCTION Check_Gate_Volo_Partenza()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.tipo_volo = 'Partenza' AND NEW.idGateFK IS NULL THEN
        RAISE EXCEPTION 'Un volo di tipo "Partenza" deve avere un GATE assegnato.';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;


```

```
45
46  -- Funzione per aggiornare lo stato delle prenotazioni quando il
        volo viene cancellato
47  CREATE OR REPLACE FUNCTION Aggiorna_Prenotazioni_Volo_Cancellato()
48  RETURNS TRIGGER AS $$
49  BEGIN
50      IF OLD.stato_volo <> 'Cancellato' AND NEW.stato_volo = '
            Cancellato' THEN
51          UPDATE prenotazione
52          SET stato_prenotazione = 'Cancellata'
53          WHERE idVoloFK = NEW.idVolo AND stato_prenotazione <> '
                Cancellata';
54      END IF;
55      RETURN NEW;
56  END;
57  $$ LANGUAGE plpgsql;
58
59
60
61
62  -- Funzione per garantire che il posto assegnato sia unico per un
        dato volo
63  CREATE OR REPLACE FUNCTION Check_Posto_Unico_Per_Volo()
64  RETURNS TRIGGER AS $$
65  BEGIN
66      IF EXISTS (
67          SELECT *
68          FROM prenotazione
69          WHERE idVoloFK = NEW.idVoloFK
70            AND posto_assegnato = NEW.posto_assegnato
71            AND (TG_OP = 'INSERT' OR idPrenotazione <> NEW.
                    idPrenotazione)
72      ) THEN
73          RAISE EXCEPTION 'Il posto % è già assegnato per il volo %.'
                , NEW.posto_assegnato, NEW.idVoloFK;
74      END IF;
75      RETURN NEW;
76  END;
77  $$ LANGUAGE plpgsql;
78
79
80
81  -- Funzione per controllare che il formato del numero del biglietto
        sia valido
82  CREATE OR REPLACE FUNCTION Check_Formato_Numero_Biglietto()
83  RETURNS TRIGGER AS $$
84  BEGIN
85      IF NOT NEW.numero_biglietto ~ '^[A-Z]{2}[0-9]{4}$' THEN
86          RAISE EXCEPTION 'Il formato del numero di biglietto non è
                valido. Deve essere: 2 lettere e 4 cifre (es. AB1234).';
87      END IF;
```

```
88      RETURN NEW;
89  END;
90  $$ LANGUAGE plpgsql;
91
92
93
94
95  -- Funzione per verificare che solo gli amministratori possano
        gestire voli
96  CREATE OR REPLACE FUNCTION Check_Ruolo_Amministratore_Gestione()
97  RETURNS TRIGGER AS $$
98  DECLARE
99      utente_ruolo VARCHAR(100);
100 BEGIN
101     SELECT ruolo INTO utente_ruolo
102     FROM utente
103     WHERE idUtente = NEW.idUtente;
104
105     IF utente_ruolo IS DISTINCT FROM 'Amministratore' THEN
106         RAISE EXCEPTION 'Solo gli utenti con ruolo ''Amministratore
                '' possono essere assegnati alla gestione dei voli.';
107     END IF;
108     RETURN NEW;
109 END;
110 $$ LANGUAGE plpgsql;
```